

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Міністерство освіти і науки України

Кваліфікаційна наукова праця

на правах рукопису

УДК 004.033

МАЗУРОК ВАЛЕНТИН ОЛЕГОВИЧ

ДИСЕРТАЦІЯ

**СТВОРЕННЯ ЗАСОБІВ ЗАХИСТУ ОПЕРАТИВНОЇ ПАМ'ЯТІ ВІД АТАК
ТИПУ ROWHAMMER**

12 – Інформаційні технології

125 – Кібербезпека та захист інформації

Подається на здобуття наукового ступеня доктора філософії
Дисертація містить результати власних досліджень. Використання ідей,
результатів і текстів інших авторів мають посилання на відповідне джерело

_____ В. О. Мазурок

Науковий керівник Луценко Володимир Миколайович, доцент, кандидат
технічних наук, старший науковий співробітник

Київ – 2025

АНОТАЦІЯ

Мазурок В. О. Створення засобів захисту оперативної пам'яті від атак типу RowHammer. – Кваліфікаційна праця на правах рукопису. Дисертація на здобуття наукового ступеня доктора філософії з галузі знань 12 – Інформаційні технології за спеціальністю 125 – Кібербезпека та захист інформації. – Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2025

Пам'ять відіграє ключову роль у функціонуванні сучасних обчислювальних систем, забезпечуючи зберігання і обробку даних, необхідних для стабільної роботи та високої продуктивності. Вона є основою для реалізації швидкого доступу до інформації, підтримки складних алгоритмів маршрутизації, обробки пакетів та роботи серверів у режимі реального часу. Однак поряд із її важливістю постають і значні загрози, пов'язані з уразливістю пам'яті до атак. Такі ризики, зокрема, можуть призводити до порушення цілісності даних, зниження продуктивності обчислювальної системи, або навіть витоку конфіденційної інформації. Уразливості пам'яті створюють потенційні точки доступу для злоумисників, що ставить під загрозу безпеку всієї мережевої інфраструктури.

Ізоляція пам'яті є однією з основних властивістю надійної та безпечної обчислювальної системи. Доступ до адреси пам'яті не повинен мати небажаних побічних ефектів для даних, що зберігаються в інших адресах. Однак у міру того, як технологічний процес зменшується, мікросхеми пам'яті стають більш уразливими до перешкод та наведень, або навіть явищ, коли різні комірки пам'яті заважають роботі одна одній.

Дисертаційну роботу присвячено розробці методів детектування та захисту від атак на пам'ять в обчислювальних системах, зокрема від RowHammer атак.

В роботі було проведено аналіз та сформовано модель оперативної пам'яті DRAM для того щоб формувати працюючі в реальному світі математичні моделі. Це дозволить створювати кращі схеми захисту обчислювальних систем при зменшенні фізичних розмірів пристроїв та їх підсистем.

Після цього було представлено основні вектори атак на пам'ять і показано основні недоліки, які працюють завдяки використанню описаних вище нереалістичних припущень про модель роботи пам'яті на нанорівні.

Завдяки зазначеним вище даним було продемонстровано, які захисні системи існують на ринку і чому вони можуть бути не ефективними з боку реального захисту даних, чи з боку енергоспоживання.

За результатами дослідження зв'язків між виявленими загрозами та вразливостями, які притаманні сучасним обчислювальним системам, було сформовано потребу в розробці більш сучасних методів захисту як в апаратній так і в програмній площині.

Для підтвердження теоретичних даних щодо недосконалості наявних захисних систем було створено тестувальну платформу, що дозволить виявити і систематизувати знайдені прогалини в захисті існуючих систем пам'яті. Також це дозволить виділити основні тренди розвитку в різних розробників та компаній і підтвердити функціональність отриманої математичної моделі пам'яті.

Завдяки математичній моделі і наявним аналітичним даним щодо тестування реальних зразків пам'яті типу DRAM було почато розробку програмних та апаратних рішень що дозволять мінімізувати вразливість вже існуючих систем. Також завдяки аналізу існуючих трендів можна буде спрогнозувати який саме рівень захищеності буде потрібен для наступних ітерації систем збереження інформації.

Розробка систем була поділена на три підгрупи – апаратні програмні та комбіновані для створення систем захисту зі своїми перевагами та недоліками. Розробка кількох систем також дасть змогу виділити найкращу чи запропонувати комбінації захисних механізмів що будуть переважати над існуючими як в плані захисту наявної інформації. Також це дозволить сконцентруватись на розробці рішення що мінімізуватимуть додаткові витрати часу, грошей та енергії на захисні механізми. За результатами даної роботи було сформовано моделі, що дозволяють захищати пам'ять за допомогою програмних та апаратних засобів. Зокрема модель детектора на основі машинного навчання здатна виявляти до 99.5% атак на пам'ять типу RowHammer, а модель на основі лічильників доступу до 99.7%.

Після цих етапів було представлено технічні приклади реалізації захисних механізмів на існуючих системах. Показано зменшення кількості успішних атак та продемонстровано повний детальний алгоритм запровадження і масштабування даної системи захисту.

Метою дисертаційної роботи є розв'язання актуальної технічної задачі захисту пам'яті комп'ютерних систем від атак, що здатні викрасти чи пошкодити важливі дані. Атаки RowHammer у сучасних системах майже неможливо відрізнити від звичайної роботи DRAM, тому їх виявлення є серйозним викликом саме по собі. У той же час методи виявлення та запобігання не повинні спричиняти погіршення продуктивності або приводити до значних витрат енергії.

Об'єкт: загрози і вразливості у пам'яті комп'ютерних систем.

Предмет: методи та моделі захисту пам'яті комп'ютерних систем.

У дисертаційному дослідженні отримані такі наукові результати:

- Вперше розроблено методологію збору даних щодо захисту від атак типу RowHammer нових систем пам'яті DDR5.

- Вперше розроблено модель захисту пам'яті DRAM систем на основі машинного навчання, що працює в реальному часі.

- Вперше розроблено модель захисту пам'яті DRAM систем на основі лічильників доступу, що не має вразливостей нерівномірного оновлення.

- Удосконалено методологію тестування вразливостей чіпів DRAM DDR5 щодо атак типу RowHammer шляхом розробки апаратно-програмного комплексу тестового обладнання.

Всі теоретичні і практичні результати дисертаційної роботи у повній мірі висвітлено у статтях, опублікованих у фахових вітчизняних наукових виданнях, що входять до відповідного встановленого переліку. Виконано їх належну апробацію на міжнародних та всеукраїнських наукових конференціях.

У дисертаційному дослідженні розв'язана задача детектування атак на пам'ять типу DDR за допомогою кількох підходів: технічного та програмного. Проаналізовано основні причини виникнення вразливості типу RowHammer та показано недоліки представлених в сучасних системах пам'яті захисних механізмів. Основним чинником визначено електромагнітне зв'язування та накопичення паразитних зарядів на субстраті електричних елементів пам'яті.

Зібрано в базу даних інформацію про захист нових моделей чіпів пам'яті DDR4 та DDR5 від атак типу RowHammer та представлено на графіках для розуміння трендів розвитку вразливості. Проаналізовано зв'язки та схожості сигнатур доступу під час атак в різних типах пам'яті для формування правил майбутніх захисних механізмів. Побудовано математичну модель представлення пам'яті типу DRAM, що дозволяє аналізувати атаки на окремі комірки пам'яті. Розроблено основні алгоритми для побудови структурної моделі системи взаємозалежностей загроз з боку атакуючих рядків пам'яті та вразливостей з боку невідповідності частоти оновлення рядків жертв.

У роботі структуровано інформацію про загрози та вразливості систем оперативної пам'яті DRAM з боку атак, зокрема типу RowHammer та те, що на

даному етапі їх неможливо детектувати. Описано основні метрики оцінки вразливості чіпів пам'яті та їх схильності до зміни бітів в наслідку атак типу RowHammer. Визначено, що внаслідок подібних атак дані можуть бути скомпрометовані або втрачені без виявлення мікропрограмами DRAM чи антивірусними програмами. На основі даних зібраних з різних моделей DRAM, які відображають всі лінійки та типи сучасної оперативної пам'яті, розроблено методи виявлення та протидії атакам на пам'ять.

Продемонстровано систему детектування атаки RowHammer на основі трьох нейромереж, що навчалися на реальних даних типів LSMP, MLP та CNN. Всі три моделі продемонстрували точність виявлення атаки в більше ніж 95%, при цьому не сповільнюючи систему. В результаті було обрано для реалізації тип мереж MLP оскільки вона має найменшу зайняту пам'ять та найменший час проходження даних до результату аналізу. Також продемонстровано підхід до детектування, що базується на представленій математичній моделі пам'яті. Даний підхід представляє собою модель, що оперує частотою активацій рядка як метрикою щодо визначення його як атакуючого. Даний метод демонструє найвищий рівень точності в 99.7% та значно швидший за нейромережевий підхід, але при цьому дуже залежить від конкретних параметрів кожної плати пам'яті.

Показано можливості практичного застосування розроблених методів захисту пам'яті з урахуванням вимог до енерговикористання та накладних витрат до імплементації у сучасних комп'ютерних системах. Здійснено порівняння результатів, отриманих при запуску на тестовому середовищі та на реальній системі, з урахуванням зовнішніх факторів шумів та паралельно працюючих процесів у тестованій системі.

За матеріалами дисертації опубліковано 9 робіт, з яких 4 – це статті у журналах і збірниках наукових праць, що входять до переліку фахових видань,

затверджених МОН України за спеціальністю дисертації та 5 – публікації у матеріалах конференцій (у тому числі, міжнародних)

Ключові слова: кібербезпека, інформаційна безпека, кібератаки, бази даних, штучний інтелект, гіпервізор, вибірка даних, конфіденційні дані, DDR, RowHammer, приховані засоби отримання інформації, програмні системи, безпека продукту.

ABSTRACT

Mazurok V. O. Creating methods of protection of DRAM from RowHammer attacks. – Qualifying scientific work, the manuscript. PhD thesis in the field of knowledge 12 Information Technology in specialty 125 Cyber security and information protection. – National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Kyiv, 2025

Memory plays a key role in the functioning of modern computational systems, providing storage and processing of data necessary for stable operation and high performance. It is the basis for implementing fast access to information, supporting complex routing algorithms, packet processing and real-time server operation. However, along with its importance, significant threats arise associated with the vulnerability of memory to attacks. Such risks, in particular, can lead to data integrity violations, reduced computational system performance, or even leakage of confidential information. Memory vulnerabilities create potential access points for attackers, which jeopardizes the security of the entire network infrastructure.

Memory isolation is one of the main properties of a reliable and secure computing system. Access to a memory address should not have unwanted side effects for data stored at other addresses. However, as the technological process decreases, memory chips become more vulnerable to interference and interference, or even phenomena where different memory cells interfere with each other.

The dissertation is devoted to the development of detection and protection methods against memory attacks in computational systems, from RowHammer in particular.

The work analyzed and formed a model of DRAM memory in order to form working mathematical models in the real world. This will allow creating better protection models of computation systems, while reducing the physical size of devices and their subsystems.

After that, the main vectors of attacks on memory were presented and the main shortcomings that work due to the use of the unrealistic assumptions described above about the model of memory operation at the nano level were shown.

Thanks to the data shown above, it was demonstrated what protection systems exist on the market and why they may not be effective in terms of real data protection or energy consumption.

According to the results of the study of the connections between the identified threats and vulnerabilities inherent in modern network systems, the need was formed to develop more modern protection methods both in the hardware and software planes.

To confirm the theoretical data on the imperfection of existing protection systems, a testing platform was created that will allow identifying and systematizing the gaps found in the protection of existing memory systems. This will also allow identifying the main development trends in different developers and companies and confirming the functionality of the resulting mathematical model of memory.

Thanks to the mathematical model and the available analytical data on testing real samples of DRAM memory, the development of software and hardware solutions was started that will minimize the vulnerability of existing systems. Also, thanks to the analysis of existing trends, it will be possible to predict what level of security will be required for the next iterations of information storage systems.

The development of systems was divided into three subgroups - hardware, software and combined to create protection systems with their own advantages and disadvantages. The development of several systems will also allow us to highlight the best or offer combinations of protective mechanisms that will prevail over existing ones in terms of protecting existing information. This will also allow us to focus on developing solutions that will minimize additional time, money and energy spent on protective mechanisms. Based on the results of this work, models were

formed that allow protecting memory using software and hardware. In particular, the detector model based on machine learning is capable of detecting up to 99.5% of RowHammer-type memory attacks, and the model based on access counters is up to 99.7%.

After these stages, technical examples of implementing protective mechanisms on existing systems were presented. A reduction in the number of successful attacks was shown and a full detailed algorithm for implementing and scaling this protection system was demonstrated.

The aim of the dissertation is to solve the current technical problem of protecting the memory of computer systems from attacks that can steal or damage faulty data. RowHammer attacks in current systems are indistinguishable from regular DRAM operation, so detecting it is a big challenge in itself. In the same time mitigation technics must not cause performance degradation of significant energy cost.

Object: threats and vulnerabilities in the memory of computer systems.

Subject: models and methods for protecting memory of computer systems.

The following scientific results were obtained in the dissertation research:

- Developed a methodology for collecting data to protect against RowHammer attacks on new DDR4 and DDR5 memory systems for the first time.
- Developed a real-time machine learning-based DRAM protection model for the first time.
- Developed a counter-based DRAM protection model for the first time, free of non-uniform update vulnerabilities.
- Improved methodology for testing vulnerabilities of DDR5 DRAM chips against RowHammer attacks by developing a hardware and software complex of new test equipment.

All theoretical and practical results of the dissertation work are fully covered in articles published in professional domestic scientific publications that are included in the corresponding established list. Their proper testing was carried out at international and all-Ukrainian scientific conferences.

The dissertation study solves the problem of detecting attacks on DDR memory using several approaches: technical and software. The main causes of RowHammer vulnerability are analyzed and the shortcomings of the protective mechanisms presented in modern memory systems are shown. The main factor is identified as electromagnetic coupling and the accumulation of parasitic charges on the substrate of electrical memory elements.

Data on the protection of new models of DDR4 and DDR5 memory chips from RowHammer attacks are collected to a database and presented in graphs to understand the trends in vulnerability development. The connections and similarities of access signatures during attacks in different types of memory are analyzed to form rules for future protective mechanisms. A mathematical model of DRAM memory representation is built, which allows analyzing attacks on individual memory cells. The main algorithms are developed to build a structural model of the system of interdependencies of threats from attacking memory rows and vulnerabilities from the mismatch of the refresh rate of the victim rows.

The paper presents information on the threats and vulnerabilities of DRAM memory systems from attacks, in particular RowHammer attacks, and the fact that they cannot be detected at this stage. The main metrics for assessing the vulnerability of memory chips and their tendency to change bits as a result of RowHammer attacks are described. It is determined that because of such attacks, data can be compromised or lost without being detected by DRAM firmware or antivirus programs. Based on data collected from various DRAM models, which reflect all lines and types of modern RAM, methods for detecting and countering memory attacks have been developed.

A RowHammer attack detection system based on three neural networks trained on real data of the LSMP, MLP, and CNN types has been demonstrated. All three demonstrated an attack detection accuracy of more than 95%, without slowing down the system. As a result, the MLP network type was chosen for implementation, since it has the smallest memory footprint and the shortest data transfer time to the analysis result. An approach to detection based on the presented mathematical model of memory is also demonstrated. This approach is a model that operates on the frequency of activations of a row as a metric for determining it as an attacker. This method demonstrates a lower level of accuracy of 90%, but at the same time it can be implemented on the controller side and is much faster than the neural network approach.

The possibilities of practical application of the developed memory protection methods are shown, taking into account the requirements for energy consumption and overhead for implementation in modern computer systems. A comparison of the results obtained when running in a test environment and on a real system is made, taking into account external noise factors and parallel running processes in the tested system.

Based on the materials of the dissertation, 9 works have been published, of which 4 are articles in journals and collections of scientific works included in the list of professional publications approved by the Ministry of Education and Science of Ukraine in the specialty of the dissertation and 5 are publications in conference proceedings (including international ones)

Keywords: cybersecurity, information security, cyberattacks, databases, artificial intelligence, hypervisor, data mining, sensitive data, DDR, RowHammer, covert means of obtaining information, software systems, product security.

List of principal publications of the applicant:

1. Mazurok, V., & Lutsenko, V. «An analytical overview and trend analysis of RowHammer vulnerabilities for various DRAM vendors.» *Social Development and Security*, 14(3), P 238-244. <https://doi.org/10.33445/sds.2024.14.3.16> (date of access 26.12.2024)

У роботі здобувачем показано процес створення тестової платформи для тестування чіпів DRAM на предмет вразливості типу RowHammer та показано дані тестування чіпів типу DDR3 та DDR4. Дані сформовано в графіки та показано прогнозовані тренди розвитку вразливості.

2. Mazurok, V., & Lutsenko, V. «Enhancing Row-Sampling-Based RowHammer defense methods with Machine Learning approach» *Theoretical and Applied Cyber Security* Vol. 6 No. 2 P 31-35 <https://doi.org/10.20535/tacs.2664-29132024.2.319008> (date of access: 26.12.2024)

У роботі здобувачем наведено розроблений алгоритм детектування атак типу RowHammer за допомогою нейронних мереж. Представлено дані навчання та тестування трьох видів мереж та показано їх недоліки та переваги.

3. Mazurok, V., & Lutsenko, V. «Improving the effectiveness of Row-Sampling methods to protect against Row-Hammer attacks». *Social Development and Security*, 14(6), P 61-67. <https://doi.org/10.33445/sds.2024.14.6.7> (date of access: 26.12.2024)

У роботі здобувачем представлено розроблену математичну модель представлення пам'яті DRAM та показано недоліки методу детектування заснованого на вибірці рядків. Показано доповнення та математичний апарат що покращує даний алгоритм детектування атак типу RowHammer за рахунок врахування неоднорідності пам'яті та радіусу дії збурень.

4. Mazurok, V., & Lutsenko, V. Detecting RowHammer attacks using frequency arrays. *Social Development and Security*. 2025. 15(1). P. 42 – 49. <https://doi.org/10.33445/sds.2024.15.1.7> (date of access: 05.03.2025)

У роботі здобувачем представлено розроблену математичну модель захисту пам'яті DRAM на основі лічильників доступу. Показано доповнення та математичний апарат що показує відсутність у даного алгоритму вразливості нерівномірного оновлення та високий рівень детектування атак типу RowHammer.

5. Мазурок В., Луценко В. «Аналітичний огляд атаки RowHammer на сучасну пам'ять» Міжнародна науково-практична інтернет-конференція «Trends of modern science and practice» 08 листопада 2022

У дослідженні здобувачем продемонстровано процес розробки тестової платформи для перевірки чіпів DRAM на наявність вразливості типу RowHammer. Проведено тестування чіпів пам'яті стандартів DDR3 та DDR4, результати якого представлені у вигляді графіків, що ілюструють прогнозовані тенденції у розвитку цієї вразливості.

6. Мазурок В., Луценко В. «Практична ефективність RowSampling для захисту від RowHammer» Міжнародна науково-практична інтернет-конференція «Розвиток освіти, науки та бізнесу: результати 2022» 19 грудня 2022

У дослідженні здобувачем розроблено математичну модель для опису пам'яті DRAM, а також проаналізовано недоліки методу детектування вразливостей, заснованого на вибірці рядків. Зокрема, було відзначено, що цей підхід потребує значної оптимізації для забезпечення більшої точності та ефективності в умовах сучасних високопродуктивних систем.

7. Mazurok, V., & Lutsenko, V. «Mathematical model of DRAM for RowHammer protection» XI Міжнародна науково-практична конференція «Integration of science as a mechanism of effective development», 28 листопада 2023 р.

У дослідженні здобувачем представлено математичну модель для опису пам'яті DRAM, яка стала основою для розробки математичного апарату. Цей апарат дозволяє враховувати неоднорідність структури пам'яті та

радіус дії збурень, що створює базу для створення ефективних алгоритмів детектування атак типу RowHammer.

8. Mazurok, V., & Lutsenko, V. «Discovering DRAM access patterns for machine learning sample finding» V Міжнародна науково-практична конференція «Modern technologies and processes of implementation of new methods», 06 лютого 2024 р.,

У роботі здобувачем показано основні схожості моделі доступу до пам'яті під час атак типу RowHammer. Це ґрунтовне дослідження показує головні елементи, які можуть допомогти детектувати атаку і які можна використовувати для навчання нейромереж та моделей.

9. Mazurok, V., & Lutsenko, V. «Machine learning methods of RowHammer mitigation» XXIV Міжнародна науково-практична конференція «Technologies of scientists and implementation of modern methods», 18-21 червня 2024 р.

У роботі здобувачем продемонстровано навчання моделей машинного навчання на даних доступу до пам'яті під час атак типу RowHammer. Показано ефективність детектування кількома моделями та оптимізацію зайнятого ними простору відносно знайдених даних.

ЗМІСТ

АНОТАЦІЯ.....	2
ABSTRACT.....	8
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	19
ВСТУП.....	21
РОЗДІЛ 1 АКТУАЛЬНІСТЬ ПРОБЛЕМИ АТАК НА ПАМ'ЯТЬ ТА ОСНОВНІ ПІДХОДИ ДО ЇЇ РОЗВ'ЯЗАННЯ.....	28
1.1 Будова пам'яті персонального комп'ютера.....	28
1.1.1 Основні поняття.....	30
1.1.2 Архітектура та робота DRAM.....	32
1.1.3 Математична модель представлення пам'яті	43
1.2 Основні вектори атак на пам'ять	45
1.2.1 Основні поняття.....	46
1.2.2 Атаки побічними каналами	49
1.2.3 Атаки при завантаженні	54
1.2.4 Атака типу RowHammer.....	58
1.3 Сучасні підходи до захисту пам'яті.....	69
1.3.1 Основні поняття.....	70
1.3.2 Апаратні рішення: оновлення DRAM, ECC	71
1.3.3 Програмні методи: алгоритми перевірки й обмеження доступу	74
1.3.4 Переваги та недоліки основних методів захисту пам'яті	78
Висновки до розділу 1	81
РОЗДІЛ 2 ТЕСТУВАННЯ НАЯВНИХ ВБУДОВАНИХ СИСТЕМ ЗАХИСТУ ПАМ'ЯТІ ТА ОЦІНКА ВРАЗЛИВОСТЕЙ.....	83

2.1 Створення платформи уніфікованого тестування захисту пам'яті.....	83
2.1.1 Побудова універсальної програмної основи.....	84
2.1.2 Вибір критеріїв для оцінювання ефективності захисту DRAM.....	92
2.2 Збір та структуризація результатів тестування.....	97
2.2.1 Стан захисту стокових чіпів пам'яті.....	99
2.2.2 Тестування додаткових методів захисту.....	108
2.3 Виявлення обмежень та вразливостей в наявних апаратних і програмних рішеннях	110
Висновки до розділу 2.....	116
РОЗДІЛ 3. РОЗРОБКА ПОКРАЩЕНИХ СИСТЕМ ЗАХИСТУ ПАМ'ЯТІ ТА ОЦІНКА ЇХ РОБОТИ.....	118
3.1 Розробка рішень оптимізації захисту на основі вибірки рядків	118
3.2 Зменшення витрати пам'яті в системах на основі лічильників	124
3.2.1 Деталізація підрахунку доступу на рівні банку та рангу	127
3.2.2 Перехід до лічильників рівня рангу.....	130
3.3 Захисний механізм на основі машинного навчання.....	137
3.3.1 Вибір моделі машинного навчання.....	138
3.3.1 Використання для захисту та порівняння результатів мереж	151
3.3.2 Порівняння зайнятої пам'яті	159
3.4 Захист на основі частотного масиву	163
Висновки до розділу 3.....	170
РОЗДІЛ 4. ПРАКТИЧНЕ ЗАСТОСУВАННЯ РОЗРОБЛЕНИХ ЗАСОБІВ ЗАХИСТУ ОБЧИСЛЮВАЛЬНОЇ СИСТЕМИ ВІД АТАК НА ПАМ'ЯТЬ	173
4.1 Формування основних характеристик тестованої обчислювальної системи.....	173

4.2 Імплементація захисних механізмів	179
4.3 Тестування отриманих захисних механізмів	186
Висновки до розділу 4	192
ВИСНОВКИ	194
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	197

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACT – Activate command

ALU – Arithmetic Logic Unit

BL – Bit Line

CAM – Content-addressable Memory

CBF – Counting Bloom Filter

CNN – Convolutional Neural Network

CSL – Column Selection Logic

CPU – Central Processing Unit

DDR – Double Data Rate (memory)

DIMM – Dual In-line Memory Module

DMA – Direct Memory Access

DRAM – Dynamic Random Access Memory

ECC – Error-Correcting Code

EMC – External Memory Controller

FN – False Negative

FP – False Positive

FPGA – Field-Programmable Gate Array

HPC – Hardware Performance Counter

LLC – Last-Level Cache

LPDDR – Low Power DDR memory

LSTM – Long Short-Term Memory

MAT – Memory Array Tile

ML – Machine Learning

MLP – Multi-Layer Perceptron

MTJ – Magnetic Tunnel Junction

NVM – Non-Volatile Memory

PRE – Precharge command

PRNG – Pseudo-Random Number Generator

RAM – Random Access Memory

ROM – read-only memory

RB – Row Buffer

REF – Refresh command

RFM – Refresh Management command

RTL – Register-Transfer Level

SA – Sense Amplifier

WL – Word Line

БД – база даних

ЦП – Центральний Процесор

ПК – Персональний Комп'ютер

ОС – Операційна Система

ВСТУП

Актуальність роботи

Пам'ять є ключовим компонентом обчислювальних систем. Вона використовується для зберігання вхідних даних, проміжних змінних і кінцевих результатів будь-якої програми. З роками, дотримуючись закону Мура, обчислювальні системи стають експоненціально потужнішими, дозволяючи комп'ютерам працювати швидше, із зростаючою кількістю все більш складних програм, що виконуються паралельно. Робота такої складності вимагає, щоб пам'ять ставала швидшою щодня, з постійно зростаючою ємністю для зберігання великої кількості даних, необхідних програмам під час виконання. Найбільш популярною на сьогодні технологією для зберігання корисної інформації процесів та програм є динамічна пам'ять з довільним доступом (DRAM). Ця технологія використовує конденсатори для зберігання даних: заряд кожного конденсатора визначає значення біта, який він зберігає. Знову ж таки, дотримуючись закону Мура, виробники DRAM щороку збільшують щільність пам'яті, зменшуючи виробничу вартість за біт і дозволяючи зберігати більше даних в тій самих фізичних межах кремнієвих кристалів.

Однак у міру того, як пам'ять стає щільнішою, конденсатори комірок пам'яті стають менше. При цьому зменшується їх максимальна ємність та заряд перемикачів. Крім того, фізичні явища та наведення, які виникають на мікрорівні та спочатку не мали впливу на поведінку пристроїв, почали спричиняти проблеми з надійністю. У 2014 році професор Кім та ін. [4] опублікував першу публічну статтю про помилки та спотворення даних в комірках DRAM, спричинені частими та прицільними активаціями потрібних рядків пам'яті. Коли рядок DRAM активується, небажані електромагнітні наведення виникають на сусідніх рядках, що в свою чергу може перешкоджати коректному запису, чи навіть спотворювати вже наявні в них дані. Конденсатори постраждалих комірок пам'яті, які за своєю конструкцією і так

втрачають заряд із часом, можуть фіксувати додаткове зменшення заряду, коли їх таке явище спостерігають. Цей додатковий витік не є вагомим і сам по собі не ставить під загрозу дані, оскільки конденсатори періодично оновлюють значення накопиченого заряду, щоб компенсувати його природний витік. Однак повторні активації сусідніх рядків можуть призвести до експоненційного накопичення вищеописаного ефекту, що в свою чергу призводить до повного дочасного розрядження конденсаторів «жертв», фактично змінюючи біти на протилежні і спотворюючи наявні в пам'яті дані.

Дослідження минулих років показали, що це фізичне явище може бути легко використано шкідливою програмою для проведення атаки під назвою RowHammer. Протягом наступних років статті та тести показали, що цю атаку можна покращити, щоб викликати серйозні проблеми в сучасних комп'ютерах, починаючи від виявлення ключів шифрування [5] для дистанційного вимкнення систем [6] або виконання підміни привілеїв із веб-переглядачів [7].

Саме тому захист від подібного роду атак є актуальною проблемою сучасних обчислювальних систем і зважаючи на вектор розвитку пам'яті, зокрема DRAM можна впевнено сказати, що це лишиться актуальним і в майбутньому.

Мета і задачі наукового дослідження. Метою дисертаційної роботи є розробка систем та підходів до захисту пам'яті обчислювальних систем від атак типу RowHammer, що дозволяє спотворювати, викрадати чи знищувати важливі для користувачів дані.

Для досягнення поставленої мети необхідно розв'язати такі завдання:

1. Створити апаратно-програмний комплекс тестового обладнання для збору інформації про актуальний рівень захисту модулів пам'яті.
2. Проаналізувати стан захисту наявних систем збереження даних від атак та вразливостей типу RowHammer.

3. Дослідити наукові напрацювання щодо теоретичних і практичних підходів до покращення захищеності пам'яті.
4. Структурувати інформацію щодо актуального стану захисних систем та розробити методологію збору даних щодо захисту нових систем пам'яті.
5. Розробити моделі захисту на основі проаналізованих напрацювань та зібраних даних про вразливість.
6. Протестувати та виділити найкращі методи для покращення захисту пам'яті обчислювальних систем від атак.

Об'єкт: загрози і вразливості пам'яті комп'ютерних систем.

Предмет: методи та моделі захисту пам'яті обчислювальних систем від атак типу RowHammer.

У ході дослідження використовувались мови програмування Python, C та VHDL.

Дослідженню проблем, пов'язаних із розробкою захисних механізмів від атак на пам'ять типу RowHammer, присвячується значна частина публікацій вітчизняних та зарубіжних вчених, серед них: Франс Л. Ф., Кім Й. К., Квонг А. А., Ліп М. Т. та інші науковці. Однак, незважаючи на існуючі підходи щодо захисту пам'яті кіберсистем, запропонований у дослідженні метод є актуальним.

Наукова новизна одержаних результатів полягає в зборі актуальної інформації щодо захисту найновіших DRAM систем та комбінуванні їх з вже існуючими підходами з удосконалення безпеки пам'яті для формування покращених методів захисту, що працюють на основі нових програмних методів. На основі проведеного дослідження представлено нові дані та висновки, а саме:

- Вперше розроблено методологію збору даних щодо захисту від атак типу RowHammer нових систем пам'яті DRAM DDR5.

- Вперше розроблено модель захисту пам'яті DRAM систем на основі машинного навчання, що працює в реальному часі.
- Вперше розроблено модель захисту пам'яті DRAM систем на основі лічильників доступу, що не має вразливостей нерівномірного оновлення.
- Удосконалено методологію тестування вразливостей чіпів DRAM DDR5 щодо атак типу RowHammer шляхом розробки апаратно-програмного комплексу тестового обладнання.

Обґрунтованість і достовірність наукових результатів забезпечується коректним застосуванням математичного апарату та відповідних обчислювальних експериментів із використанням сучасного програмного забезпечення.

Практичне значення отриманих результатів.

Отримані в дисертаційній роботі результати можуть бути застосовані в усіх типах обчислювальних систем, що використовують DRAM для захисту даних від атак типу RowHammer. Застосування запропонованих методів захисту було продемонстровано на практичному прикладі обчислювальної системи з параметрами, що найчастіше зустрічаються в комп'ютерній інфраструктурі. Всі теоретичні і практичні результати дисертаційної роботи у повній мірі висвітлено у статтях, опублікованих у фахових вітчизняних наукових виданнях, що входять до відповідного встановленого переліку; виконано їх належну апробацію на міжнародних та всеукраїнських наукових конференціях.

За матеріалами дисертації опубліковано 9 робіт, з яких 4 – це статті у журналах і збірниках наукових праць, що входять до переліку фахових видань, затверджених МОН України за спеціальністю дисертації та 5 – публікації у матеріалах конференцій (у тому числі, міжнародних).

Особистий внесок здобувача. Основні положення і результати дисертаційної роботи, що виносяться до захисту, отримані автором самостійно. Всі фахові публікації написані автором дисертації у співавторстві. Належна апробація результатів була проведена на конференціях та наведена в 8 тезах конференцій. Здобувачеві належать такі результати: У роботі [1] здобувачем показано процес створення тестової платформи для тестування чіпів DRAM на предмет вразливості типу RowHammer. Тут показано нові дані щодо вразливості, продемонстровано еволюцію захисту в старіших і новіших версіях DRAM. Особливо важливо, що в дослідженні показано дані тестування чіпів типу DDR4 та кількох DDR5, інформації щодо яких була донині відсутня. Інформацію по всім чіпам сформовано в графіки та показано прогнозовані тренди розвитку вразливості. У роботі [2] здобувачем розроблено алгоритм покращення детектування та оновлення рядків жертв атаки RowHammer. Він базується на підході ймовірнісної вибірки рядків і впливає з математичної моделі пам'яті показаної в дослідженні. Цей алгоритм використовується для зменшення загрози зміни бітів та оновлення рядків до того як атака відбудеться. У роботі [3] здобувачем запропоновано метод, захисту від атак RowHammer на основі моделей машинного навчання, який дозволяє виявляти складні патерни доступу під час атаки на пам'ять. У дослідженні показано навчання кількох моделей з подальшим їх порівнянням на швидкість і об'єм зайнятої пам'яті. Наведено приклад застосування в реальних системах на рівні драйверів для виявлення деяких потенційних загроз та їх комбінацій.

Апробація результатів дисертації. Результати та основні положення роботи подавалися та обговорювалися на:

1. Mazurok, V., & Lutsenko, V. «An analytical overview and trend analysis of RowHammer vulnerabilities for various DRAM vendors.» *Social Development and Security*, 14(3), 238-244.

2. Mazurok, V., & Lutsenko, V. «Enhancing Row-Sampling-Based RowHammer defense methods with Machine Learning approach» *Theoretical and Applied Cyber Security* Vol. 6 No. 2
3. Mazurok, V., & Lutsenko, V. «Improving the effectiveness of Row-Sampling methods to protect against Row-Hammer attacks». *Social Development and Security*, 14(6), 61-67.
4. Mazurok, V., & Lutsenko, V. Detecting RowHammer attacks using frequency arrays. *Social Development and Security*. 15(1). 42 – 49
5. Мазурок В. Луценко В. «Аналітичний огляд атаки RowHammer на сучасну пам'ять» Міжнародна науково-практична інтернет-конференція «Виклики і загрози для критичної інфраструктури» м. Київ, 8–10 лист. 2022 року, С. 316–317.
6. Мазурок В. Луценко В. «Практична ефективність RowSampling для захисту від RowHammer» V Міжнародна науково-практична конференція “Trends of modern science and practice”, 8–11 лют. 2022 р., Анкара, Туреччина, С. 97–104.
7. Mazurok, V., & Lutsenko, V. «Mathematical model of DRAM for RowHammer protection» XI Міжнародна науково-практична конференція «Integration of science as a mechanism of effective development», 28 листопада 2023 р.
8. Mazurok, V., & Lutsenko, V. «Discovering DRAM access patterns for machine learning sample finding» XXI Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики», м. Київ, 12–13 трав. 2023 р С. 348–352
9. Mazurok, V., & Lutsenko, V. «Machine learning methods of RowHammer mitigation» IX Міжнародна науково-практична конференція «Development of innovation systems: trends, challenges, prospects», 04–07 бер. 2025 р., Гамбург, Німеччина С. 342–345.

Публікації. За матеріалами дисертації опубліковано 9 робіт, 4 з яких – статті у журналах і збірниках наукових праць, що входять до переліку фахових видань, затверджених МОН України за спеціальністю дисертації або у періодичних виданнях іноземних держав та 5 тез науково-практичних конференцій.

Структура та обсяг дисертації. Дисертація складається із анотації, вступу, чотирьох розділів, висновків, списку використаних джерел. Робота містить 208 сторінок, у тому числі: 167 сторінок основного тексту, 41 рисунок, 8 таблиць, список використаних джерел із 107 найменувань на 12 сторінках.

РОЗДІЛ 1

АКТУАЛЬНІСТЬ ПРОБЛЕМИ АТАК НА ПАМ'ЯТЬ ТА ОСНОВНІ ПІДХОДИ ДО ЇЇ РОЗВ'ЯЗАННЯ

1.1 Будова пам'яті персонального комп'ютера

У сучасних обчислювальних системах – як промислових серверах, так і побутових комп'ютерах – пам'ять є ключовим компонентом. Вся пам'ять персонального комп'ютера складається з кількох основних компонентів, кожен з яких відіграє критичну роль у забезпеченні зберігання та обробки даних. Основними складовими є оперативна пам'ять (RAM), постійна пам'ять (ROM), кеш-пам'ять і зовнішні запам'ятовуючі пристрої. Їхня структура та функції оптимізовані для різних типів завдань, які виконуються комп'ютером.

Оперативна пам'ять (RAM) є тимчасовим сховищем даних, що активно використовуються процесором під час роботи системи. Під час роботи найбільша частина даних, які використовуються запущеними програмами, зберігається саме в оперативній пам'яті. Вона складається з мільярдів крихітних комірок, організованих у вигляді рядків і стовпців. Кожна комірка містить транзистор і конденсатор, які зберігають один біт інформації. У сучасних ПК найчастіше використовується DRAM (динамічна RAM), яка потребує регулярного оновлення заряду в конденсаторах для збереження даних. Швидкість доступу до оперативної пам'яті значно вища, ніж до зовнішніх накопичувачів, що робить її важливим елементом для продуктивності системи.

Кеш-пам'ять є швидким проміжним сховищем, розташованим між процесором і оперативною пам'яттю. Вона організована в кілька рівнів (L1, L2, L3), причому L1 є найшвидшою, але найменшою за обсягом. Основна функція кешу полягає в збереженні найбільш часто використовуваних даних і команд,

що дозволяє зменшити затримки під час роботи. Постійна пам'ять (ROM) зберігає дані, які не змінюються під час експлуатації, наприклад, мікропрограму для запуску системи (BIOS або UEFI).

Окрім цього, до структури пам'яті персонального комп'ютера входять зовнішні запам'ятовуючі пристрої, такі як жорсткі диски (HDD), твердотільні накопичувачі (SSD) та флеш-пам'ять. Вони забезпечують довготривале зберігання даних і виконують роль основного сховища інформації. Дані передаються між цими компонентами за допомогою контролерів пам'яті, які оптимізують взаємодію між ними для підвищення швидкодії.

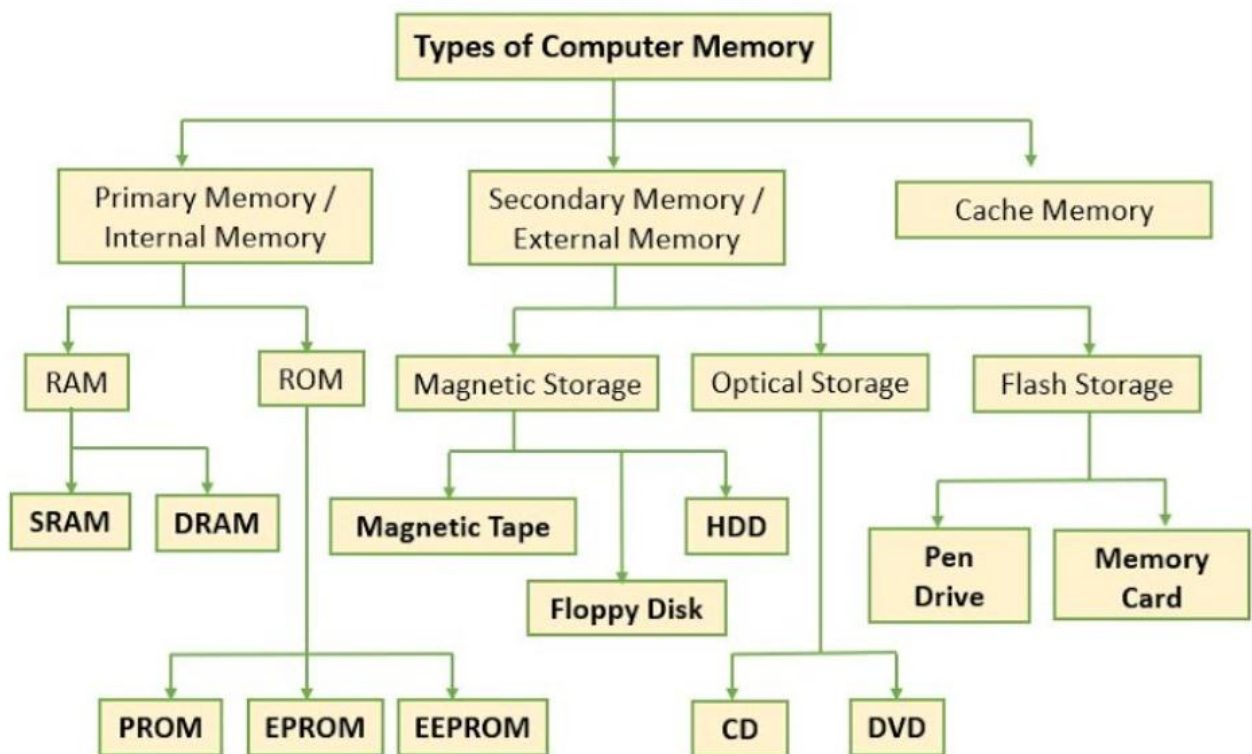


Рисунок 1.1 – Види пам'яті в сучасних комп'ютерних системах

Сучасні архітектури пам'яті постійно вдосконалюються, щоб забезпечити баланс між швидкістю доступу, ємністю та енергоспоживанням. Взаємодія різних типів пам'яті дозволяє комп'ютеру працювати ефективно, обробляючи складні завдання та великі обсяги даних у режимі реального часу.

1.1.1 Основні поняття

Передача даних для роботи програм відбувається між основною пам'яттю та процесором через кілька рівнів кеш-пам'яті, вбудованих у процесор. Доступ до кеш-пам'яті швидший, ніж до основної пам'яті, але вона набагато менша за об'ємом, тож має містити лише найпотрібніші дані. Кеш організовано за зазвичай за чотирма рівнями (від L1 до L4). Кеш-пам'ять кожного вищого рівня має більший об'єм за попередній, але при цьому доступ до нього також повільніший, ніж доступ до кешу рівнем нижче. Така пам'ять використовується для зберігання даних, які найчастіше використовуються процесом, в ідеалі – чим частіше використовуються дані, тим нижче вони зберігаються на рівнях кешу. Зазвичай останні рівні є глобальними для всього процесора, а локально кожне ядро має власні набори кешів лише першого рівня (зазвичай L1 і L2). Крім того, перший рівень кешу розділений на кеш інструкцій і кеш даних. Насамкінець, кожне ядро має власний розділ реєстрів (файл реєстру), який воно використовує для зберігання даних під час виконання над ними операцій. Такі файли реєстру здатні зберігати дуже мало інформації, але вони безпосередньо підключені до арифметико-логічних пристроїв (ALU), тому забезпечують мінімально можливий час доступу. Коли ж реєстри потрібні для зберігання нових даних, вже присутня в них інформація переноситься в адресний простір оперативної пам'яті RAM, яка знаходиться зовні процесора. Якщо запрохані процесором дані не зберігаються в кеш-пам'яті, вони вивантажуються туди з RAM. А коли дані в кеші вже не потрібні, вони зберігаються назад в оперативну пам'ять.

У критичних випадках коли обсягу оперативної пам'яті (або ж основної пам'яті) недостатньо, використовується зовнішній SWAP розділ. Це частина жорсткого диска, яка служить пам'яттю підкачки та переповнення простору для оперативної пам'яті. Використання розділу резервної пам'яті служить для

тимчасового зберігання даних під час виконання, які використовуються найрідше. Така вищеописана архітектура пам'яті показана на рисунку 1.2.

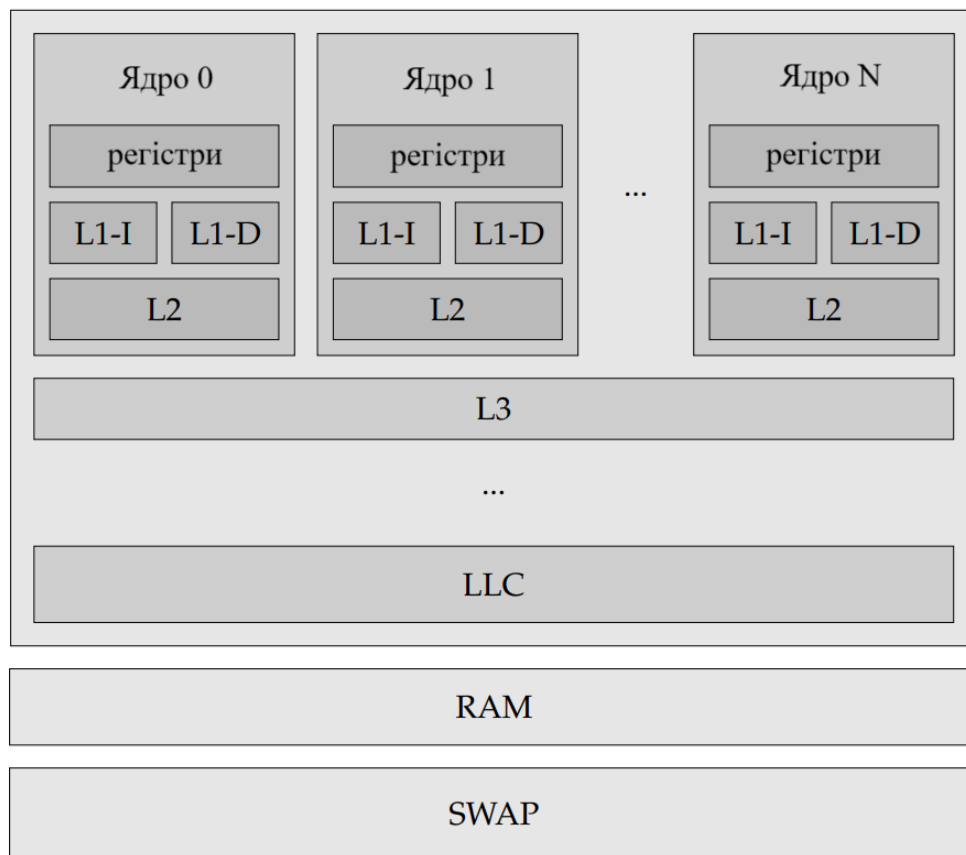


Рисунок 1.2 – Архітектура пам'яті сучасних комп'ютерних систем

Коли ядро запитує деякі дані з пам'яті, воно зазвичай виконує процес, показаний на рисунку 1.3. Запит спочатку переходить до першого рівня кешу (L1-I для інструкцій або L1-D для інших даних). Якщо дані зберігаються в цьому кеші, вони повертаються. В іншому випадку запит переходить до наступного рівня кешу, на якому виконується той самий тест. Якщо дані не зберігаються на жодному рівні кеш-пам'яті, запит передається в основну пам'ять. Після отримання з вищого рівня кешу або основної пам'яті нові дані можна зберігати в поточному кеші, замінюючи старіші дані відповідно до деяких правил вилучення. Подію коли данні були знайдені у кеш-пам'яті називають кеш-попаданням (cache hit), в протилежному ж випадку, коли дані були відсутні в кеш-пам'яті, — кеш-промахом (a cache miss).

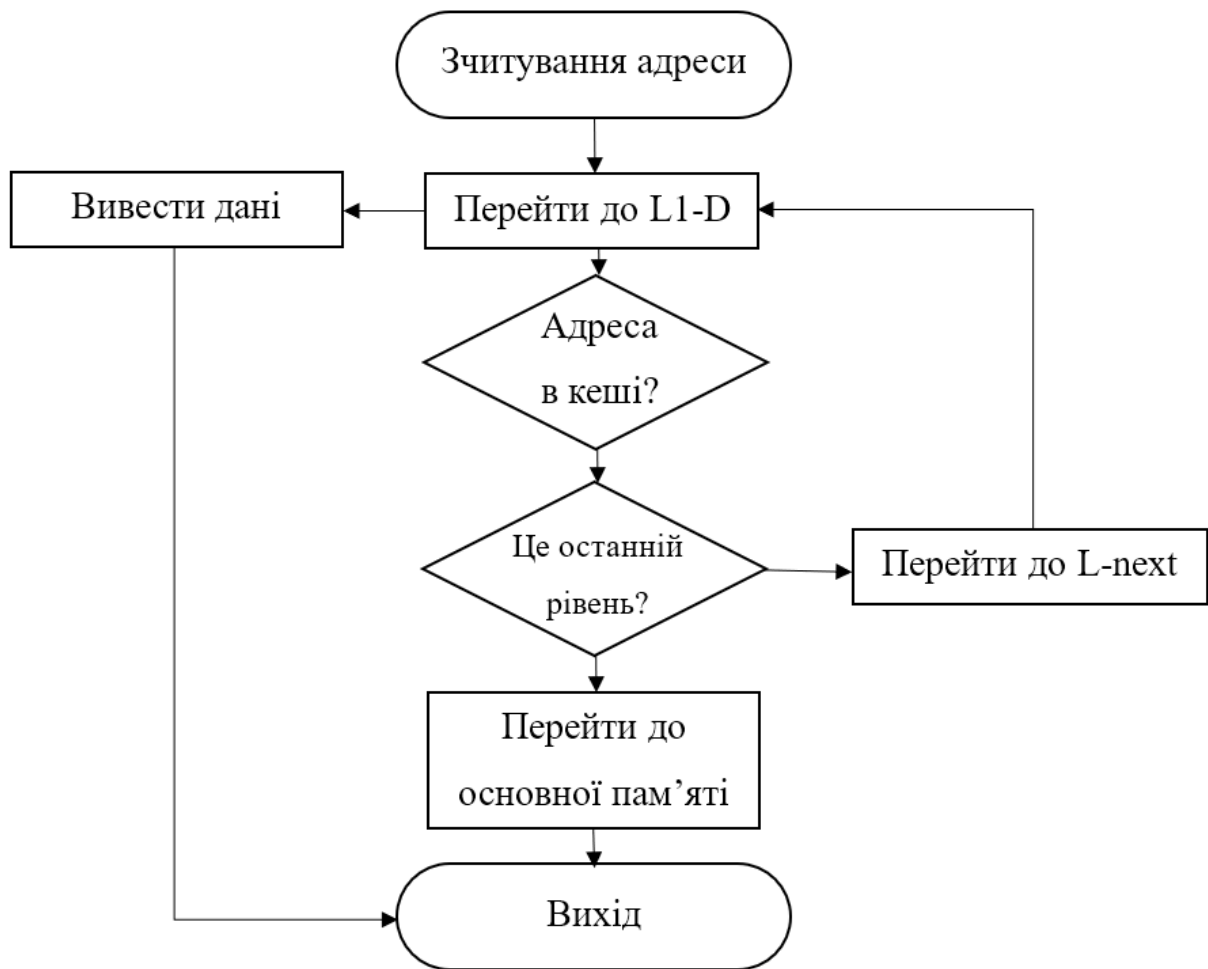


Рисунок 1.3 – Спрощена блок-схема операції зчитування даних з ядра CPU

1.1.2 Архітектура та робота DRAM

У сучасних комп'ютерних системах як споживацького так і виробничого рівня для основної пам'яті програм та процесів використовується технологія динамічної оперативної пам'яті (DRAM). Назва DRAM (Dynamic Random Access Memory) відображає основні характеристики цього типу пам'яті, які визначають її функціонал та архітектурні особливості. Перше слово, "динамічна" (Dynamic), пов'язане з принципом роботи DRAM, де кожна комірка пам'яті складається з конденсатора і транзистора. Конденсатор використовується для зберігання заряду, який представляє логічне значення 0 або 1. Оскільки заряд у конденсаторі поступово розряджається через витік струму, інформація в пам'яті з часом втрачається. Щоб уникнути цього,

необхідно періодично оновлювати дані, виконуючи процес під назвою "refresh". Ця особливість відрізняє DRAM від інших типів пам'яті, зокрема від статичної оперативної пам'яті (SRAM), де дані зберігаються без необхідності оновлення завдяки використанню тригерів.

Друга частина назви, "з довільним доступом" (Random Access), підкреслює можливість прямого доступу до будь-якої комірки пам'яті незалежно від її розташування. Це означає, що для отримання або запису інформації в конкретну комірку не потрібно послідовно переглядати інші комірки, як у стрічкових або інших послідовних запам'ятовуючих пристроях. Завдяки цій властивості DRAM забезпечує швидкий доступ до даних, що є критично важливим для виконання обчислювальних задач у реальному часі.

DRAM слугує основним видом оперативної пам'яті у сучасних системах, оскільки вона пропонує високу ємність зберігання за порівняно низької вартості виробництва. Це досягається завдяки простій структурі комірок, що складаються лише з одного транзистора і одного конденсатора, на відміну від складнішої архітектури SRAM.

У такому типі пам'яті дані адресуються за допомогою наступних рівнів архітектури (від вищого до нижчого): канал, ранг, група банків, банк, рядок і стовпець. Кожен рівень має свої власні біти в слові адреси, залежно від вибраного макета. Дві найпоширеніші структури адрес: Ка-Ра-Ба-Ря-Ст та Ря-Ба-Ра-Ст-Ка (за першими буквами назв рівнів, випускаючи групу блоків), показано на рисунку 1.4.

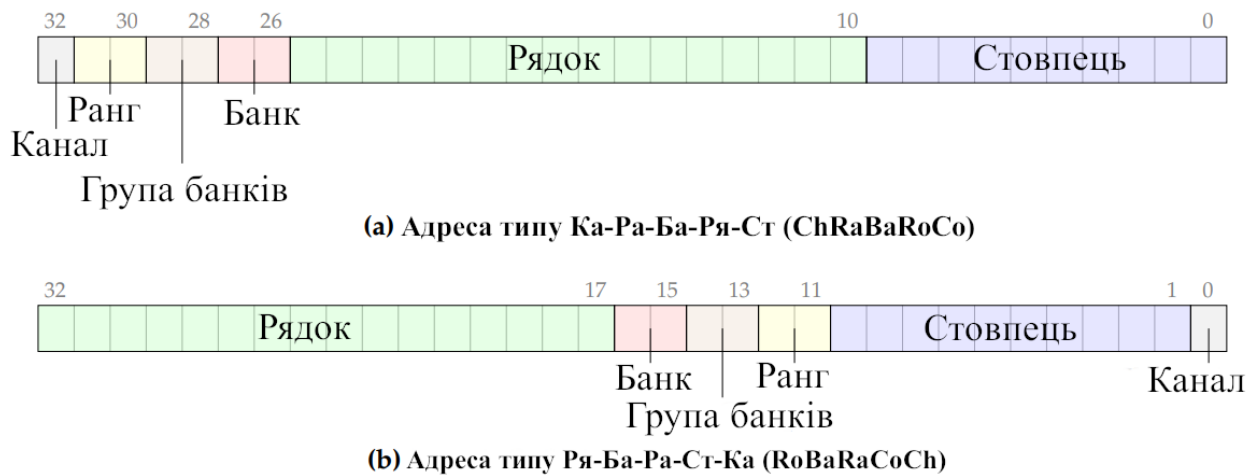


Рисунок 1.4 – Розташування бітів адреси для пам'яті DDR4 з 2 каналів

Внутрішню ж архітектуру пам'яті DRAM детально продемонстровано на рисунку 1.5. Процесор зв'язується з основними запам'ятовуючими пристроями через окремі канали пам'яті (1 на рисунку 1.5), кожен з яких налічує власний набір шин команд, даних і адрес. До одного каналу можна підключити кілька модулів пам'яті (зазвичай модулі пам'яті Dual In-line Memory Modules або ж DIMM), що позначені як 2 на риунку 1.5. Один такий модуль містить кілька мікросхем пам'яті згрупованих в ранги. Командні і адресні шини є спільними для всіх мікросхем, але шина даних розділена для кожної рангової мікросхемами (як показано 3 на рисунку 1.5). Ранговий чіп пам'яті містить кілька банків, об'єднаних в групи (4 на рисунку 1.5). При цьому банк - це найменша структура, видима для контролера пам'яті. Саме банки є кінцевою точкою усіх запитів на доступ до даних оперативної пам'яті. Банк складається з інтерфейсних портів, драйверів для вибору рядка для активації (декодер рядка) і стовпця для читання або запису (логіка вибору стовбців, CSL) і масив індивідуальних комірок пам'яті (англ. Memory Array Tiles, MAT) (5 на рисунку 1.5).

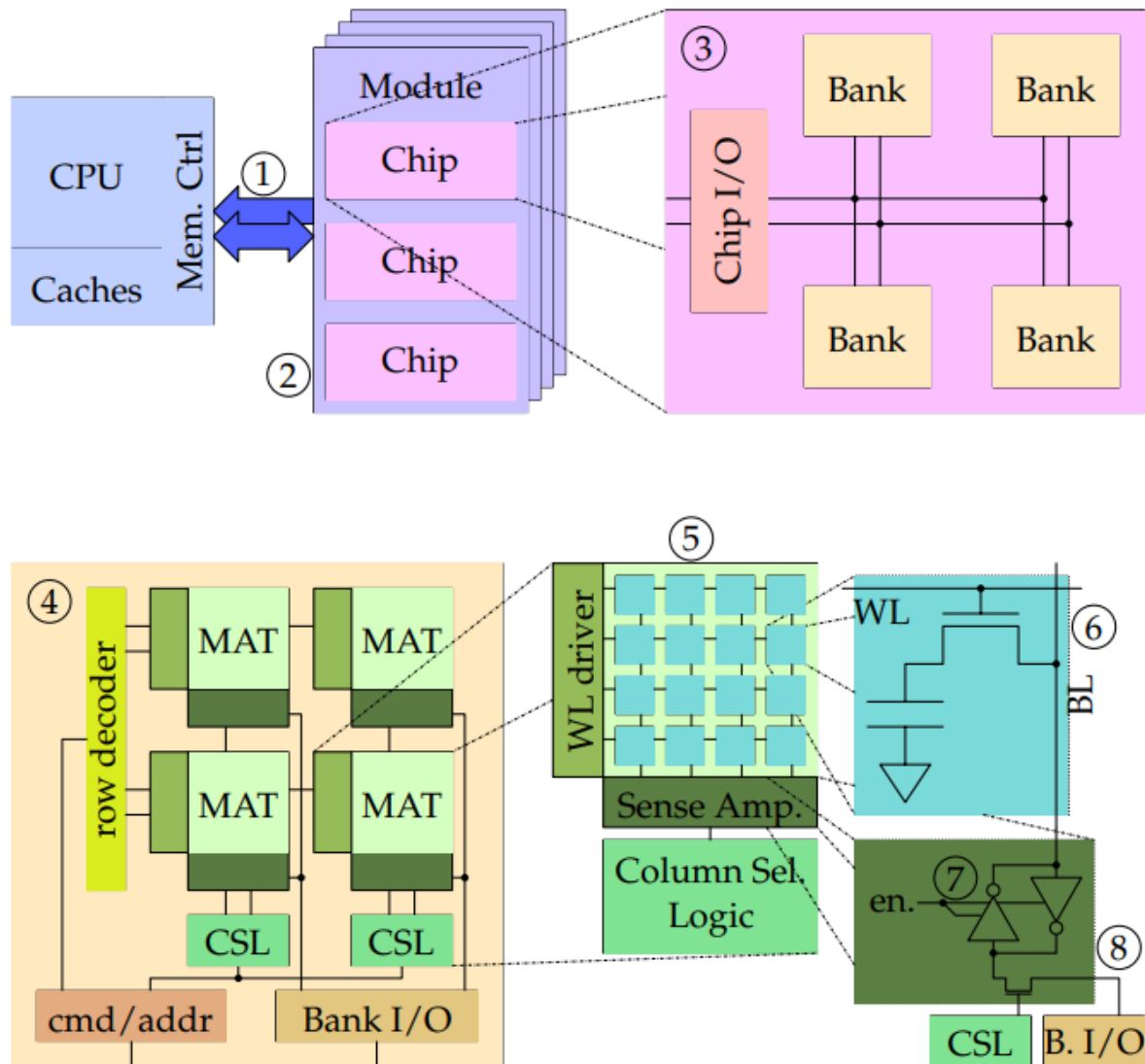


Рисунок 1.5 – Внутрішня будова пам'яті DRAM

У MAT, запам'ятовуючі елементи організовані в матрицю, де всі комірки рядка мають спільну лінію слів (англ. wordline, WL), а всі комірки стовпця мають спільний лінію бітів (англ. bitline, BL) (6 на рисунку 1.5). MAT також містить драйвер WL що відповідає за отримані сигнали і набір підсилювачів сигналу, які використовуються для читання та запису значень у клітинках. Фізично комірка пам'яті являє собою комбінацію одного конденсатора та одного транзистора. Транзистор, який називається транзистором доступу, керується з WL ряду. При активації, елемент підключає кінець конденсатора до BL, насичуючи його. При цьому інший кінець конденсатора постійно

підключений до землі. Заряд даного накопичувального конденсатора і визначає значення біта, що зберігається коміркою.

На кінці бітової лінії знаходиться підсилювач сигналу (англ. Sense Amplifier, SA), який складається з двох інверторів у конфігурації з перехресним зчепленням (7 на рисунку 1.5). Коли відбувається доступ до комірки пам'яті на відповідному рядку, CSL підключає інший кінець SA до шини введення/виведення банку даних. (8 на рисунку 1.5) Підсилювачі сигналу в цій конфігурації також використовуються для зберігання останнього рядка, до якого зверталися, тому набір таких підсилювачів також іноді називають буфером рядків (англ. Row Buffer, RB).

Щоб зчитувати або записувати дані в пам'ять, цільовий рядок повинен бути активований для завантаження свого вмісту в RB, перш ніж отримати доступ до даних у RB. Для роботи з комірками контролер пам'яті оперує чотирма основними командами – дозарядження (PRECHARGED), активація (ACTIVATE), зчитування (READ) та запис (WRITE) як показано на рисунку 1.6. Алгоритм роботи з пам'яттю і цими командами виглядає наступним:

- Якщо жоден рядок не завантажений до RB, такий банк знаходиться в стані очікування.
- Щоб завантажити рядок DRAM в RB, контролер пам'яті видає банку команду активації ACTIVATE (ACT).
- Коли рядок завантажується в RB, контролер пам'яті може видавати команди READ для запиту даних з RB,
- Або ж команди WRITE для зміни даних в RB. Оскільки рядок все ще активований, будь-які зміни в RB відображаються в рядку DRAM.
- Коли здійснюється доступ до іншого рядка, з'єднання між поточним активованим рядком і RB має бути спочатку закрито за допомогою команди PRECHARGE (PRE), скидаючи банк до стану PRECHARGE.

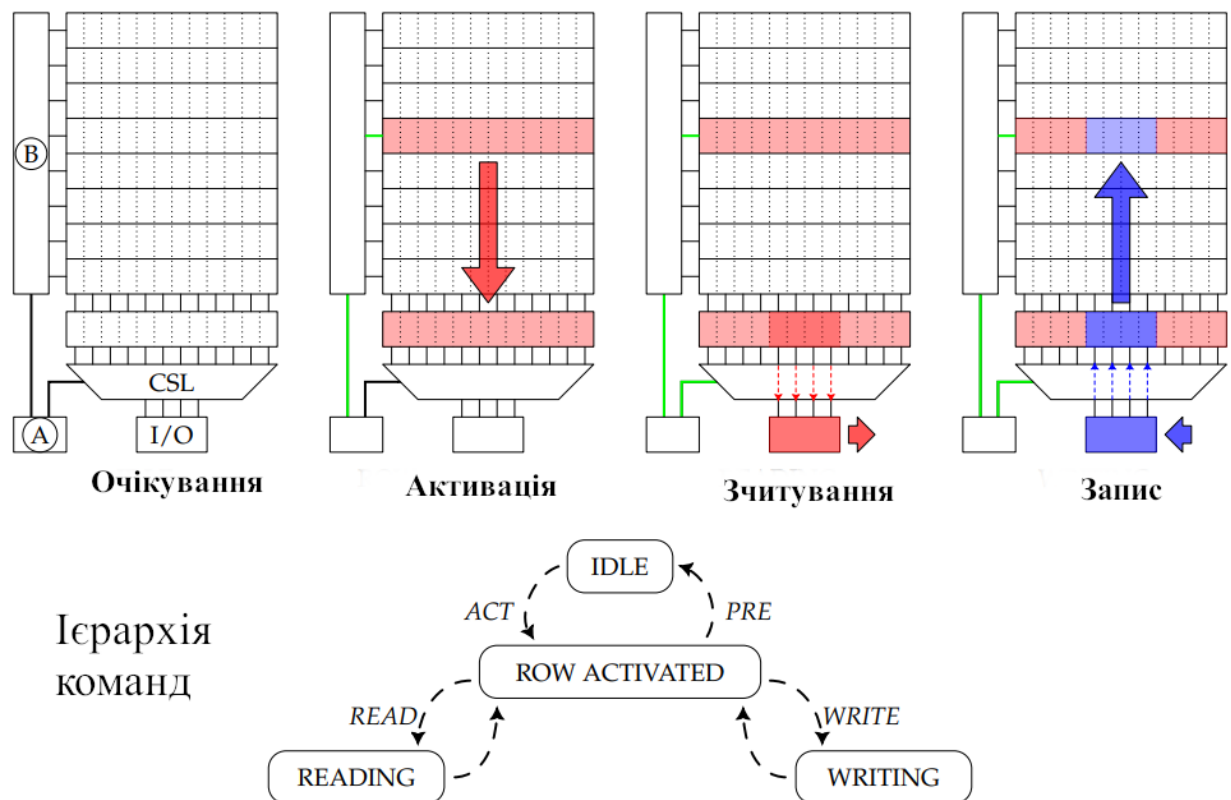


Рисунок 1.6 – Запис та зчитування даних з банку DRAM

Стани на рівні банку та команди, пов'язані з доступом до даних. При цьому декодер адреси позначено як **A** на рисунку 1.6, а декодер рядків і драйвер **WL** позначено як **B** на рисунку 1.6. Якщо потрібно записати логічну одиницю (1), конденсатор заряджається, а якщо логічний нуль (0), він залишається розрядженим. Цей стан конденсатора визначає, яке значення буде зберігатися в комірці до наступного циклу оновлення або перезапису. Однак збереження заряду в конденсаторі є тимчасовим, оскільки через властивості матеріалів і конструкцію елементів відбувається поступовий витік заряду. Щоб запобігти втраті даних, DRAM використовує механізм оновлення, який періодично оновлює заряд у конденсаторах, забезпечуючи стабільність записаної інформації. Частота цих циклів оновлення залежить від фізичних властивостей конденсаторів і температурних умов, у яких працює пам'ять: підвищення температури зазвичай прискорює витік заряду, що потребує частішого оновлення. На електричному рівні завантаження рядків у **RB** виконується за

допомогою циклу ACTIVATE-PRECHARGE і основу його роботи представлено на рисунку 1.7.

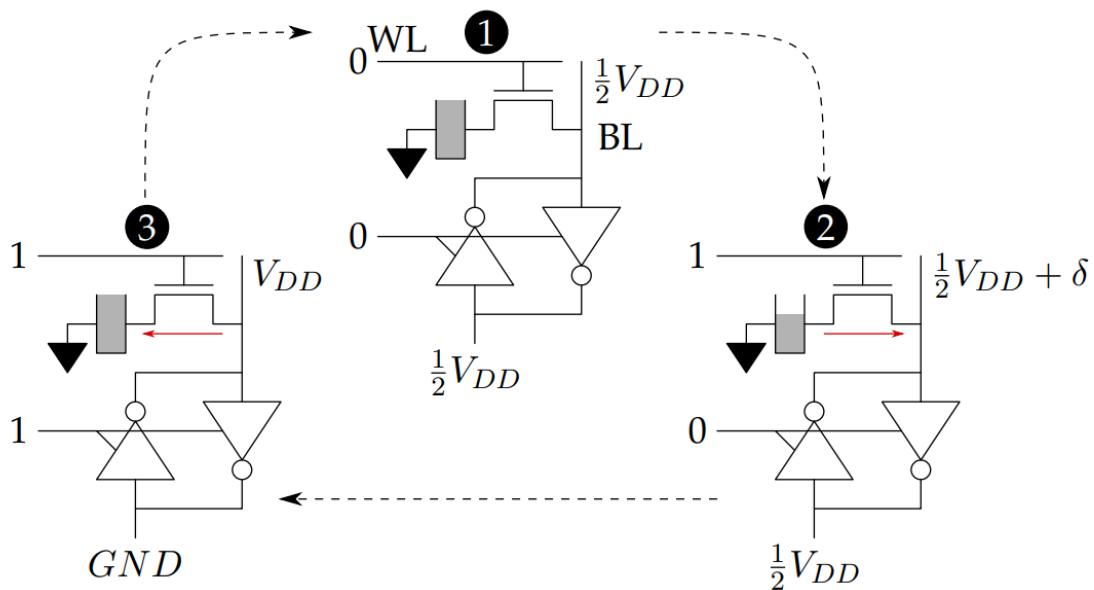


Рисунок 1.7 – Цикл запису та зчитування інформації з комірок DRAM

За замовчуванням кожна запам'ятовуюча комірка пам'яті знаходиться в попередньо зарядженому стані як показано на зображенні 1 на рисунку 1.7 і конденсатор повністю зафарбований як такий, що повністю заряджений (і відповідно повністю розряджений у випадку логічного нуля), WL знижується, і обидва кінці SA, включаючи BL, підтримуються під напругою що рівна половині максимального значення насичення(V_{DD}) тобто рівною в $\frac{1}{2}V_{DD}$. Щоб завантажити вміст комірки пам'яті в буфер рядків, контролер пам'яті видає відповідному банку пам'яті в якому знаходиться потрібна комірка АСТ. Коли банк отримує цю команду, відповідна лінія слів підключається до високого входу, з'єднуючи конденсатор комірок пам'яті в цільовому рядку з їх лінією бітів. Кожен конденсатор і його бітова лінія BL поділяють свій заряд по половині, підвищуючи напругу цієї лінії до $V_{DD} + \delta$, або ж, відповідно опускаючи його до $V_{DD} - \delta$ для протилежного значення комірки. Цей процес проілюстровано на малюнку 2 рисунку 1.7. Тобто напруга бітової лінії змінюється на значення біту, що зчитується. Після цього

Після цього вмикається підсилювач сигналу SA, та виявляє різницю напруги між своїми входами відповідно підсилюючи її, доки різниця потенціалів бітової лінії не досягне рівня V_{DD} , а інший вхід SA не знизиться до рівня GND (відповідно для протилежного значення ці входи будуть змінені місцями на GND і V_{DD}). Саме це ми і можемо бачити на частині 3 рисунку 1.7. Так як конденсатор все ще підключений до бітової лінії, його заряд повністю відновлюється (і відповідно виснажується в протилежному випадку) завдяки вищеописаному процесу зміни напруги в BL.

Під час запису даних у рядковий буфер зміна значення в ньому також змінює і напругу в бітовій лінії BL і, отже, як наслідок і заряд конденсатора, зберігаючи комірку пам'яті оновленою і надалі заповненою належним значенням. Коли ж RB стає потрібен для зберігання іншого рядка, поточний активований рядок має бути спочатку закритий. Коли банк отримує PRE команду дозарядження, лінія слів активного рядка опускає свій потенціал, ізолюючи її від відповідної бітової лінії. Напруга обох кінців підсилювача SA повертається до $\frac{1}{2}V_{DD}$, як на 1 частині рисунку 1.7 і система повертається до попередньо зарядженого стану, при цьому будучи готовою до наступної команди активації АСТ від банку.

Під час доступу до даних, якщо цільовий рядок уже завантажено в RB (ми називаємо цю подію а потрапляння по рядку row hit), операція виконується безпосередньо на рядковому буфері. Якщо ж якийсь інший рядок є активним в буфері (ми називаємо цю подію а конфлікт рядків row conflict), банк повинен попередньо дозарядити попередній рядок і активувати цільовий рядок який було запитано перед поверненням даних. Якщо ж в банку наразі немає активного рядка (ми називаємо цю подію а промах рядка row miss), банку потрібно лише активувати цільовий рядок, який було запрошено перед поверненням даних.

Конденсатори не є ідеальними елементами для утримування заряду. З часом вони його втрачають через процес відомий як витікання заряду. Якщо до комірки пам'яті не відбувається доступ (тобто в той час коли її значення, завантажене у буфер рядку) з достатньою частотою, щоб повторно заповнити конденсатор, заряд буде зменшуватися, доки він не опуститься нижче порогу, при якому конденсатор може підвищити напругу бітової лінії, коли рядок активовано. За температури навколишнього середовища більшість комірок пам'яті втрачають записані дані через кілька секунд бездіяльності. Цей ефект змінюється від клітини до клітини та обернено пропорційний температурі, тому він знижується в холодному середовищі. Також це дуже залежить від фізичних властивостей матеріалів з яких зроблено елементи комірки пам'яті, тож це значення можуть варіюватись навіть в межах рядка, не кажучи вже про банк чи цілий чіп пам'яті. Щоб протистояти цій проблемі та уникнути втрат даних під час роботи системи, контролер пам'яті періодично видає до банку пам'яті команди REFRESH (REF). Щоб обробити цю команду, банки пам'яті активують всі свої рядки, (або ж по чергово для оптимізації пам'яті та її енерговикористання) щоб заповнити накопичувальні конденсатори.

Відповідні параметри синхронізації і оновлення DRAM проілюстровано на рисунку 1.8 і наведені в таблиці 1.1 для трьох моделей DDR3, DDR4 і DDR5. Мінімальний інтервал між двома командами активації АСТ визначається за допомогою t_{RC} , t_{RRD_L} або t_{RRD_C} якщо рядки для активації відповідно розташовані в тому самому банку, в різних банках однієї групи банків або в різних групах банків. Крім того, на кожен ранг можна видати максимум чотири АСТ за час вікна доступу t_{FAW} . Магічне число становить лише 4, оскільки (в межах одного рангу) є лише 8 банків, куди можуть бути розподілені послідовні доступи. Проблема полягає в тому, що струм активації банку вичерпується з усього масиву (або банку) комірок пам'яті (транзистор + конденсатор), необхідних для підготовки місця для зчитування. Якщо послідовні операції читання потрапляють щоразу в інший рядок у різному банку, проблема

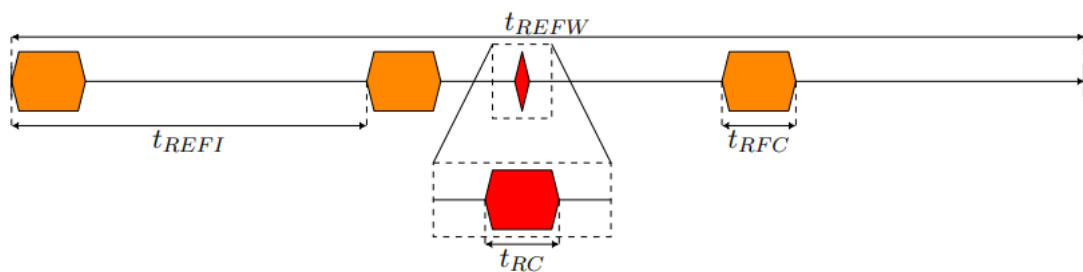
активації виникає щоразу. Щоб запобігти сильному споживанню струму з масивів, потрібно було обмежити кількість послідовних активацій банку протягом ковзаючого вікна часу. Причина в тому, що потрібно запобігти високим стрибкам енергії/струму споживання банку, оскільки це може спричинити спотворення опорної напруги (V_{DD}) в момент зчитування/запису. Так як V_{DD} це стандартний рівень напруги лінії слів – з ним порівнюються дані комірок, зчитані з банку, щоб зрозуміти, чи те, що ви читаєте, дорівнює 1 чи 0. Ризик зміни рівня V_{DD} може означати, що достовірність того, що буде зчитано або записано буде під загрозою. Однак ці параметри як t_{FAW} і t_{RRD} зазвичай завищені, і тюнер пам'яті може підвищити продуктивність системи, трохи змінивши ці параметри. При цьому команди REF видаються контролером пам'яті один раз в інтервалі вікна оновлення (t_{REFI}) і тривають за часом проміжок в t_{RFC} . Так у циклі t_{REFW} , усі рядки пам'яті оновлюються принаймні один раз.

Таблиця 1.1: Часові параметри синхронізації для DDR3 1600 8Gb x8[6], DDR4 2400 8Gb x8[7], та DDR5 4000 8Gb x8[8].

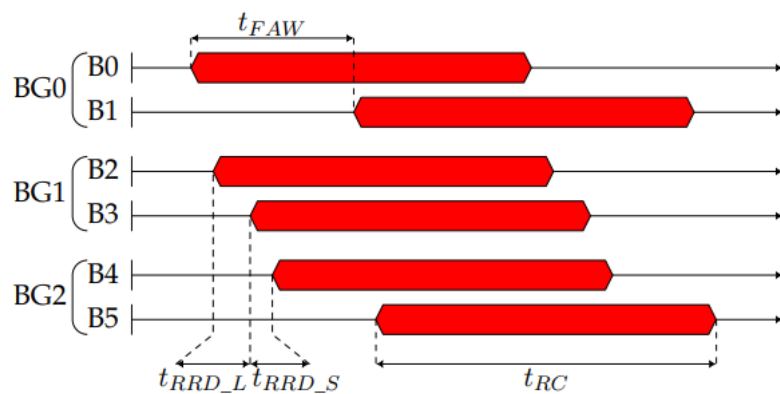
Умовне позначення	Опис	DDR3	DDR4	DDR5
t_{RC}	Мінімальний інтервал доступу того самого банку	48,75 нс	45,8 нс	46 нс
t_{RRD_C}	Мінімальний інтервал доступу для різних груп банків	6,25 нс	3,3 нс	4 нс
t_{RRD_L}	Мінімальний інтервал доступу для тої ж групи банків	-	4,9 нс	5 нс

t_{FAW}	Вікно активації рангу (максимум 4 команди)	30 нс	21,6 нс	16 нс
t_{REFW}	Вікно оновлення	64 мс	64 мс	32 мс
t_{REF}	Інтервал REF (мкс)	7,8 мкс	7,8 мкс	3,9 мкс
t_{RFC}	Тривалість команди REF	350 нс	350 нс	195 нс

На ілюстрації таймінгів DDR на рисунку 1.8 горизонтальна вісь - час. Кольорові секції представляють періоди, коли банк зайнятий. Помаранчева частина представляє банк, зайнятий командою оновлення REF. Червона ділянка позначає період після команди активації АСТ, коли банку не можна надіслати іншу команду АСТ. Для часового представлення таймінгів доступу до рівня рангу (2 на рисунку 1.8) представлено 3 групи банків, по 2 банки для кожної групи банків.



(1) Часові інтервали на рівні банку



(2) Часові інтервали на рівні рангу

Рисунок 1.8 – Часова діаграма запису та зчитування інформації з DRAM

1.1.3 Математична модель представлення пам'яті

Математичне моделювання пам'яті є важливим інструментом для опису та аналізу її структури, функціонування та взаємодії з іншими компонентами системи. Основою моделі пам'яті є представлення її як набору комірок, кожна з яких має унікальну адресу та значення. Математична модель представлення пам'яті використовується для формалізації структури, організації та процесів доступу до пам'яті в комп'ютерних системах. Вона дозволяє описати пам'ять як множину адресованих комірок, де кожна комірка може зберігати деяке значення.

Нехай M представляє модель пам'яті, де:

$$M = \{(a, v) \mid a \in A, v \in V\}, \quad (1.1)$$

де A — множина адрес (наприклад, $A = \{0, 1, \dots, N - 1\}$ для пам'яті з N адресами), а V — множина можливих значень, які можуть зберігатися в комірках пам'яті (наприклад, множина бітових рядків фіксованої довжини). Для кожної адреси $a \in A$, функція $M(a)$ повертає відповідне значення даних, що зберігається в цій комірці. При цьому також якщо ми говоримо про сучасні комп'ютери, то v приймає значення 0 чи 1.

Основні операції роботи з пам'яттю — це *читання* та *запис*. Операція читання $read(a)$ повертає значення $M(a)$, яке зберігається за адресою a . Операція запису $write(a, v)$ змінює стан пам'яті, зберігаючи значення $v \in D$ за вказаною адресою a . Формально це можна записати як:

$$M_{new}(a') = \begin{cases} v, & \text{якщо } a = a' \\ M(a'), & \text{в іншому випадку} \end{cases} \quad (1.2)$$

де M_{new} — новий стан пам'яті після виконання операції запису.

З цієї спрощеної системи виводиться і основна математична модель пам'яті з розділенням на багаторівневу ієрархію. Нехай M_1, M_2, \dots, M_n — рівні пам'яті (кеш, RAM, диск тощо), тоді загальний час доступу до даних можна виразити як зважену суму:

$$T_{avg} = \sum_{i=1}^n P_i T_i \quad (1.3)$$

де T_i — час доступу до i -го рівня пам'яті, а P_i — ймовірність звернення до цього рівня. Для врахування специфічних властивостей пам'яті, таких як час оновлення в DRAM, додаються часові характеристики. Час циклу пам'яті t_{cycle} можна описати як суму трьох компонентів:

$$t_{cycle} = t_{read} + t_{write} + t_{refresh} \quad (1.4)$$

де t_{read} — час читання, t_{write} — час запису, $t_{refresh}$ — час, потрібний для оновлення зарядів у комірках (для DRAM).

Математичні моделі також враховують імовірнісні характеристики, наприклад, у моделюванні атак на пам'ять, таких як RowHammer. Для цього можна описати ймовірність появи зміни бітів на протилежні у пам'яті як функцію від частоти активації рядків $P_{flip} = f(N_{activations})$, де $N_{activations}$ — кількість активацій агресорського рядка. Ці моделі на базовому рівні дозволяють аналізувати продуктивність, стабільність і безпеку пам'яті, а також оптимізувати її параметри для зменшення ризику виникнення помилок або атак. Для пам'яті DRAM модель базується на розгляді кожної комірки як окремого елемента, що може перебувати у двох станах: зарядженому (1) або розрядженому (0). Ці стани відображаються у вигляді бінарної змінної C_{ij} , де i і j позначають рядок і стовпець відповідної комірки у матриці пам'яті. Таким чином, стан всієї пам'яті можна подати як матрицю M , де $M[i,j] = C_{ij}$.

Збереження стану комірки залежить від процесу запису, який описується як функція $W(C_{i,j}, D)$, де D — це вхідне значення, яке потрібно записати. Якщо $D=1$, функція заряджає конденсатор $C_{i,j}$, переводячи його у стан 1. Якщо $D=0$, конденсатор розряджається, що відповідає стану 0. Оновлення або рефреш моделюється як функція часу $R(C_{i,j}, t_{REF})$, яка відновлює початковий стан комірки в момент часу t_{REF} , якщо зчитування показує втрату заряду через витік. Це дозволяє враховувати як природні характеристики комірок, так і необхідність циклічного оновлення.

Важливим аспектом моделі є динаміка втрати заряду, яка може бути описана через експоненційне зменшення рівня заряду в часі. Для цього можна використати функцію $Q(t) = Q_0 e^{-\lambda t}$, де $Q(t)$ — залишковий заряд у конденсаторі, Q_0 — початковий заряд, а λ — коефіцієнт витоку, який залежить від фізичних властивостей матеріалу та температурних умов. Якщо $Q(t)$ падає нижче певного порогового V_{DD} значення або ж Q_{min} , у випадку математичного представлення, стан комірки змінюється з логічної одиниці на нуль, що моделює втрату даних. Останній елемент моделі — це оптимізація роботи пам'яті, яка включає розрахунок частоти рефрешу F_r , що мінімізує втрати заряду без зайвого збільшення енергоспоживання. Це може бути сформульовано як задача оптимізації: $\min(F_r)$, за умови $Q(t) \geq Q_{min}, \forall t$. Використання такої моделі дає змогу проаналізувати, як змінюються характеристики пам'яті залежно від технологічних параметрів, і дозволяє проектувати ефективніші системи захисту та управління пам'яттю.

1.2 Основні вектори атак на пам'ять

Пам'ять стає джерелом доступу для кібератак, що викликає занепокоєння щодо безпеки на системному рівні, оскільки пам'ять майже всюдишуща в електроніці, а порушення важко виявити. Щодня зростає кількість новин про те, що хакери атакують споживацьку електроніку, промислові та комерційні

сегменти комунікацій, а також зростаючу кількість малих та повсякденних пристроїв, підключених до Інтернету та один до одного. Згідно з опитуванням, проведеним Splunk[9], 54% підприємств стикалися принаймні щомісяця з кібернетичними збоями системи/мережі. Також повідомлялося, що близько 70% вразливостей у продуктах Microsoft пов'язані з проблемами безпеки пам'яті.

1.2.1 Основні поняття

Атаки на основі пам'яті, також відомі як безфайлові атаки або атаки без зловмисного програмного забезпечення, швидко стають значною загрозою у сфері кібербезпеки. Вони описують категорію кіберзагроз, у яких зловмисник використовує вразливі місця в пам'яті комп'ютерної системи для виконання зловмисних дій, уникаючи виявлення традиційними файловими антивірусними рішеннями. Цей складний і неортодоксальний спосіб розгортання кібератак ставить перед розробниками рішень з кібербезпеки та антивірусним програмним забезпеченням нові виклики. Розібравшись на аналітичному рівні обговорення і розуміння атак на основі пам'яті, ми можемо поглибити і оцінити масштаби цієї зростаючої проблеми. На базовому двійковому рівні центрального процесора комп'ютерна програма — це лише серія інструкцій — коли ці інструкції записуються під час виконання в пам'яті комп'ютера (RAM), вона стає програмою, з якою ми взаємодіємо. Традиційно атаки на основі зловмисного програмного забезпечення зосереджені і містяться на файлах, встановлених на жорсткому диску комп'ютера. Ці файли часто були замасковані, приховані або вбудовані в, здавалося б, нешкідливі частини програмного забезпечення чи вмісту. Інструменти захисту від вірусів або зловмисного програмного забезпечення виявляли і видалятимуть ці шкідливі файли на основі попередньо визначених сигнатур або евристик, таким чином захищаючи хост комп'ютер від шкоди, що такі програми можуть заподіяти.

Атаки, засновані на пам'яті, принципово відрізняються за режимом роботи та ознаками, які неможливо виявити. Такі вторгнення не встановлюють постійні копії зловмисного програмного забезпечення на жорсткий диск системи, а отже, обходять виявлення на основі сигнатур і не залишають слідів для розслідування після інциденту. Вони використовують дозволені системні інструменти та процеси, щоб використовувати вразливість простору в пам'яті, позиціонуючи шкідливий код для безпосереднього виконання в оперативній пам'яті. Це робить атаки на основі пам'яті прихованими, ефемерними та неймовірно складними для виявлення та усунення за допомогою звичайних заходів безпеки. Ін'єкція зашифрованого, трансформованого зловмисного корисного навантаження або камуфляжу під час звичайної роботи системи, виконання корисного навантаження лише за певних умов і динамічні зміни з оновленнями роблять ці послідовності в пам'яті надзвичайно складними для ідентифікації та блокування з самого початку.

Ключовим прикладом є зловмисне програмне забезпечення на основі PowerShell, яке може запускати складні шкідливі сценарії безпосередньо в оперативній пам'яті, не записуючи жодного додаткового файлу на диск. Використовуючи власний інструмент адміністрування Windows, ці атаки можуть органічно поєднуватися зі звичайною діяльністю системи, залишаючись поза увагою більшості рішень безпеки. Настільки ж сприятливим для атак на основі пам'яті є фішинг, у якому використовуються соціально створені вразливості. Зловмисники імітують довіреного контакта та спонукають співробітника завантажити та запустити шкідливий файл. Потім файл використовує дозволені системні інструменти для виконання команд у фоновому режимі та вводить шкідливий код у системну пам'ять. Деякі поширені типи атак на основі пам'яті включають атаки переповнення буфера, атаки DLL ін'єкції і атаки руткітів. Ці атаки можуть здійснюватися різними способами, наприклад фішинговими електронними листами, шкідливими веб-сайтами або завантаженням зараженого програмного забезпечення. У недавній

презентації щодо вразливості RowHummer було продемонстровано атаку[12] що можлива лише завдяки браузеру.

Щоб захиститися від атак на основі пам'яті, окремі особи та організації повинні переконатися, що їхнє антивірусне програмне забезпечення оновлено та проводить регулярне сканування. Їм також слід бути обережними, завантажуючи або відкриваючи файли з невідомих джерел, і регулярно оновлювати свою операційну систему та програми, щоб виправити будь-які відомі вразливості. Крім того, впровадження політики надійних паролів і багатофакторної автентифікації може допомогти запобігти несанкціонованому доступу до конфіденційної інформації.

Нездатність традиційних антивірусних інструментів сканувати та захищати системну пам'ять у поєднанні з безфайловою природою цих атак полегшує їх проникнення, виправдовуючи перехід від звичайних атак на основі файлів до атак на основі пам'яті. Ці атаки потребують спеціалізованих систем виявлення на основі поведінки, які можуть відстежувати поведінку системи та мережі, позначати або ізолювати аномалії або впроваджувати передові методи машинного навчання для запобігання, виявлення та протидії таким атакам. Інші включають сувору політику безпеки, навчання співробітників, агресивне встановлення виправлень, зміну апаратного забезпечення маршрутизатора, сегментацію мережі та використання реальних експертів, які можуть допомогти передбачити, виявити та помістити в карантин ці безфайлові вторгнення.

Атаки на основі пам'яті сигналізують про новий рубіж у кібератаках, оскільки поточні рівні захисту від вірусів і шкідливого програмного забезпечення виявляються недостатніми. Боротьба з цими загрозами вимагає розвитку наших засобів захисту для належного захисту комп'ютерних систем. Це вимагає нових методів і складних комбінацій технологій, включаючи

аналітику великих даних , машинне навчання та штучний інтелект , що веде кібербезпеку в нову еру. Такий, до якого ми неминуче маємо підготуватися.

1.2.2 Атаки побічними каналами

Атаки побічними каналами – це клас атак, коли зломисник намагається оцінити стан криптографічного пристрою та його вміст. Це досягається шляхом спостереження та аналізу інформації, яку можна спостерігати за допомогою різних методологій доступу.

Оскільки різні процеси, які виконуються на мікросхемах, мають різні параметри виконання, вони мають унікальні сигнатури живлення. Аналіз цих сигнатур потужності може дати підказки щодо того, які саме дані обробляються в даний момент. Існує два типи аналізу потужності: диференціальний аналіз потужності (DPA) і простий аналіз потужності (SPA). Обидва методи повинні мати прямий доступ до штифтів живлення на чіпі та аналізувати дані шляхом прямого дослідження та трансляції або статистичного аналізу флуктуацій. Це потрібно через те, що живлення всієї системи може бути дуже зашумленим іншими процесами, але все ж диференційний аналіз навіть в таких випадках показує деякі позитивні результати.

Отримання даних щодо живлення мікросхем відносно нескладне і представлене на рисунку 1.9. Все, що потрібно, це резистор, розміщений паралельно відповідним контактам, який контролює потужність, споживану криптографічною операцією. Пристрій для відбору проб, наприклад осцилограф, розміщується на резисторі, і зміни напруги на резисторі збираються та аналізуються.

SPA ж перевіряє функції, такі як синхронізація, атрибути пристрою, структура алгоритму тощо, які можна спостерігати безпосередньо в одній лінії потужності або шляхом порівняння пар записів використання енергії. Він

більше покладається на розпізнавання образів, ніж на математичний аналіз, і корисний для більш масштабних коливань потужності. Його сильною стороною є те, що він може розкривати послідовність виконуваного коду. Отже, він може розкривати криптографічну інформацію, таку як обчислення та перестановки ключів DES.

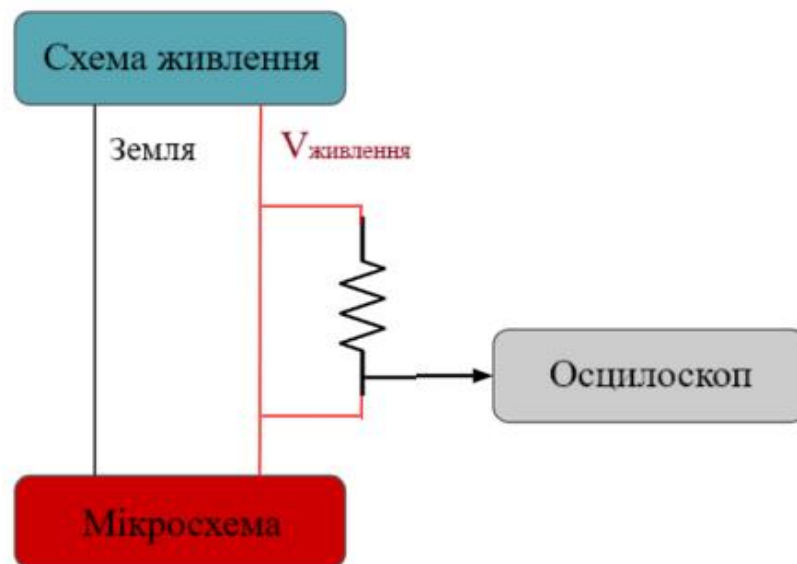


Рисунок 1.9 – Схема отримання показників потужності мікросхеми

Простий аналіз потужності передбачає спостереження за часовою залежністю енергоспоживання і пошук характерних шаблонів, які відповідають певним криптографічним операціям або інструкціям. Основна залежність виглядає так:

$$P(t) = P_{static} + P_{dynamic}(D, t) \quad (1.5)$$

де:

- $P(t)$ — загальна потужність в момент часу t ;
- P_{static} — статична складова потужності, яка є постійною (наприклад, через витоки струму);
- $P_{dynamic}(D, t)$ — динамічна складова, яка залежить від оброблюваних даних D і операцій у момент часу t .

Для SPA характерне пряме співставлення шаблонів енергоспоживання із відповідними етапами виконання алгоритму, що дозволяє виділити окремі операції, наприклад, множення, додавання або зчитування ключа.

DPA набагато краще аналізує процедури живлення, ніж SPA, оскільки він може аналізувати аномалії, пов'язані зі значеннями даних, використовуючи статистичний аналіз. Процедура аналізує зібрані дані для підмножин, бере середні значення та обчислює різницю середніх значень. Потім підмножини призначаються кластерам даних (не має значення, яка підмножина якому кластеру призначена). Як виявилось, якщо підмножини пов'язані з кластерами, перестановки підмножини будуть наближатися до деякого кінцевого числа. Якщо вони некорельовані, то перестановки будуть наближатися до нуля. Зрештою, за наявності достатньої кількості семплів даних, навіть дуже крихітні кореляції можна ідентифікувати в межах дослідження. Основні формули для такого аналізу виглядає так:

Розрахунок потужності для гіпотетичних значень ключа:

$$P_h = f(D, K_h) \quad (1.6)$$

де:

- P_h — прогнозована потужність для гіпотетичного значення ключа K_h ;
- D — дані, що обробляються;
- K_h — гіпотетичне значення ключа;
- $f(D, K_h)$ — функція, що описує залежність потужності від обробки даних і гіпотетичного ключа.

Розрахунок середнього значення споживання енергії для кожної гіпотези ключа:

$$\overline{P_h} = \frac{1}{N} \sum_{i=1}^N P_h(i) \quad (1.7)$$

де N — кількість обчислювань для різних вхідних даних.

Кореляційний аналіз між виміряною потужністю $P(t)$ і прогнозованими значеннями P_h :

$$\text{Corr}(P(t), P_h) = \frac{\text{Cov}(P(t), P_h)}{\sigma_{P(t)} \cdot \sigma_{P_h}} \quad (1.8)$$

де:

- $\text{Cov}(P(t), P_h)$ — коваріація між виміряним і прогнозованим споживанням потужності;
- $\sigma_{P(t)}$ та σ_{P_h} — стандартні відхилення.

Максимальна кореляція тут вказує на правильне значення ключа.

Так аналізуючи електромагнітне випромінювання, що виходить із пристрою, можна неінвазивно витягти ключі та іншу конфіденційну інформацію з пристрою. По суті зломисник за допомогою допоміжних пристроїв і виходів системи вимірює рівні потужності та розбирає наведення в різних частинах чіпа та використовує статистичний аналіз. Вимірювання цих коливань потужності може визначити тип обчислень, які виконує система, а повторні перестановки та аналіз можуть виявити біти криптоключа. Достатня кількість повторень зрештою дасть повний ключ. Це просто питання часу та запису сигналів одного і того ж шифрованого тексту та витоку сигналів на бічні канали, застосування постобробки, і не випадкові ключі [17].

До більш поширених типів атак побічними каналами належать:

1. Електромагнітний що дозволяє знімати дані, коли процесори мікросхеми виконують свої функції та алгоритми і відповідно створюються електромагнітні поля. Так як рух електронів спричиняє результуюче електромагнітне поле (ЕМ-поле), яким би малим воно не було, озброївшись деякими знаннями та відповідним обладнанням, це поле можна виміряти й

проаналізувати. Такі поля вільно та повсюдно доступні практично на будь-якому чіпі, який не має якогось типу радіочастотного екранування або процесів усунення витоків. Обладнання, яке використовується для захоплення та аналізу ЕМ-поля, що випромінюється криптопроцесором, таке ж, як і будь-яка установка ЕМ-аналізу. Він просто повинен вміти вловлювати найменші електромагнітні поля. До нього входять зонди; живлення – це не що інше, як датчик напруги або струму, і певна конфігурація котушки та моніторингу. Інше обладнання — це цифровий накопичувач, підсилювач із високою смугою пропускання та робоча станція з програмним забезпеченням аналізу РЧ/ЕМ.

2. Моніторинг потужності базується на вищеописаному феномені, що різні процеси, що виконуються на мікросхемах, характеризуються унікальними параметрами роботи, які створюють відмінні сигнатури споживання потужності. Дослідження цих сигнатур може вказати на те, які саме дані обробляються в певний момент часу. Для такого аналізу використовують два основні підходи: простий аналіз потужності (SPA) та диференціальний аналіз потужності (DPA). Обидва методи описані вище формулами 1.5-8 і можуть бути використані як вид атаки побічними каналами по потужності.

3. Часові атаки. Ці атаки аналізують час, необхідний для виконання різних криптографічних операцій. Зловмисник аналізує алгоритми та визначає часові інтервали для них. Потім вимірювання вводяться в статистичну модель, яка виводить, наприклад, певну варіацію ключа. Хоча це може бути не точний ключ, він матиме певну міру достовірності. Такий ключ знов запускається в аналізовану систему і такий ітеративний процес використовується для виконання статистичного кореляційного аналізу інформації про час, щоб, зрештою, відновити правильний ключ. Часові атаки найбільш ефективні проти таких алгоритмів шифрування, як RSA, ElGamal і цифрові підписи.

4. Атака через помилки(DFA). Це вже більш інвазивний тип атак, через те, що вони роблять щось із чіпом, щоб порушити його функціональність. Такі

атаки все ще вважаються атаками по бічним каналам, оскільки використовують ту саму методологію аналізу, що й деякі неінвазивні атаки, зокрема диференційований аналіз помилок. Як і у випадку з DPA, DFA намагається витягнути ключі або криптографічні дані подібним чином до аналізу потужності, за винятком того, що це спричиняє відхилення в алгоритмах як одну з частин процесу атаки і аналізу.

Це створює відому аномалію в криптографічних процесах (для цього прикладу описано, алгоритми DES, але такий процес також може бути застосований до RSA, IDEA, RC5, DSA та інших криптографічних шифрів), щоб спричинити їх збій. Такі несправності можуть включати, наприклад, нагрівання, перевищення/зниження напруги, зсув внутрішнього таймера, паразитні електромагнітні поля або випромінювання. Успішна атака через помилки може призвести до порушення роботи програми. Це може призвести до, пропуску кроку підтвердження PIN-коду до прикладу. Також можливо створити дамп всього вмісту пам'яті, включаючи, наприклад, секретний ключ.

Хоча методології відрізняються залежно від шифру, основна передумова полягає в тому, щоб внести помилку під час процесу шифрування. До прикладу, за допомогою збою напруги чи тактового сигналу, або ін'єкції помилки в одному з етапів виконання. При цьому буде спостерігатись різниця в результатах двох або більше запусків шифрування з тим самим відкритим текстом і ключем. Оскільки криптографічний алгоритм повністю специфікований і відомий зловмиснику, за такою різницею, стає можливим відстежити вхідні дані в зворотному напрямку через алгоритм.

1.2.3 Атаки при завантаженні

Якщо зловмисник має фізичний доступ до пристрою з цінною інформацією, атаки холодного завантаження можуть бути використані для отримання такої інформації навіть якщо доступ до неї закритий програмно.

Коли пристрій примусово вимикається, наприклад, утримуючи кнопку живлення протягом кількох секунд, хакери можуть створити дамп пам'яті з DRAM та RAM протягом кількох секунд або хвилини. Протягом цього часу вміст пам'яті все ще доступний. І якщо вміст не зашифровано, хакери можуть використовувати його для доступу до інших мереж або серверів. Якщо оперативну пам'ять охолодити, наприклад, за допомогою рідкого азоту або іншого охолоджувача, час збереження залишкових даних можна значно збільшити, що відкриває можливість їх відновлення.

Під час атаки зловмисник вимикає комп'ютер і швидко перезавантажує його із зовнішнього носія, який містить спеціалізоване програмне забезпечення для зчитування залишкових даних із RAM. Інший підхід полягає у вилученні модулів пам'яті з одного комп'ютера та підключенні їх до іншого пристрою, здатного зчитувати інформацію. У випадку криптографічних систем це може дозволити відновити ключі шифрування, які тимчасово зберігалися у пам'яті під час виконання операцій. Цей метод атак найбільш ефективний проти систем, які не шифрують оперативну пам'ять або не очищають її перед вимкненням живлення. Операційні системи та програми зазвичай зберігають у RAM критичні дані, зокрема сесійні ключі, дані авторизації або навіть паролі користувачів. Навіть якщо диск комп'ютера зашифровано, залишковий заряд у RAM може містити ключі, необхідні для розшифрування, що дозволяє обійти захист.

Захист проти таких атак включає кілька заходів. Один із них — це використання механізмів, які шифрують пам'ять у реальному часі, наприклад, технології Trusted Platform Module (TPM) або сучасні апаратні рішення для захищеної пам'яті. Інший підхід — швидке очищення оперативної пам'яті під час вимкнення живлення, щоб мінімізувати ризик залишкового заряду. Крім того, фізична безпека комп'ютера, включаючи захист від несанкціонованого

доступу до обладнання, залишається ключовим фактором у боротьбі з cold boot-атаками.

Також якщо розглядати цикли вимикання та запуску важливо згадати про атаки на завантажувач. Під час увімкнення або перезавантаження комп'ютера або вбудованого пристрою запускається базова система вводу-виводу (BIOS) — частина вбудованого програмного забезпечення, розташована в ROM або EPROM, яка постачається разом із пристроєм. Після цього BIOS шукає завантажувальний пристрій, у якому знаходяться код завантажувача або менеджера завантаження. Функцією завантажувача є різними, залежно від ОС, але основною є завантаження операційної системи. Після цього ОС візьме на себе всю роботу системи. Новіші системи, такі як Windows 11, постачаються з уніфікованим розширюваним інтерфейсом прошивки (UEFI) BIOS. Специфікація UEFI визначає, як мікропрограма, що одразу при запуску отримує доступ до всього обладнання.

Якщо BIOS (або UEFI) заражено шкідливою підпрограмою, уся система/мережа буде скомпрометована, оскільки ці програми мають повний доступ до всіх системи. Уразливості в Grand Unified Bootloader (GRUB), популярному завантажувачі ОС на базі Unix, мали серйозні наслідки. У 2020 році BleepingComputer повідомив про серйозну вразливість, відому як помилка завантажувача BootHole GRUB, яка перериває процес завантаження ОС. Це вплинуло на всі ОС Unix і деякі ОС Windows[4]. Більш тривожним є те, що механізм перевірки безпечного завантаження не зміг запобігти атаці. Цей тип зловмисного програмного забезпечення може бути активовано під час завантаження та воно здатне перебувати в стані спокою протягом певного періоду часу.

Одним із поширених способів атаки є підміна завантажувача. Зловмисник може замінити оригінальний завантажувач модифікованим, який містить бекдор або шкідливий код. Це дозволяє отримувати доступ до даних на

пристрої або навіть виконувати зловмисний код до запуску операційної системи, обходячи її захист. Такий тип атак особливо небезпечний для систем без перевірки цілісності завантажувача. Ще один вид атаки — атака через незахищений завантажувальний режим. Багато пристроїв мають опцію "розблокованого" або "розширеного" режиму завантаження для розробників, яка дозволяє змінювати завантажувач або завантажувати сторонні операційні системи. Якщо цей режим не захищений паролем або криптографічною перевіркою, зловмисник може скористатися ним для модифікації системи або викрадення даних. Rootkit-атаки на завантажувач також є серйозною загрозою. У цьому випадку зловмисний код впроваджується в завантажувач, що дозволяє йому залишатися прихованим від операційної системи та антивірусного програмного забезпечення. Rootkit може працювати на рівні ядра системи, виконуючи небажані дії та отримуючи доступ до конфіденційної інформації.

Для захисту від атак на завантажувач використовують різні методи. Один із них — цифровий підпис завантажувача, який дозволяє перевіряти його цілісність перед запуском. Якщо завантажувач був змінений, система відмовиться запускати його. Також популярним є використання технологій на кшталт Secure Boot, яка гарантує, що завантажувач є лише довіреним програмним забезпеченням. Ще один спосіб захисту — блокування завантажувача або обмеження доступу до режиму розробника, що значно ускладнює спроби модифікації. Атаки на завантажувач є важливим нагадуванням про те, що безпека інформаційних систем повинна охоплювати всі рівні, починаючи з найнижчих шарів програмного забезпечення. Оскільки завантажувач є першим етапом завантаження системи, його компрометація може відкрити шлях до повного захоплення контролю над пристроєм.

1.2.4 Атака типу RowHammer

Оскільки виробничі процеси з роками стають ефективнішими, виробники можуть збільшують щільність пам'яті чіпів DRAM, що призводить до зниження собівартості виробництва, нижчого споживання енергії для тієї самої ємності пам'яті та більшої густини пам'яті для тієї самої кремнієвої площі. Основний тренд цього зростання можна бачити на рисунку 1.10, що показує лінійки пам'яті трьох основних виробників чіпів. Як бачимо дана тенденція може бути зведена до двох основних характеристик — збільшення кількості комірок пам'яті та зменшення площі чіпів на платі, що відповідно призводить до збільшення щільності пам'яті.

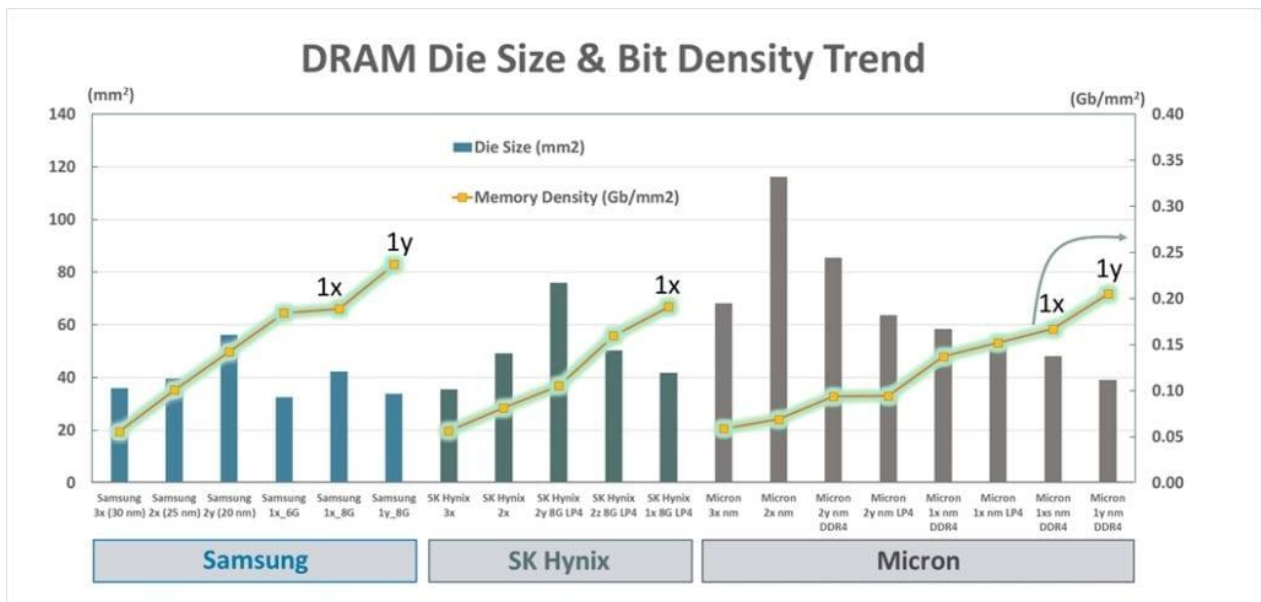


Рисунок 1.10 — Тенденції розвитку оперативної пам'яті

Однак збільшення щільності банків DRAM також призводить до того, що при зближенні розташування рядів даних один до одного, знижується допустимий рівень шуму та спотворень для бітових комірок. Це призводить до посилення паразитних електричних взаємодій між сусідніми комірками і відкриває шлях до нових значних атак на пам'ять, однією з яких і є RowHammer. Типова атака RowHammer передбачає частий повторний доступ до певного рядка в мікросхемі банку DRAM. Цей агресивний доступ може

ненавмисно спровокувати зміну значення бітів у сусідніх рядках жертви, спричиняючи зміни в роботі активних програм в пам'яті. RowHammer — це вразливість у безпеці DRAM, яка дозволяє зловмиснику змінювати дані, що зберігаються в комірках пам'яті. Повторно та швидко активуючи певний ряд, може призводити до витоку заряду з конденсаторів клітин-жертв у сусідніх рядках [13]. Було висунуто три потенційні причини цього явища:

1. Створення мосту між сусідніми рядками: дослідження професора Ал-Арса та ін [14], продемонструвало утворення провідних каналів між окремими проводами та конденсаторами в DRAM. Крім того, дослідження показує, що часте перемикання рядків слів може прискорити потік заряду між двома елементами, з'єднаними таким магнітним містком.

2. Електромагнітний зв'язок: зміна напруги в лінії слів (WL) може викликати шум у сусідній лінії слів через електромагнітний зв'язок, викликаючи витік заряду з клітин-жертв [15].

3. Ін'єкція гарячого носія: тривале перемикання рядка слів може призвести до ін'єкції гарячого носія [16]. Ін'єкція гарячих носіїв у сусідні ряди може посилити витік заряду з постраждалих клітин. Цей витік заряду може призвести до того, що деякі клітини-жертви не зможуть зберегти свій заряд протягом встановленого інтервалу оновлення.

Отже, це явище призводить до зміни збережених даних, що призводить до перевороту біта з 0 до 1 або навпаки, як показано на рисунку 1.11.

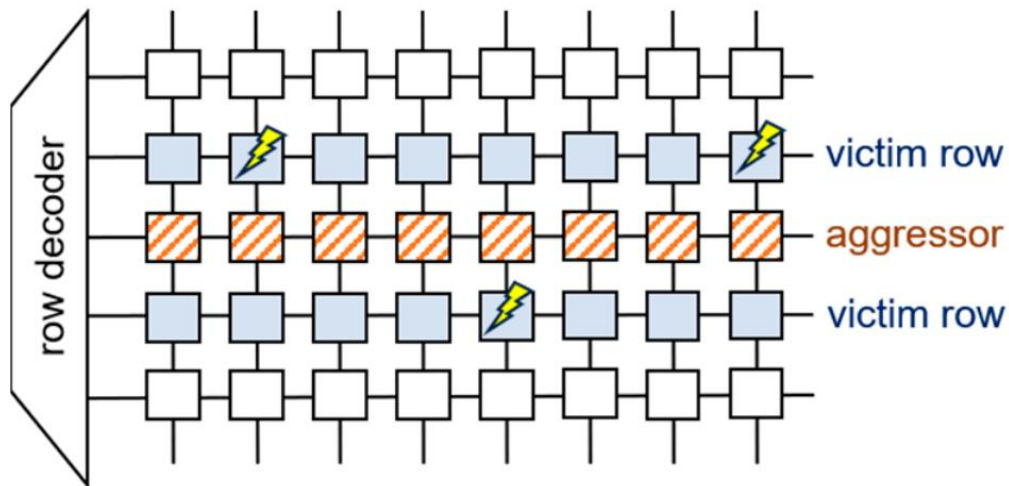


Рисунок 1.11 – Ілюстрація механізму атаки RowHammer

В першій статті, що описує атаку професор Кім та ін. [1] виявили, що коли ряд активовано, конденсатори сусідніх рядів зазнають паразитного впливу, що спричиняє невеликий витік заряду, але достатній щоб повпливати на звичайний хід роботи та втрати заряду. Як і дослідники до нього автор припустив кілька причин, що це спричиняють: електромагнітний зв'язок, як вже вказано вже був доведено, що викликає небажані взаємодії між сусідніми рядками слів; між сусідніми проводами та/або ємностями можуть утворюватися мости для прискорення втрати заряду під час перемикання WL; неодноразове перемикання WL протягом тривалих періодів часу може назавжди пошкодити рядок і його сусідів введенням гарячого носія, змінюючи властивість транзисторів доступу або вводячи заряди в сусідні конденсатори. На той час автор не зміг точно визначити основну причину цього питання. Нещодавно професор Янг та інші. [13] продемонстрував, що причиною цієї проблеми є те, що при перемиканні станів WL вгору та вниз, під лінією слів формуються так звані «пастки заряду» (1 на рисунку 1.12), захоплюючи в себе деякі негативні заряди, які випромінюються підкладкою. Потім такі заряди мігрують до найближчих конденсаторів (2 на рисунку 1.12), викликаючи витік заряду, як показано на рисунку 1.12 і на частині 3.

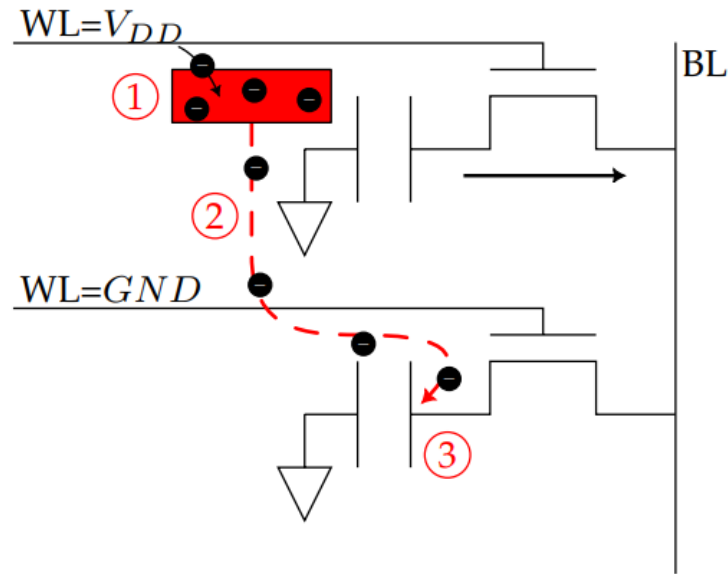


Рисунок 1.12 – Процес захоплення заряду і розряду комірок пам'яті DRAM

Сама по собі ця проблема не є небезпечною, оскільки витік заряду врахований в архітектурі DRAM. Наступний REF або ACT у цьому рядку має оновити конденсатор і значення в ньому. Однак, якщо цю операцію повторити достатньо разів, щоб заряд конденсатора опустився нижче порогу, за якого він може підвищити напругу BL під час ACT і біт буде неправильно інтерпретовано під час наступного циклу оновлення. Контролер вважатиме, що значення було змінено і заряд вичерпано, що призведе до пошкодження даних. Еволюція заряду конденсатора за нормальної роботи та за збурень від повторюваних ACT у сусідніх рядках проілюстрована на рисунку 1.13.

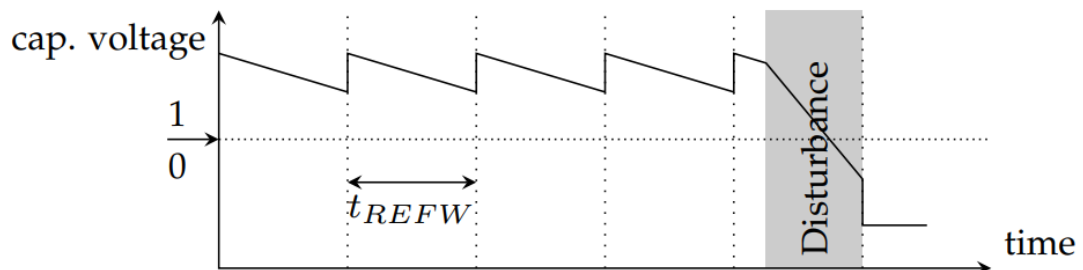


Рисунок 1.13 – Еволюція напруги конденсатора комірки пам'яті за нормальної поведінки та при збуреннях.

Тут ми визначимо поняття – *рівень збурення* (PЗ) рядка як кількість команд активації АСТ, які були виконані сусідами з моменту останнього дозарядження або команди оновлення. Зміни значення бітів з’являються лише після того, як рівень збурення досягає певного значення. Ми називаємо це значення RowHammer порогом (T_{RH}). Це в першу чергу визначається технологією: через менші конденсатори та ближче розміщення лінії слів щільніша пам’ять матиме нижчий T_{RH} . Значення T_{RH} було значно скорочено протягом багатьох років дослідження атаки і розвитку пам’яті, перейшовши від 138k АСТ для створення біт-фліпу на старішій DDR3 до 9,6k для недавнього LPDDR4 [15].

Крім того, багато факторів можуть додавати своє значення на вплив міжклітинних збурень та результуючу зміну значення бітів. У той час як конденсатори можуть розряджатися лише під впливом перешкод між комірками, бітове перетворення може відбуватися в обох напрямках ($1 \rightarrow 0$ або $0 \rightarrow 1$) залежно від того, як напруга інтерпретується логічними схемами. Однак, оскільки цей напрямок визначається логічними схемами, він завжди сталий для бітів однієї моделі пам’яті. Згідно з експериментами, проведеними Кімом та ін. у 2020 році [15], шаблон даних у пам’яті (тобто, позиції 1-ць і 0-ів в смугах рядків чи смугах стовпців), здається, мають дуже важливий вплив на наявність біт-фліпів після атаки. Це дослідження показує, що шаблон атаки, який створює найбільшу кількість біт-фліпів, суттєво відрізняється в різних поколіннях DRAM, але не так сильно у різних виробників.

Багато дослідницьких груп досліджували різні шаблони повторюваного доступу, такі як односторонні, двосторонні або багатосторонні шаблони доступу, намагаючись підвищити ймовірність індукції перевертання бітів або обійти захисні механізми RowHammer, такі як цільове оновлення рядка (TRR) [10]. Попередні дослідження підкреслювали важливість і складність підготовчого етапу перед доступом до рядка DRAM, який залежить від розуміння операційної системи та базової архітектури процесора. Наприклад,

ідентифікація суміжності рядків DRAM для атаки RowHammer вимагає профілювання пам'яті, оскільки послідовні адреси пам'яті, які використовує програма, не вирівнюються лінійно з рядками DRAM. Більше того, швидка повторювана активація рядка DRAM для виклику перевороту бітів вимагає обходу ієрархії кешу, яка використовується для мінімізації доступу до пам'яті DRAM.

Збурення між клітинами можуть бути викликані навмисно, при цьому мета агресора полягає в тому, щоб направити сусідам жертви в рядках достатню кількість команд активації, до оновлення, щоб змінити якісь біти. Основними перешкодами для атакуючого є два елементи:

- *буфер рядка*: він містить останній активований рядок, і доступ до рядка, що зберігається в ньому, ініціює потрапляння по рядку (row hit). Слід уникати потраплянь у рядок, оскільки вони не призводять до АСТ безпосередньо у рядку DRAM, а отже, не спричиняє шум сусідам. Агресор повинен використовувати принаймні два ряди агресора одного банку, щоб уникнути потрапляння в рядки;
- *кеш-пам'ять*: вона містить дані, до яких найчастіше звертаються. Повторний доступ до одних і тих самих рядків створюватиме звернення до кешу, і жоден запит не досягне основної пам'яті. Агресор повинен або обійти кеш [16] або використовувати кеш механізми очистки для видалення рядків-агресорів із пам'яті кешу, використовуючи, наприклад, інструкції очищення кешу, коли вони доступні [1, 17], або техніку видалення кешу [4].

Пошкодження пам'яті за допомогою атаки RowHammer можна виконати за допомогою простого циклу збирання x86, представленого в лістингу 1.1. Позиції рядків-агресорів і клітин-жертв у банку DRAM, атакованому цією програмою, показано на рисунку 1.14. Рядки-агресори тут пофарбовані в червоний колір, а клітини-жертви в помаранчевий.

```

loop:
    mov (X), %eax ; read value from address X
    mov (Y), %eax ; read value from address Y
    clflush (X)   ; evict address X from cache
    clflush (Y)   ; evict address Y from cache
    mfence        ; wait for previous instructions to complete
    jmp loop      ; restart the loop

```

Лістинг 1.1 – цикл RowHammer на системі x86

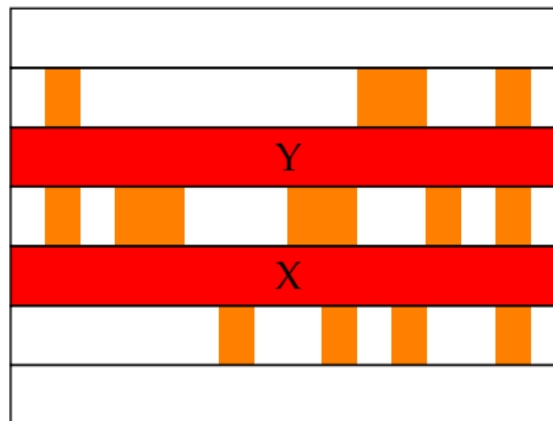


Рисунок 1.14 – Банк пам'яті піддається атаці на рядки X і Y.

Однак кеш-пам'ять не можна очистити вручну програмами рівня користувача на всіх системах. Процесори ARMv7-A обмежили інструкції з керування кеш-пам'яттю для привілейованих програм, а процесори ARMv8-A можна налаштувати так, щоб обмежити їх використання непривілейованими програмами. Інструкції з керування кешем також недоступні для веб-сторінок, оскільки доступні інструкції обмежені веб-браузером. У цих випадках агресор може покладатися на стратегії видалення кешу, які використовують політику заміни кеш-пам'яті для заміни використаної адреси на інші дані в кешах [4, 3], на DMA, щоб обійти кеш [16], або на вбудований графічний процесор, який має дуже простий кеш, з якого легко видалити дані [19].

Обхід ієрархії кешу є важливою передумовою для отримання прямого доступу до DRAM і подальшого виконання атаки RowHammer. Існуючі методи

обходу кешу можна загалом розділити на три основні підходи: очищення кешу, видалення кешу та тимчасовий обхід на основі сховища [22]. Очищення кешу, яке передбачає видалення даних із кешу, є найпростішим методом гарантування того, що кожен доступ до пам'яті походить із DRAM, а не з кешу ЦП. Для архітектури x86 дослідження [22] ефективно використовувало команду `clflush`, спеціально розроблену для очищення кешу. У контексті архітектури ARM, такої як ARMv7, може бути використаний системний виклик `cacheflush()`[23]. Дослідники також запропонували методи вилучення кешу, які не покладаються на інструкції `clflush` або `cacheflush()`, що іноді можуть бути недоступні на останніх архітектурах ЦП. Ці методи передбачають ретельну розробку шаблонів доступу до пам'яті для опосередкованого видалення рядків кешу, гарантуючи, що наступні звернення спрямовані на точно в пам'ять DRAM. Крім того, нетимчасові інструкції зберігання, такі як `movnti` або `movntdq` для архітектури x86, як показано професором Руні К. та іншими [24], пропонують альтернативний підхід до обходу кешу. Щоб гарантувати, що кожна інструкція тимчасового зберігання досягне мікросхеми DRAM пропонується очищати буфери комбінування запису. Цього можна досягти, виконавши інструкцію тимчасового зберігання з доступом до кешованої пам'яті за тією ж адресою, куди інструкція записує дані.

Також для розвитку атак було запропоновано методи виходу з пісочниці, тобто перехід до привілейованості програм, що хочуть отримати прямий доступ до пам'яті. JavaScript, що є основою теперішніх браузерних програм, працює в суворо ізольованому середовищі, обмежуючи доступ до файлів і системних служб. Це віртуалізує пам'ять переходячи при виконанні до таких понять, як віртуальні адреси та вказівники. Крім того, його точність синхронізації нижча за точність у машинному коді, що робить атаки RowHammer складними але не неможливими. Дослідження, проведене Даніелем К. та іншими [25], показало, що атаки на кеш-пам'ять на основі JavaScript можуть використовувати точність синхронізації по часу, щоб відрізнити потрапляння в кеш-пам'ять від промахів,

відкриваючи двері для атак часових атак по побічним каналам на пам'ять. У JavaScript, коли пам'ять виділяється, такі браузері, як Firefox і Google Chrome, призначають анонімну сторінку розміром 2 МБ для великого масиву даних операцій. Доступ до цього масиву з адресою розміром 4 КБ викликає помилки доступу, викликаючи стрибки з затримкою програми кожного разу, коли починається сторінка розміром 2 МБ. Ця відмінна поведінка від роботи всередині масиву дозволяє ідентифікувати кадр сторінки розміром 2 МБ. Маючи початкове знання про зміщення масиву, можна отримати і уявлення про молодший 21-й біт як віртуальної, так і фізичної адреси. Озброївшись цими даними, можна створити інструмент для перетворення віртуальних адрес у відповідні їм фізичні адреси.

Нещодавно також було продемонстровано, що атака RowHammer може бути розширена за межі безпосередніх сусідів агресорських рядів [15, 22]. Збільшення щільності пам'яті в новіших моделях DDR4 і вже DDR5 дозволило збуренням виходити за межі безпосередніх сусідів[15]. Але що важливіше, дослідження показало, що такі наведення можуть поширювати збурення від безпосереднього сусіда ряду агресора до його наступного сусіда, за два ряди від головного агресора [22]. Ця атака поєднує багато доступів до головних рядів агресора з кількома доступами до другорядних агресорів, що знаходяться поруч із першими. Йому вдається індукувати біт-фліпи в рядах жертв, безпосередніх сусідів вторинних рядів агресорів, на відстані двох рядів від основних агресорів.

Існують різні шаблони доступу до рядків для виконання атак RowHammer, які поділяються на три основні типи: односторонні, двосторонні та багатосторонні атаки. Під час односторонньої атаки RowHammer повторювані звернення до пам'яті зосереджені лише на одному рядку який знаходиться фізично поруч із цільовим рядком. Під час двосторонньої атаки RowHammer такі звернення відбуваються на два ряди пам'яті, фактично оточуючи цільовий

ряд жертви. Багатостороння атака RowHammer включає більше двох рядків пам'яті для обходу методів зменшення впливу RowHammer, таких як TRR [10].

Односторонній вид атак в основному спрямований на один ряд, який прилягає до цільового ряду жертв. Однак, коли контролер пам'яті використовує політику відкритої сторінки (open-page policy), де рядок пам'яті залишається буферизованим, доки не буде запиту на доступ до наступного рядка пам'яті, одностороння атака вимагає патерну доступу до двох окремих рядків в одному банку, щоб очистити вміст буфера рядків. Тож незважаючи на назву «одностороння атака», яка передбачає націлювання лише на одну ділянку пам'яті доступ все одно починається з двох цільових рядків і більше. На відміну від застарілої open-page policy, сучасні контролери пам'яті інтергують і більш просунуті політики доступу, які проактивно закривають доступ до рядків раніше, ніж це дійсно необхідно, з метою підвищення загальної продуктивності [28]. Спираючись на цей часовий зсув зсув, професор Даніель Г та інші в своєму дослідженні [29] запропонували нову техніку, відому як забивання в одному місці (one-location hammering). У цьому підході зловмисник виконує цикл Flush + Reload виключно на одній адресі пам'яті, працюючи на максимально можливій частоті. Ця безперервна активність фактично знову відкриває той самий рядок DRAM щоразу, коли контролер пам'яті намагається його закрити. Оскільки забивання в одному місці не має доступу до різних рядків в одному банку, воно має здатність обходити певні існуючі захисні механізми, призначені для виявлення оригінальних шаблонів односторонніх атак.

Двостороння ж атака типу RowHammer передбачає одночасне ураження і доступ до двох рядків пам'яті, фактично закриваючи з двох боків цільовий ряд жертви, як показано на рисунку 1.15. На відміну від односторонньої атаки, двосторонній підхід зазвичай може спричинити більшу кількість змін бітів на протилежні. Однак це вимагає глибшого розуміння віртуальних і фізичних

властивостей цільової пам'яті DRAM, оскільки два ряди, які піддаються атаці, повинні бути стратегічно розташовані по різні боки цільового ряду. Дослідження, обговорювані вище, також ефективно використовували двосторонню атаку, щоб викликати додаткові успішні зміни бітів за допомогою RowHammer.

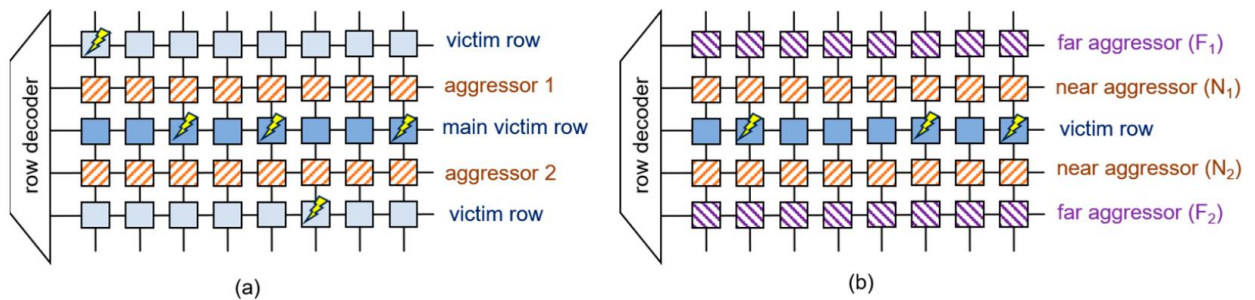


Рисунок 1.15 – Ілюстративний приклад (а) двосторонньої атаки та (б) атаки типу напівдубль

Враховуючи, що як одностороння атака, так і двостороння атака призначені для виклику спотворень значення, що зберігається в комірках пам'яті у сусідньому ряду, захисні механізми від RowHammer, такі як цільове оновлення (TRR), часто працюють за припущенням, що пари агресор–жертва справді є суміжними. Щоб уникнути виявлення, дослідники також продемонстрували концепцію багатосторонніх атак, прикладом яких є такі методи, як half-double [30] і TRRespass [10]. Метод напівдубль спочатку спрямований на двох далеких агресорів, F₁ і F₂, як показано на рисунку 1.15. Такий початковий вибір агресорських рядків гарантує, що в ряді жертви відбудеться лише незначний витік заряду, якого недостатньо, щоб викликати зміну бітів. Цікаво також і те, що напівдубль підхід використовує TRR у своїх цілях. Завдяки постійному доступу до F₁ і F₂ понад порогове значення, яке запускає TRR, TRR індукується в сусідньому рядку поблизу агресорів: N₁ і N₂. Згодом ця дія передбачає доступ до рядів ближніх агресорів, які, у свою чергу, впливають на ряди жертви, що призводить до зміни бітів. Крім того, зараз досліджується бластер [31] з відстанню ряду 4 від ряду жертв.

TRRespass також показав метод багатостороннього фаззера RowHammer, який виявляє шаблони доступу для атаки, ефективні в умовах TRR, на основі спостереження, що сучасні реалізації TRR зазвичай слабкі у виявленні атак з багатьма рядами агресора [32]

1.3 Сучасні підходи до захисту пам'яті

Незважаючи на наполегливі та постійні дослідницькі зусилля щодо розробки стратегій захисту від атак RowHammer, уразливості залишаються поширеними. Зменшення технологічних вузлів у мікросхемах DRAM посилює загрозу, дозволяючи атакам RowHammer досягати успіху з меншою кількістю активацій рядків. Це підкреслює необхідність переоцінки та посилення існуючих механізмів захисту. У цьому розділі ми класифікуємо стратегії захисту від RowHammer за трьома областями залежно від того, де застосована техніка пом'якшення: програмне забезпечення, контролер пам'яті та DRAM, як показано в таблиці 3. Згодом ми аналізуємо стратегії в кожній категорії за критеріями концепції захисту, відстеження механізми та засоби правового захисту. Розширені технологічні вузли та більш висока щільність кремнію підвищують сприйнятливність DRAM до атак RowHammer, оскільки зменшений відстань між осередками значно зменшує кількість молотків, необхідних для перевертання бітів.

RowHammer використовує конструкцію DRAM з одним конденсатором на біт, щоб запускати зміну бітів у сусідніх комірках через повторювані звернення до рядків пам'яті. Ця вразливість дозволяє зловмисникам маніпулювати даними, відновлювати конфіденційну інформацію та виводити з ладу процеси чи системи. Вперше виявлені в 2014 році нові варіанти RowHammer продовжують націлюватися на DRAM, успішно обходячи методи безпеки, такі як код виправлення помилок (ECC) і оновлення рядків транзакцій (TRR). Ефективний захист DRAM від RowHammer вимагає багаторівневої реалізації

надійних методів безпеки на системному рівні, від шифрування та обфускації до примусової ізоляції даних і розширених схем виправлення помилок. Однак це легше сказати, ніж зробити, оскільки контрзаходи можуть потенційно вплинути на потужність, продуктивність і площу (PPA). Тому інженери повинні оцінити компроміси безпеки PPA разом із ключовими функціями та компонентами на початку процесу проектування.

1.3.1 Основні поняття

Традиційно під час розробки мікро чіпів враховуються лише вплив на їх PPA. Однак нещодавні апаратні атаки, включно з RowHammer, Meltdown і Spectre, а також атаки, що використовують функції DVFS для впровадження помилок програмного забезпечення, наприклад `clkscrew` і `Plundervolt`, підкреслюють важливість визначення пріоритетів безпеки під час процесу проектування. Часто саме нові функції, додані для підвищення продуктивності, створюють плацдарм для атак, оскільки технологія DRAM з часом зменшується, щільність і продуктивність покращуються, сприйнятливість до RowHammer зростає. Щоб вирішити ці проблеми, були запропоновані різні методи шифрування та обфускації для захисту DRAM від атак RowHammer. Якщо зашифрувати або обфускувати свої дані, а потім хтось забиває рядок атакою і спричиняє зміну бітів, це не може вплинути на один відповідний біт початкової зашифрованої інформації.

Головним зараз має бути те, що питання безпеки повинні бути частиною процесу проектування, починаючи з архітектурного рівня. Цілісний аналіз системи також дозволяє команді розробників зменшити вплив на погіршення безпеки через оптимізацію PPA. Якщо ж виявляти кожну слабкість окремо, тоді накладні витрати зростуть набагато більше, тому що техніки захисту можуть збігатися чи виключати одна одну. Тож слід дивитися на вищий рівень безпеки. Важливо дивитися знизу вгору на найвищий рівень, а потім керувати

програмою безпеки з нього. Так як проектування навпаки викликає багато складності та проблем.

Зрештою, загальносистемні апаратні і програмні рішення у поєднанні із суворим контролем доступу та примусовою ізоляцією даних є поки найбільш ефективним методом протидії експлойтам, таким як RowHammer. У будь-якому мультитенантному або загальному середовищі для ізоляції даних потрібні контейнери. Дані також повинні бути призначені, наприклад, потоку процесора, де вони не можуть бути прочитані іншим потоком. Звичайно, це не може бути просто програмне забезпечення. Потрібен апаратний захист базового рівня. Інакше захист програмного забезпечення буде порушено.

1.3.2 Апаратні рішення: оновлення DRAM, ECC

Одним із найпростіших і водночас ефективних способів захисту від атак RowHammer є збільшення частоти оновлення комірок пам'яті (Refresh Rate Increase). Цей метод базується на розумінні того, що атака RowHammer реалізується шляхом багаторазового зчитування певного рядка в пам'яті, що спричиняє витік заряду з сусідніх комірок. Чим частіше виконується оновлення даних у пам'яті, тим менше часу залишається для того, щоб витік заряду призвів до бітових помилок. Стандартна динамічна оперативна пам'ять (DRAM) використовує механізм періодичного оновлення, при якому кожен рядок пам'яті перезаписується через певний проміжок часу. Наприклад, у традиційних модулях DRAM оновлення всіх рядків здійснюється кожні 64 мілісекунди (ms). У відповідь на загрозу RowHammer виробники пам'яті та розробники контролерів почали впроваджувати механізми, які дозволяють скоротити цей інтервал удвічі або навіть учетверо (наприклад, 32 ms або 16 ms). Це суттєво ускладнює реалізацію атаки, оскільки потенційний злоумисник не встигає виконати достатню кількість зчитувань, щоб спричинити витік заряду в сусідніх комірках. Збільшення частоти оновлення може реалізовуватися на

рівні контролера пам'яті, який керує циклом оновлення для всіх підключених модулів DRAM. У сучасних системах контролери пам'яті підтримують адаптивні механізми керування частотою оновлення. Наприклад, коли система виявляє підозріло частий доступ до певних рядків, вона може динамічно збільшувати частоту оновлення саме для цих рядків, зменшуючи ймовірність витоку заряду. Такий підхід знижує вплив на загальну продуктивність пам'яті, оскільки оновлення відбувається вибірково для потенційно вразливих комірок, а не для всієї пам'яті.

Хоча збільшення частоти оновлення є ефективним методом захисту, він має і свої недоліки. Головний з них – це підвищене енергоспоживання. Операція оновлення вимагає додаткових витрат енергії, а часте повторення цього процесу збільшує загальне навантаження на систему живлення та охолодження. Особливо це стає проблемою для мобільних пристроїв, серверів та систем із великою кількістю встановленої пам'яті, де навіть невелике зростання енергоспоживання може мати критичний вплив. Ще одним викликом є вплив на продуктивність. Хоча оновлення пам'яті займає відносно малий проміжок часу, його надмірне збільшення може призводити до коротких переривань доступу до даних, що може негативно позначитися на швидкості роботи програм, які активно взаємодіють із пам'яттю. Саме тому сучасні рішення комбінують збільшення частоти оновлення з іншими механізмами, такими як Target Row Refresh (TRR), які дозволяють застосовувати цей метод лише до потенційно вразливих ділянок пам'яті, зменшуючи негативний вплив на продуктивність.

Незважаючи на ці обмеження, збільшення частоти оновлення залишається одним із ключових методів захисту, який використовується в сучасних системах, оскільки не потребує суттєвих змін у архітектурі пам'яті. Багато виробників DRAM, включаючи Samsung, SK Hynix та Micron, активно вдосконалюють свої технології оновлення, додаючи адаптивні алгоритми, які

забезпечують баланс між захистом від RowHammer та ефективністю роботи системи.

Другим із ключових підходів до апаратного захисту пам'яті DRAM від атак типу RowHammer є використання механізмів корекції помилок (ECC – Error-Correcting Code). Технологія ECC застосовується в серверних, високопродуктивних обчислювальних системах та критично важливих додатках, де надійність пам'яті є пріоритетом. Вона дозволяє виявляти та виправляти помилки, що виникають у комірках пам'яті, включаючи ті, що можуть бути спричинені атаками RowHammer.

У типовій реалізації ECC-пам'яті до кожного слова даних додаються додаткові біти перевірки, які зберігають інформацію про парність або більш складні коди корекції. Одним із найпоширеніших методів є код Хеммінга (Hamming Code), який дозволяє виправляти одноразові помилки та виявляти подвійні. Для захисту від RowHammer сучасні реалізації ECC використовують більш потужні коди, такі як SECDED (Single Error Correction, Double Error Detection), що дозволяють виправляти помилки в одному біті і виявляти помилки у двох. Це важливо, оскільки атака RowHammer може спричиняти множинні помилки, які потребують складніших алгоритмів корекції. Розширені реалізації ECC у новітніх модулях DRAM включають можливості багаторівневого контролю, такі як символне ECC (Chipkill) або технології, що розподіляють контрольні біти між кількома чіпами пам'яті. Це дозволяє не лише виправляти помилки в окремих бітах, а й компенсувати збій цілих чіпів або окремих комірок, які можуть піддаватися атаці RowHammer. Наприклад, у модулях пам'яті DDR5 застосовуються вдосконалені варіанти ECC, які можуть виправляти більше ніж одну помилку на комірку завдяки внутрішнім алгоритмам контролю.

Однак використання ECC не є панацеєю від атаки RowHammer. У деяких випадках, коли RowHammer спричиняє множинні помилки в одній комірці або

розподілені помилки у кількох різних банках пам'яті, навіть розширений ЕСС може не впоратися з корекцією всіх збоїв. Тому сучасні системи безпеки часто комбінують ЕСС з іншими методами захисту, такими як вибіркове оновлення рядків (TRR – Target Row Refresh), щоб значно зменшити ризик успішної атаки. Загалом, ЕСС залишається одним із найважливіших механізмів апаратного захисту від атак RowHammer. Його ефективність залежить від типу використаних кодів корекції, рівня інтеграції з контролерами пам'яті та поєднання з іншими заходами безпеки. У сучасних системах високої продуктивності вдосконалені механізми ЕСС у поєднанні з оновленими архітектурними рішеннями допомагають суттєво зменшити вразливість пам'яті до подібних атак.

1.3.3 Програмні методи: алгоритми перевірки й обмеження доступу

Враховуючи те, що атака реалізується на апаратному рівні, її виявлення та запобігання є складним завданням, якщо ми говоримо про ситуації коли немає можливості змінювати апаратну архітектуру. Однак існує низка програмних методів захисту, які можуть допомогти мінімізувати ризики RowHammer. Одним із ключових підходів є використання алгоритмів перевірки цілісності пам'яті та виявлення підозрілої активності. Програмні методи, засновані на алгоритмах перевірки, передбачають моніторинг операцій із пам'яттю та аналіз закономірностей доступу до комірок. Оскільки атака RowHammer базується на багаторазовому читанні одного й того ж рядка пам'яті за короткий проміжок часу, одним із найпростіших методів є відстеження частоти доступу до окремих рядків. Якщо система виявляє аномально велику кількість зчитувань із певного рядка, це може сигналізувати про потенційну атаку. У таких випадках операційна система або мікропрограми можуть вжити запобіжних заходів, наприклад, ініціювати примусове оновлення вразливих рядків або змінити розташування критичних даних у пам'яті.

Ще одним ефективним алгоритмічним методом є використання перевірки контрольних сум і хеш-функцій для виявлення несподіваних змін у даних. Наприклад, операційна система може періодично обчислювати контрольні суми певних областей пам'яті й порівнювати їх із попередньо збереженими значеннями. Якщо виявляються невідповідності, це може свідчити про несанкціоновані зміни, викликані атакою RowHammer. Цей підхід може використовуватися у високо захищених системах, де критично важливо забезпечити цілісність даних, наприклад, у банківських системах або державних інформаційних ресурсах. Щоб уникнути значних накладних витрат на постійний моніторинг усієї пам'яті, сучасні алгоритми використовують техніки вибіркового тестування. Наприклад, система може розділяти пам'ять на окремі сегменти й здійснювати перевірку випадкових ділянок за допомогою алгоритмів статистичного аналізу. Це дозволяє ефективно виявляти потенційні атаки, не сповільнюючи роботу всієї системи. Подібний підхід застосовується в багатьох антивірусних системах і механізмах виявлення аномалій у програмному забезпеченні.

Додатково, деякі сучасні операційні системи та гіпервізори використовують алгоритмічні методи управління пам'яттю, які запобігають надмірному доступу до одного й того ж рядка. Це може включати, наприклад, випадкове переміщення важливих структур у пам'яті або спеціальні політики планування використання оперативної пам'яті, які перешкоджають занадто частим доступам до конкретних рядків. Таким чином, навіть якщо атакуючий спробує реалізувати атаку RowHammer, йому буде складно передбачити, які саме комірки пам'яті будуть уразливими, що значно знижує ефективність атаки.

Також одним із найбільш ефективних способів протидії атакам типу RowHammer є програмні методи, засновані на обмеженні доступу до пам'яті. Такі методи зменшують можливість зловмисника впливати на фізичну

структуру DRAM за допомогою частих операцій зчитування та запису, які можуть призвести до неконтрольованої зміни бітів у сусідніх комірках пам'яті. Основна ідея цього підходу полягає в обмеженні кількості та частоти звернень до певних областей пам'яті, щоб атакуючий не міг виконати достатню кількість операцій для реалізації атаки RowHammer. Один із простих способів реалізації обмеження доступу полягає у впровадженні контролю кількості звернень до конкретних рядків пам'яті. Наприклад, операційна система або гіпервізор можуть відстежувати частоту звернень до кожного рядка DRAM і блокувати доступ до тих, які використовуються надто часто. Якщо певний процес намагається багаторазово зчитувати або записувати дані в ту саму область пам'яті, система може тимчасово припинити його виконання або перенести його дані в іншу частину пам'яті, що робить атаку менш ефективною.

Ще одним дієвим механізмом є обмеження привілеїв доступу до пам'яті на рівні програмного забезпечення. Високопривілейовані програми, такі як ядро операційної системи або віртуальні машини, можуть мати безпосередній доступ до всіх областей пам'яті, тоді як користувацькі програми та навіть деякі драйвери можуть працювати в умовах обмеженого доступу. Це означає, що зловмисник, який працює з додатком на рівні користувача, не матиме змоги безпосередньо взаємодіяти з фізичними адресами пам'яті, що значно знижує ризик реалізації атаки RowHammer. Деякі сучасні операційні системи застосовують методи рандомізації доступу до пам'яті. Наприклад, технологія ASLR (Address Space Layout Randomization) дозволяє змінювати розташування важливих даних у пам'яті щоразу при запуску програми. Це означає, що навіть якщо атакуючий знайшов спосіб впливати на певні області пам'яті, після перезапуску процесу або системи його знання стають застарілими, і він не зможе відтворити атаку.

Додатково можна реалізувати підхід до обмеження доступу, що полягає в використанні так званих «тіньових сторінок» пам'яті. Віртуалізаційні

технології можуть створювати ізольовані області пам'яті, в яких кожен процес бачить тільки власну віртуальну копію даних, тоді як справжня фізична пам'ять змінюється лише під контролем операційної системи. Це робить атаки RowHammer менш ефективними, оскільки зловмисник не може напряду маніпулювати реальними комірками пам'яті. Обмеження доступу також може бути реалізовано через механізми контролю виконуваного коду. Деякі сучасні процесори підтримують технології, які дозволяють розділяти пам'ять на сегменти для зберігання даних та виконання коду, забороняючи змінювати виконувані області пам'яті під час роботи програми. Це ускладнює експлуатацію RowHammer для зловмисників, які намагаються виконати атаку з метою модифікації критичних структур у пам'яті, таких як таблиці сторінок або ключові дані операційної системи.

Таким чином, програмні методи захисту від RowHammer на основі обмеження доступу є важливим інструментом у боротьбі з цією вразливістю. Вони можуть бути реалізовані на рівні операційних систем, гіпервізорів та вбудованого програмного забезпечення, ефективно запобігаючи атакам без необхідності зміни апаратного забезпечення. Алгоритмічні методи перевірки є потужним інструментом захисту від RowHammer, особливо в тих випадках, коли апаратні механізми не можуть бути змінені. Вони дозволяють не лише виявляти підозрілі активності, але й активно реагувати на загрози шляхом застосування захисних заходів у реальному часі. Завдяки комбінації моніторингу доступу до пам'яті, контрольних сум, вибіркового тестування та динамічного керування доступом до даних, сучасні системи можуть значно зменшити ризики експлуатації RowHammer без необхідності модифікації обладнання. Використання таких підходів у поєднанні з іншими методами захисту, такими як алгоритмічний аналіз доступу до пам'яті або механізми виправлення помилок (ECC), дозволяє значно підвищити стійкість систем до атак, що базуються на феномені RowHammer.

1.3.4 Переваги та недоліки основних методів захисту пам'яті

Кожен із вищеописаних методів захисту має свої переваги та недоліки, які виробникам варто враховувати при розробці комплексної системи захисту від атак на пам'ять зокрема RowHammer.

Якщо говорити про Апаратний захист на основі ECC то його основна перевага в тому, що метод ECC (Error-Correcting Code) базується на вбудованих механізмах виявлення та виправлення помилок у пам'яті. Він дозволяє виявляти одиничні бітові збої та навіть виправляти деякі з них, що робить його ефективним у боротьбі з наслідками атаки RowHammer. Головна перевага такого підходу полягає в тому, що він працює на апаратному рівні і не потребує модифікації програмного забезпечення. ECC-модулі широко використовуються в серверних і критично важливих системах, що підтверджує їхню ефективність. Крім того, цей метод не залежить від операційної системи чи конкретних програм, оскільки працює безпосередньо в контролері пам'яті. Незважаючи на ефективність ECC в таких підходах, він має свої обмеження. Найголовніше – стандартний ECC може виправляти лише один біт у слові пам'яті та виявляти двобітові помилки, але атака RowHammer може спричинити множинні бітові помилки, які виходять за межі можливостей базового ECC-коду. Щоб захистити пам'ять від таких атак, потрібно використовувати розширені схеми ECC, що збільшує апаратні витрати та знижує продуктивність системи. Крім того, модулі пам'яті з підтримкою ECC дорожчі, ніж звичайні DRAM-модулі, що робить цей метод менш доступним для споживчих пристроїв.

Збільшення частоти оновлення (refresh rate) є ефективним і універсальним способом захисту від RowHammer, оскільки цей метод зменшує час, протягом якого зломисник може впливати на рядки пам'яті шляхом багаторазових доступів. При підвищеній частоті оновлення комірки пам'яті періодично відновлюють свої заряди до того, як внаслідок атаки RowHammer в них можуть виникнути помилки. Цей підхід є повністю апаратним і може бути реалізований

без значних змін у програмному забезпеченні. Основним недоліком ж збільшення частоти оновлення є значне зростання енергоспоживання пам'яті. Оскільки DRAM споживає енергію під час оновлення, підвищена частота таких операцій призводить до зменшення енергоефективності системи, що особливо критично для мобільних пристроїв і вбудованих систем. Також, часті оновлення можуть знижувати загальну продуктивність пам'яті, оскільки контролер буде змушений витрачати більше часу на виконання цих операцій замість обробки запитів від процесора. Окрім того, такий підхід не усуває сам механізм RowHammer, а лише ускладнює його експлуатацію, що означає, що злоумисники можуть знаходити нові способи обходу цієї технології.

Алгоритмічні методи захисту можуть виявляти підозрілу активність у пам'яті та запобігати атакам без необхідності модифікації апаратного забезпечення. Вони дозволяють виявляти часті доступи до певних рядків та відповідно реагувати, наприклад, виконуючи примусове оновлення сусідніх рядків. Головна перевага такого підходу – його гнучкість, оскільки програмні алгоритми можуть бути оновлені або покращені без необхідності заміни обладнання. Також вони можуть використовуватися в уже існуючих системах без додаткових апаратних витрат. Основним недоліком тут є те, що алгоритмічні методи потребують значних обчислювальних ресурсів, оскільки вимагають постійного моніторингу активності пам'яті, що може впливати на продуктивність системи, при цьому маючи низький рівень виявлення на Ват витраченої енергії. Крім того, алгоритми можуть давати хибні спрацьовування, обмежуючи продуктивність легітимних програм або створюючи додаткове навантаження на контролер пам'яті. Додатковим недоліком є те, що такі програмні методи можуть бути повністю неефективними проти нових варіантів атаки RowHammer, які модифікують схему доступу до пам'яті, щоб уникати виявлення.

Методи, що обмежують доступ до пам'яті, можуть значно ускладнити виконання атаки RowHammer. Вони дозволяють контролювати частоту доступу до пам'яті та обмежувати можливість користувацьких програм напряму взаємодіяти з критично важливими областями DRAM. Це особливо ефективно у віртуалізованих середовищах, де гіпервізор або операційна система можуть ізолювати пам'ять процесів один від одного. Використання таких методів також дозволяє динамічно оновлювати політики безпеки без необхідності зміни апаратного забезпечення. Головний же мінус цього підходу полягає у складності його реалізації. Обмеження доступу до пам'яті вимагає модифікації ядра операційної системи або гіпервізора, що може вплинути на сумісність із програмами та апаратним забезпеченням. Крім того, такі методи можуть викликати значне падіння продуктивності через необхідність постійного контролю за доступами до пам'яті. Ще одним недоліком є те, що атакуючі можуть знаходити способи обходу таких обмежень, наприклад, використовуючи побічні канали або маніпулюючи привілеями доступу.

Жоден із розглянутих методів не є ідеальним і не гарантує абсолютного захисту від атак RowHammer. Апаратні методи, такі як ECC та збільшення частоти оновлення, забезпечують високу надійність, але мають недоліки, пов'язані з вартістю та зниженням продуктивності. Програмні методи є більш гнучкими, проте потребують значних обчислювальних ресурсів та можуть бути вразливими до обхідних технік.

Найефективнішим підходом до захисту є комбіноване використання кількох методів. Наприклад, поєднання ECC з підвищеною частотою оновлення пам'яті може значно ускладнити реалізацію RowHammer. Так само програмні алгоритми перевірки разом із обмеженням доступу можуть створити багаторівневий захист, який значно зменшить ризик успішної атаки.

Таким чином, ефективний захист від RowHammer потребує комплексного підходу, що включає як апаратні, так і програмні рішення, адаптовані до конкретних умов використання та рівня безпеки системи.

Висновки до розділу 1

Отже, атака RowHammer залишається серйозною загрозою для сучасних DRAM-чипів, оскільки вона дозволяє змінювати дані у фізичній пам'яті без прямого доступу до них. Як було показано, існують різні методи боротьби з цією загрозою, кожен із яких має свої переваги та обмеження. Жоден із розглянутих методів не є ідеальним сам по собі, тому розуміння їхніх характеристик дозволяє розробникам будувати ефективні системи захисту, які найкраще відповідають конкретним умовам використання пам'яті.

Аналізуючи всі ці методи, можна зробити висновок, що кожен із них має свої сильні та слабкі сторони, а отже, для максимально ефективного захисту потрібно використовувати комбінацію декількох підходів. Наприклад, поєднання ECC та збільшеної частоти оновлення може значно ускладнити експлуатацію атаки RowHammer, хоча при цьому залишиться питання продуктивності та енергоспоживання. З іншого боку, програмні методи можуть забезпечити додатковий рівень безпеки, але вони ніколи не зможуть повністю замінити апаратні механізми захисту. Також варто враховувати, що виробники пам'яті постійно вдосконалюють технології виробництва DRAM, що зменшує ймовірність вразливостей, проте не усуває їх повністю. Останні дослідження показують, що навіть новітні модулі пам'яті можуть бути вразливими до нових варіантів атак RowHammer, що свідчить про необхідність постійного моніторингу ситуації та розробки нових захисних механізмів. Найбільш перспективними підходами для майбутнього можуть стати гібридні методи захисту, що поєднують апаратні та програмні рішення. Наприклад, розробка інтелектуальних контролерів пам'яті, які зможуть самостійно визначати атаки

RowHammer та динамічно змінювати стратегію оновлення або доступу до пам'яті, може стати наступним кроком у розвитку безпеки DRAM. Також досліджуються методи, засновані на машинному навчанні, які дозволять виявляти аномальну активність у пам'яті та прогнозувати потенційні атаки, аналізуючи поведінку програм у реальному часі. У підсумку, боротьба з RowHammer є складною і багатогранною проблемою, яка потребує комплексного підходу. Жоден із методів не є універсальним, тому найбільш ефективним рішенням є їхня комбінація, що дозволить враховувати всі можливі вектори атаки та зменшити ризик компрометації даних.

РОЗДІЛ 2

ТЕСТУВАННЯ НАЯВНИХ ВБУДОВАНИХ СИСТЕМ ЗАХИСТУ ПАМ'ЯТІ ТА ОЦІНКА ВРАЗЛИВОСТЕЙ

2.1 Створення платформи уніфікованого тестування засобів захисту пам'яті

Щоб покращити DRAM у всіх аспектах і подолати проблеми масштабування DRAM, дуже важливо експериментально зрозуміти роботу та характеристики справжніх мікросхем DRAM. Зокрема, такі загрози, як RowHammer, викликають збої на рівні апаратного забезпечення, які можуть бути використані для ескалації привілеїв або несанкціонованого доступу до даних. У зв'язку з цим розробка уніфікованої платформи для тестування засобів захисту пам'яті стає надзвичайно актуальною. Таким чином існує потреба в розробці експериментальної інфраструктури для ефективного та легкого тестування справжніх найсучасніших мікросхем DRAM.

Існуючі підходи до захисту пам'яті значно варіюються за своєю складністю, ефективністю та впливом на продуктивність системи. Наприклад, механізми на основі апаратного забезпечення, такі як ECC (Error-Correcting Code), пропонують високий рівень захисту, але мають обмеження щодо масштабованості та вимагають додаткових апаратних ресурсів. Програмні рішення, такі як моніторинг активності пам'яті або ізоляція процесів, забезпечують гнучкість, але можуть бути недостатньо швидкими для роботи в реальному часі. Відсутність уніфікованої платформи, яка дозволяла б порівнювати ці методи, аналізувати їх ефективність і виявляти вразливості в різних сценаріях, створює значний розрив між розробниками апаратного та програмного забезпечення. Мотивацією для створення платформи є необхідність стандартизації процесів тестування та оцінювання засобів захисту пам'яті. Універсальна платформа дозволить дослідникам і розробникам

моделювати реальні атаки, такі як RowHammer чи спроби переповнення буфера, оцінювати ефективність існуючих захисних механізмів і пропонувати нові підходи. Крім того, така платформа сприятиме зниженню витрат на розробку та інтеграцію захисних рішень, оскільки тестування буде проводитися в контрольованому середовищі з чітко визначеними параметрами. Актуальність розробки уніфікованої платформи також підкріплюється зростаючим використанням пам'яті у критичних системах, таких як фінансові, медичні чи військові обчислення. Помилки чи атаки на пам'ять у таких середовищах можуть призвести до серйозних наслідків, включаючи втрату даних, компрометацію конфіденційності або навіть фізичний збиток. Тому платформа, яка здатна забезпечити всебічне тестування засобів захисту пам'яті, має не лише наукове, але й практичне значення для безпеки сучасних технологій.

Масштабування технологічного процесу впливає на різні характеристики DRAM, такі як ємність комірки DRAM і ємнісні перехресні перешкоди, що негативно впливають на надійність, затримку та характеристики безпеки DRAM. Тому важливо ретельно проаналізувати ці ефекти, щоб зрозуміти, як змінюються реальні характеристики чіпа DRAM із масштабуванням технології. Використовуючи інфраструктуру тестування DRAM, попередні дослідження [34] розкривають ідеї, які надихають на нові механізми/методології для покращення затримки доступу до комірок DRAM, підвищення енергоефективності, розуміння та мінімізація проблеми RowHammer, а також розуміння збоїв збереження даних DRAM. Таке розуміння дуже важко, якщо не неможливо, розвинути без тестування справжніх мікросхем.

2.1.1 Побудова універсальної програмної основи

SoftMC [34] і Litex RowHammer Tester (LRT) [35] наразі є єдиними двома доступними платформами тестування DRAM на основі FPGA з відкритим

кодом. SoftMC — це інфраструктура з відкритим кодом, призначена для тестування DDR3. Він генерує трасування команд DDR3, які передаються головною машиною на плату FPGA, на якій працює SoftMC. Тим часом LRT — це платформа тестування RowHammer, спеціально створена для проведення експериментів на мікросхемах DDR3, DDR4 і LPDDR4. Він отримує послідовності команд від хост-машини та пересилає їх на пристрій DRAM, що тестується. Незважаючи на свою корисність, і SoftMC, і LRT стикаються зі значними обмеженнями, які не дозволяють їм відповідати вимогам універсальної, зручної та розширюваної платформи тестування DRAM на основі FPGA[.]

Основними недоліками SoftMC для цього дослідження є:

1. Обмеження передачі даних: SoftMC обмежує передачу даних DDR3 шляхом ініціалізації блоків кешу DRAM (512 біт) за допомогою повторюваних шаблонів на основі 8-бітних даних, введених користувачем. Це обмежує дослідження різноманітних шаблонів даних, важливих для інших експериментів з DRAM.

2. Недетерміновані затримки: виконання команд SoftMC DDR3 на основі трасування вводить непередбачувані затримки в кілька мікросекунд між послідовними трасуваннями команд. Це робить його непридатним для експериментів, таких як вимірювання енергії, які вимагають точного відліку часу протягом тривалих періодів (секунд або хвилин).

3. Застарілі компоненти: прототип дизайну SoftMC створено виключно для плати ML605 FPGA, яка більше не виробляється. Крім того, його середовище розробки спирається на застарілі апаратні та програмні компоненти.

4. Обмежена стандартна підтримка DRAM: інтерфейс DDR3 від SoftMC тісно інтегрований у дизайн апаратного забезпечення, йому бракує модульного чітко визначеного інтерфейсу. Як наслідок, адаптація SoftMC до нових

стандартів DRAM (наприклад, DDR4) потребує масштабної переробки апаратного забезпечення.

Тому платформу SoftMC для тестування новітніх плат і типів атак було відкинуто і перейдено до LRT. Відносно LRT то причинами невикористання цього фреймворку можна виділити такі:

1. Часові обмеження: LRT не може підтримувати часові затримки між послідовними командами DRAM коротшими за 10 нс, що значно обмежує його експериментальні можливості. Це обмеження робить його непридатним для досліджень зменшення затримки активації або скорочення часу запитів.

2. Повільна передача даних: LRT підтримує швидкість передачі даних лише до 1,9 Мбіт/с [35] між головною машиною та платою FPGA. Оскільки багато експериментів з DRAM включають передачу великих наборів даних для аналізу, цей повільний зв'язок створює значні накладні витрати.

Ці обмеження підкреслюють потребу в більш адаптивних, ефективних і розширюваних структурах тестування DRAM на основі FPGA бо як бачимо з причин вище жодна з існуючих інфраструктур тестування DRAM з відкритим кодом[36] не може бути використана для легкого та комплексного тестування найсучасніших мікросхем DDR4 та DDR5 DRAM. Тому важливо розробити універсальну, просту у використанні та розширювану інфраструктуру тестування DRAM з відкритим вихідним кодом, щоб уможливити експериментальні дослідження нових мікросхем DRAM. На основі цієї вимоги та нашої оцінки існуючих інфраструктур тестування DRAM ми визначаємо три ключові функції, які має надавати платформа тестування DRAM.

Інтерфейс без обмежень - інтерфейс DRAM зазвичай надає команди, які можна використовувати для доступу до комірок DRAM і керування ними (наприклад, READ і WRITE), стану банку DRAM (наприклад, АСТ рядка) і стану пристрою (наприклад, входу/виходу з режиму самооновлення). Щоб увімкнути багатфункціональне середовище тестування DRAM,

інфраструктура повинна забезпечувати необмежувальний інтерфейс прикладного програмування (API), який безпосередньо надає інтерфейс DRAM тестовій програмі. API не повинен накладати жодних часових обмежень або будь-яких обмежень на порядок команд DRAM, виданих до пристрою DRAM.

Простота використання - інфраструктура має бути легкою для розробки, для легкої інтеграції нових інтерфейсів DRAM і створення прототипів на нових платах FPGA, щоб легко створити робочий прототип за допомогою широко доступних програмних і апаратних засобів, і використовувати, щоб легко проектувати та проводити експерименти з DRAM.

Розширюваність - інтерфейси DRAM продовжують розвиватися в міру появи нових робочих навантажень і масштабування технології DRAM. Тому інфраструктура тестування DRAM повинна бути легко розширюваною для підтримки нових інтерфейсів DRAM (наприклад, HBM2, DDR5) для проведення експериментів на пристроях DRAM за останніми технологіями. Існуюча інфраструктура тестування DRAM не має принаймні однієї з цих функцій і не може використовуватися для легкого тестування найсучасніших мікросхем DDR4. Таким чином, існує потреба в інфраструктурі, яка забезпечує всі три функції для стимулювання майбутніх досліджень DRAM.

Стандартні інтерфейси DRAM визначають обмеження (наприклад, параметри синхронізації), яким підпорядковується контролер пам'яті, щоб гарантувати правильну роботу DRAM.[93] Ці обмеження за своєю конструкцією перешкоджають виявленню різноманітних незадокументованих операцій, на які технологія DRAM або конструкція чіпа можуть бути здатні за своєю суттю. Наприклад, нещодавні роботи[40] демонструють методи, які дозволяють використовувати кілька обчислювальних примітивів у DRAM (наприклад, масове копіювання даних, побітові операції, справжнє генерування випадкових чисел і фізичні неклоновані функції) з використанням стандартних пристроїв DRAM шляхом порушення параметрів синхронізації DRAM. Таку

незадокументовану поведінку DRAM можна виявити та продемонструвати лише шляхом тестування реальних мікросхем DRAM з використанням інфраструктур, які можуть порушувати параметри синхронізації DRAM. Тож першим пунктом на який потрібно звернути увагу при проектуванні тестувальної системи є Контролер пам'яті, оскільки саме він відповідає за встановлення обмежень та доступу до комірок з даними.

Контролер пам'яті зазвичай складається з двох модулів: планувальника (scheduler), що генерує та планує відповідні команди DRAM для обслуговування запитів пам'яті (наприклад, запити на завантаження та збереження) та фізичного інтерфейсу PHY (DFI), що маніпулює радіосигналами для передачі команд і даних через інтерфейс DRAM до мікросхем пам'яті. Інтерфейс DFI стандартизує зв'язок між контролером пам'яті та PHY, підвищуючи можливість повторного використання конструкцій контролерів пам'яті. Сигнали DFI класифікуються на дві великі групи - сигнали керування, які використовуються для видачі команд DRAM, які здебільшого аналогічні сигналам фізичного інтерфейсу (наприклад, сигнали cas, ras та we) та сигнали читання та запису даних, які використовуються для передачі даних через інтерфейс DRAM. Ці сигнали в DFI надають базовий інтерфейс зв'язку для контролера пам'яті і саме з ними ми будемо оперувати для тестування наявних чіпів пам'яті.

Використовуючи FPGA та вбудовані програмні функції, необхідні користувачеві для написання тестової програми DRAM можна збудувати початкову налагоджувальну програму. Схему тестувальної платформи можна побачити на рисунку 2.1, що відображає основні її компоненти і зв'язки. FPGA відіграє роль високочастотного ретранслятора команд до DRAM і відповідно зчитувача можливих помилок і стану бітів пам'яті. Для цього було використано високочастотні процесори та розпаралелювання програмної імплементації. Також для швидшого виконання тестів плата розширення має адаптер

інтерфейсу DRAM, який діє як посередник між внутрішнім конвеєром і фізичним інтерфейсом DRAM. Конвеєр DRAM виводить однократно закодований сигнал інструкції на зовнішній вихід. До прикладу, інструкція з першою командою ACT та третьою командою WRITE встановить найменш значущий біт сигналу (LSB) `dram_act` у значення логічної одиниці, та третій найменш значущий біт сигналу `dram_write` у те ж значення, залишаючи всі інші сигнали на нульовими. Ці сигнали вловлюються модулем адаптера DRAM, який, по суті, перетворює цей спрощений сигнал на більш складний, зрозумілий інтерфейсу PHY. Адаптер відокремлений від решти конструкції, тому його можна легко модифікувати для інтеграції нових інтерфейсів DRAM.

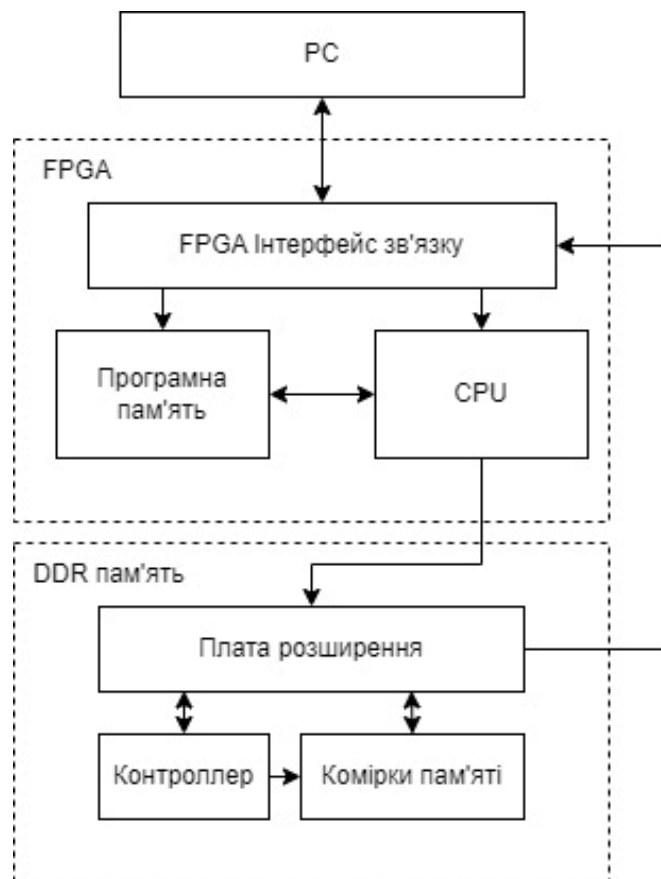


Рисунок 2.1 – Блок-схема платформи тестування чіпів DRAM

Функції програмного забезпечення FPGA в парі з платою розширення включають видачу послідовностей команд DRAM з потоком керування (наприклад, у циклі) та динамічну зміну широкого (наприклад, 512-бітного)

слова даних, яке записується в DRAM за допомогою виконання команди WRITE. Щоб дозволити програмованому ядру FPGA досягати високих частот, необхідних для взаємодії з пристроями DRAM (за нормальних робочих умов мінімальна робоча частота для пристроїв DDR4 становить 666 МГц), ми розробили його як порядкове ядро з п'ятьма потоками, що виконують функції вибірки, декодування та три етапи виконання. Етап вибірки вибирає одну інструкцію (72 біти) із пам'яті інструкцій. Етап декодування декодує інструкцію або в мікрооперацію виконання (μ -op), або в чотири μ -операції DRAM. Щоб виконати команди до контролера програма спочатку зчитує GPR, щоб отримати адресні операнди (наприклад, адресу банку команди PRE) відповідної команди DRAM, а потім видає команду пристрою DRAM через фізичний інтерфейс. Параметри синхронізації послідовності команд DRAM опосередковано визначаються звичайними інструкціями SLEEP або DRAM NOP. Основне API для ПК реалізоване на C++, а програмні засоби для FPGA на VHDL. Цей API є розширюваним завдяки модульній конструкції, що дозволяє розробникам легко інтегрувати нові інтерфейси DRAM і адаптуватись до нових плат FPGA. API складається з трьох компонентів: клас програми визначає, як створюються програми атак і їх сценарії, клас платформи надає користувачеві платформу на основі FPGA за допомогою ключових функцій, клас плати реалізує функції оболонки які спілкуються з FPGA через інтерфейс цієї плати (наприклад, драйвери PCIe).

Клас програми інкапсулює функції, які використовуються для створення програм та сценаріїв атак на пам'ять. Користувач створює цей сценарій, додаючи інструкції до списку одну за одною. Користувачі також можуть додавати мітки до програми, які можна використовувати як цілі розгалуження інструкцій потоку керування. Ця умовність дозволяє користувачеві легко створювати програми для тестування одного сценарію на багатьох пристроях без переписування логіки. Щоб продемонструвати простоту використання, на

рисунку 2.2 наведено приклад коду C++, який використовує API для створення сценарію, що читає рядок DRAM.

```

1  // The program below reads all cells in Bank 0, Row 0
2  // R5: bank address, R4: row address, R3: column address
3  // CASR: column stride register, R6: last column to read
4  Program p; // A sequentially-generated instruction list
5  p.appendLI(R5, 0); // Add a new instruction to the list
6  p.appendLI(R4, 0);
7  p.appendLI(R3, 0);
8  p.appendLI(CASR, 8); // Column address will increase by 8
9  p.appendLI(R6, 1024); // Read until 1024th column address
10 p.appendACT(R5, false, R4, false, 11);
11 p.appendLabel("read");
12 p.appendREAD(R5, false, R3, true, false, false, 0);
13 p.appendBL("read", R3, R6);
14 p.appendPRE(R5, false, false, 0);

```

Рисунок 2.2 – Блок програмованого сценарію доступу до пам'яті DRAM

Використовуючи додаткову затримку, користувачі можуть легко створювати послідовності команд DRAM із довільними параметрами синхронізації. Прапор приросту регістра адреси дозволяє шаблони доступу з довільними кроками. Як приклад, аргументи функції `appendACT` (рядок 10) на рисунку: R5, адресний регістр блоку пам'яті (BAR), `false` - не збільшувати BAR крок регістру блоку пам'яті, R4 регістр адреси рядка (RAR), `false`, не збільшувати RAR на регістр кроку адреси рядка та чекати 11 додаткових циклів командної шини DRAM після видачі команди ACT у порядку зліва направо. У функції `appendREAD` користувач зчитує всі блоки кешу DRAM у рядку DRAM, встановлюючи прапор приросту регістра адреси стовпця, без виконання інструкцій ADD між командами READ. Це дозволяє користувачеві планувати команди DRAM з максимально короткими часовими затримками.

Клас платформи визначає методи для виконання програм в інфраструктурі, маніпулювання станом платформи (наприклад, скидання,

увімкнення автоматичного оновлення рядків DRAM) та отримання даних із платформи через високошвидкісний інтерфейс.

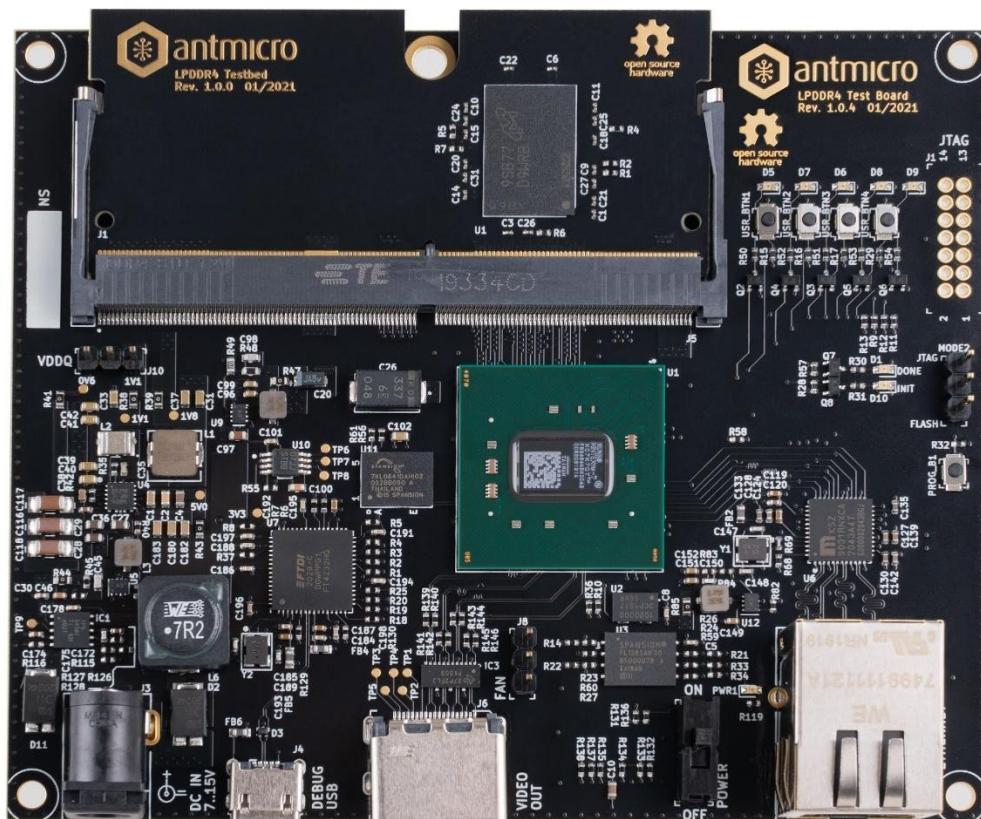


Рисунок 2.3 – Плата тестування чіпів DRAM

Клас плати визначає, як дані передаються між головною машиною та інфраструктурою FPGA через високошвидкісний інтерфейс. Він надає дві ключові функції для класу платформи, `sendData` і `receiveData`, які перевантажені кодом, який реалізує інтерфейс плати FPGA (наприклад, Xilinx XDMA). `sendData` використовується класом платформи для надсилання програми до FPGA, а `receiveData` використовується для зчитування даних із тестованого пристрою DRAM.

2.1.2 Вибір критеріїв для оцінювання ефективності захисту DRAM

Після створення тестової платформи важливо також визначити які саме параметри вона буде вимірювати для подальшого порівняння критеріїв захисту.

Ці критерії повинні враховувати різноманітність загроз, з якими стикається пам'ять, а також баланс між безпекою, продуктивністю та економічною доцільністю. Ефективний захист пам'яті неможливий без об'єктивних і вимірюваних показників, які дозволяють оцінювати, наскільки добре конкретний метод виконує свої функції в різних умовах. Таким чином, формування цих критеріїв є основою для побудови надійних систем безпеки.

Рівень стійкості до атак є фундаментальним критерієм оцінювання ефективності захисту пам'яті, оскільки саме ця властивість визначає здатність механізму протидіяти реальним загрозам. Захисні системи повинні ефективно виявляти та блокувати відомі вразливості, зокрема атаки, що активно використовуються, як-от RowHammer. Ця атака експлуатує фізичні обмеження DRAM, що може призвести до серйозних наслідків, включаючи зміну прав доступу або витік конфіденційних даних. Для оцінювання стійкості до таких загроз необхідно вимірювати, з якою ймовірністю захисний механізм може запобігти успішному виконанню атаки. Це включає аналіз ефективності різних методів, таких як пом'якшення (mitigation), виявлення (detection) або повне усунення уразливості. В нашому випадку успішність атаки визначається кількістю спотворених бітів в контролері пам'яті, не залежно від того впливають вони на подальшу роботу чи ні. В ідеальному варіанті такі зміни мають бути відсутні повністю, але в реальному світі цього досягти майже неможливо, тому основною метрикою тут буде відношення кількості зміни бітів від часу атаки чи частоти оновлення комірок.

Крім оцінювання стійкості до окремих атак, важливо враховувати здатність механізмів протистояти комбінованим або мультивекторним атакам. Зловмисники часто використовують кілька типів атак одночасно, комбінуючи їх для обходу систем безпеки. Наприклад, RowHammer може бути поєднаний з атаками на кеш або слабкі місця в операційній системі, що підвищує його ефективність. Захисні системи повинні враховувати ці сценарії та бути

здатними до адаптації у реальному часі. Для оцінювання стійкості до таких атак можна використовувати моделювання складних сценаріїв, де атакуючі вектори взаємодіють між собою, перевіряючи межі роботи захисних механізмів. Стійкість до атак має враховувати довговічність механізмів захисту в умовах змінних або ескалаційних загроз. У сучасному світі атакуючі технології постійно розвиваються, і захист, який є ефективним сьогодні, може стати малоефективним у майбутньому. Наприклад, розвиток більш потужних інструментів для генерації високої частоти активацій рядків у DRAM може значно ускладнити протидію RowHammer. Тому критерії оцінювання стійкості мають включати аспекти довговічності, визначаючи, як добре механізм справляється із загрозами за умов їх модифікації або еволюції. Також необхідно оцінювати можливість обходу захисту через побічні канали або недоліки реалізації. У деяких випадках, навіть якщо механізм ефективно блокує прямі атаки, зломисники можуть знайти способи обійти його, використовуючи слабкі місця в суміжних компонентах системи. Наприклад, атакуючий може використовувати специфічні часові затримки для вимірювання роботи захисту і таким чином адаптувати свої дії. Відповідно оцінити кількість помилок пам'яті в стерильному тесті буде недостатньо, так як в реальному світі є вплив часу, атака в кінці іншої чи мультивекторні атаки одночасно. Тому стійкість до атаки є першим критерієм захисного механізму, що буде виражатись N змін бітів за T атаки.

$$\text{Ефект атаки} = \frac{N_{\text{зміни_бітів}}}{T_{\text{атаки}}} \quad (2.1)$$

Також не слід забувати, що захист пам'яті не повинен ставати причиною значного зниження пропускну здатності чи збільшення затримок при доступі до пам'яті, особливо у високонавантажених системах, таких як дата-центри або обчислювальні платформи в реальному часі. Тому важливо оцінювати, як різні захисні механізми впливають на продуктивність і наскільки це зниження є прийнятним для конкретного застосування. Одним із важливих аспектів

продуктивності є швидкість доступу до пам'яті, яка може бути порушена через використання таких механізмів, як періодичне оновлення рядків (refresh) або активація захисних алгоритмів. Наприклад, у методах захисту типу PARA (Probabilistic Adjacent Row Activation) кожна активація пам'яті супроводжується додатковими операціями, які забезпечують зниження ризику атак RowHammer. Однак такі додаткові операції можуть викликати затримки, особливо якщо частота доступу до пам'яті є високою. Для оцінювання продуктивності необхідно проводити тести використання енергії як на Рисунку 2.4 для створеної системи в порівнянні з існуючими.

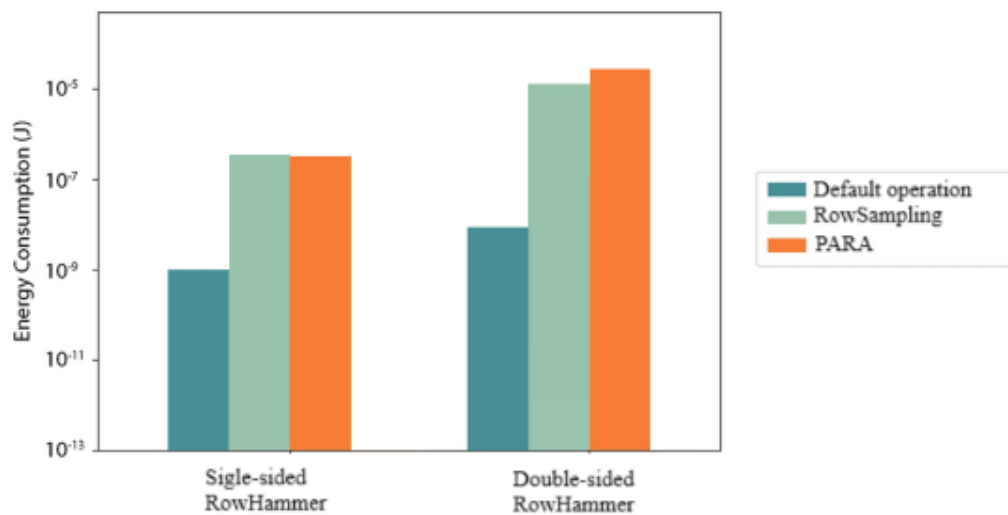


Рисунок 2.4 – Використання енергії в існуючих схемах захисту

Також важливо враховувати вплив захисних механізмів на багатозадачність системи. У сучасних обчислювальних платформах пам'ять одночасно використовується кількома процесами чи потоками. Захисні алгоритми можуть створювати конфлікти під час доступу до спільних ресурсів, особливо якщо вони передбачають блокування певних операцій для перевірки чи оновлення даних. Наприклад, у випадках інтенсивного використання пам'яті, таких як великі бази даних або моделі машинного навчання, захист може викликати затримки, які негативно впливають на загальну

продуктивність. Тому важливо моделювати багатопроцесорні середовища і вимірювати, як захист впливає на їхню ефективність.

Окрім цього, продуктивність також залежить від ефективності реалізації самого захисного механізму. Наприклад, апаратні рішення зазвичай забезпечують швидшу обробку, оскільки інтегровані безпосередньо в контролери пам'яті, однак вони можуть бути менш гнучкими. З іншого боку, програмні методи можуть мати більшу варіативність і можливість налаштування, але їхня робота може створювати навантаження на центральний процесор, що впливає на швидкість виконання інших задач. Тому необхідно проводити порівняння між різними реалізаціями захисних механізмів, враховуючи як їхній вплив на продуктивність, так і контекст, у якому вони застосовуються. Тому через це є доцільним створення кількох підходів до захисту як програмних так і апаратних, щоб оцінити їх сильні та слабкі сторони.

Ще одним важливим фактором є економічна вигідність і засобів захисту пам'яті, оскільки впровадження та підтримка таких механізмів завжди пов'язані з певними витратами. Розробка, інтеграція й експлуатація засобів захисту не повинні перевищувати допустимих бюджетних обмежень, особливо в умовах масового виробництва апаратного забезпечення чи застосування в масштабних інфраструктурах, таких як дата-центри. У цьому контексті важливо оцінювати не лише початкові витрати на розробку та впровадження, але й довгострокові витрати, пов'язані з енергоспоживанням, обслуговуванням і оновленням.

Одним із основних аспектів економічної вигідності є початкова вартість інтеграції захисного механізму. Апаратні методи, наприклад, можуть вимагати модифікацій у конструкції контролерів пам'яті чи чипів DRAM, що призводить до збільшення витрат на виробництво. Водночас програмні рішення зазвичай є дешевшими для розробки, оскільки не потребують змін у апаратній частині,

але можуть збільшувати навантаження на центральний процесор чи інші компоненти системи, що вимагає додаткових ресурсів. Для оцінювання цього аспекту необхідно враховувати, наскільки розробка та впровадження конкретного механізму відповідають економічним можливостям організації або виробника. Довгострокові витрати на підтримку та оновлення також є невід’ємною частиною економічної оцінки. У міру розвитку технологій і появи нових видів атак захисні механізми потребують адаптації та вдосконалення. Це може включати оновлення прошивок, вдосконалення алгоритмів або навіть повну заміну певних компонентів. Наприклад, захисні механізми, які є ефективними сьогодні, можуть стати застарілими через кілька років через появу нових методів атак, таких як удосконалені варіанти RowHammer. Тому важливо оцінювати, чи передбачає захист гнучкість і можливість оновлення з мінімальними витратами.

Зрештою, економічна вигідність також залежить від масштабів застосування захисного механізму. Для великих організацій або дата-центрів навіть невелике підвищення вартості обладнання чи енергоспоживання може масштабуватися до значних сум. З іншого боку, для спеціалізованих систем із високими вимогами до безпеки (наприклад, у військових або медичних застосуваннях) економічні аспекти можуть мати нижчий пріоритет порівняно з рівнем безпеки. Саме тому всі ці фактори потрібно враховувати при комплексній оцінці методу захисту.

2.2 Збір та структуризація результатів тестування

Після створення платформи для уніфікованого тестування засобів захисту пам’яті можна переходити до збору даних чіпів DRAM від різних сучасних виробників. Цей етап є важливим не лише для перевірки роботи самої платформи, але й для формування бази даних, яка стане основою для порівняльного аналізу характеристик пам’яті та їхньої вразливості до атак.

Сучасні DRAM-чіпи значно варіюються за архітектурою, технологічними процесами виробництва, обсягом пам'яті та оптимізаціями для конкретних застосувань, тому залучення даних від різних виробників дозволить отримати повну та об'єктивну картину їхніх сильних і слабких сторін.

Збір даних необхідно організувати з урахуванням кількох важливих аспектів. По-перше, тестування має охоплювати широкий спектр моделей DRAM, зокрема як масово доступні чіпи для споживчого ринку, так і високопродуктивні варіанти для серверів або спеціалізованих пристроїв. По-друге, важливо враховувати особливості технологічних поколінь, адже кожен новий виток технології, наприклад, перехід від DDR4 до DDR5, супроводжується змінами в архітектурі, які можуть впливати на стійкість до атак RowHammer та інших загроз. Цей багатогранний підхід дозволить виявити загальні тренди в розвитку технологій пам'яті та слабкі місця, які потребують підвищеної уваги.

Додатково, тестування слід проводити в умовах, що максимально наближені до реальних сценаріїв використання пам'яті. Це означає, що потрібно враховувати різні режими роботи, температурні умови, напругу живлення та частотні параметри. Наприклад, деякі чіпи можуть показувати високу стійкість до атак при номінальних умовах, але втрачати її при збільшенні температури чи підвищенні напруги. Тому результати тестів повинні включати в себе інформацію про всі фактори, які впливають на вразливість пам'яті, щоб забезпечити об'єктивну та комплексну оцінку.

Отримані дані також можуть бути використані для класифікації чіпів за рівнем їхньої безпеки. На основі тестування можна створити рейтинг виробників і конкретних моделей DRAM, який стане орієнтиром для розробників апаратного забезпечення та системних інтеграторів. Це дозволить їм вибирати найбільш захищені компоненти для своїх продуктів і підвищувати загальний рівень безпеки пристроїв. Крім того, така база даних стане важливим

ресурсом для виробників пам'яті, які зможуть використовувати її для вдосконалення своїх продуктів і розробки більш стійких архітектур.

2.2.1 Стан захисту стокових чіпів пам'яті

Багато статей [41] містять детальний експериментальний аналіз різних характеристик RowHammer для чіпів своїх років. Ці роботи концентрували свою увагу на наступних параметрах:

- поширеність у чіпах одного виробника;
- залежність успішності атаки від шаблону доступу;
- залежність ймовірності удачі від шаблону даних;
- залежність від температури;
- кореляція адрес між рядками пам'яті жертви та агресора;
- сигнатура бітів у захищеному рядку, які змінюються через RowHammer;
- кількість рядків, які постраждали через RowHammer у сусідньому рядку;
- зв'язок уразливих RowHammer комірок із компрометованими комірками;

Завдяки цим вже існуючим тестам ми можемо виділити багато інваріант що будуть незмінними в одній партії пристроїв, при одній температурі чи при однакових сценаріях. Також в попередніх дослідженнях показано, що той факт, що рядок пам'яті є вразливим до RowHammer можна прослідкувати в багатьох схожих за структурою та часом виробництва системах, тому цю інформацію можна систематизувати та аналізувати з кожним новим поколінням чіпів. Одним із ключових висновків зібраної інформації із зазначених вище статей є те, що помилки, викликані RowHammer, передбачувано повторюються. Іншими словами, якщо значення комірки пам'яті буде змінено за допомогою RowHammer, значення тієї самої комірки, швидше за все, знову буде змінено якщо застосувати RowHammer повторно. Ця характеристика дозволяє

створювати повторювані атаки на безпеку контрольованим способом, якщо зловмисник знає основні зовнішні характеристики встановлених на машині фізичних пристроїв.

Використовуючи експериментальну інфраструктуру тестування DRAM на основі FPGA було протестовано 239 модулів DRAM, виготовлених трьома основними виробниками (А, В, С) за останні одинадцять років (2013–2023). Дані також було оновлено для найновіших чіпів. Дослідження виявили, що 220 із протестованих схем вразливі до RowHammer. Найперша помилка була в чіпах виробництва 2013 року. Це показано на рисунку 2.5, де показано частоту помилок, яку ми виявили в усіх 239 перевірених модулях, де модулі класифіковані на основі дати виробництва.

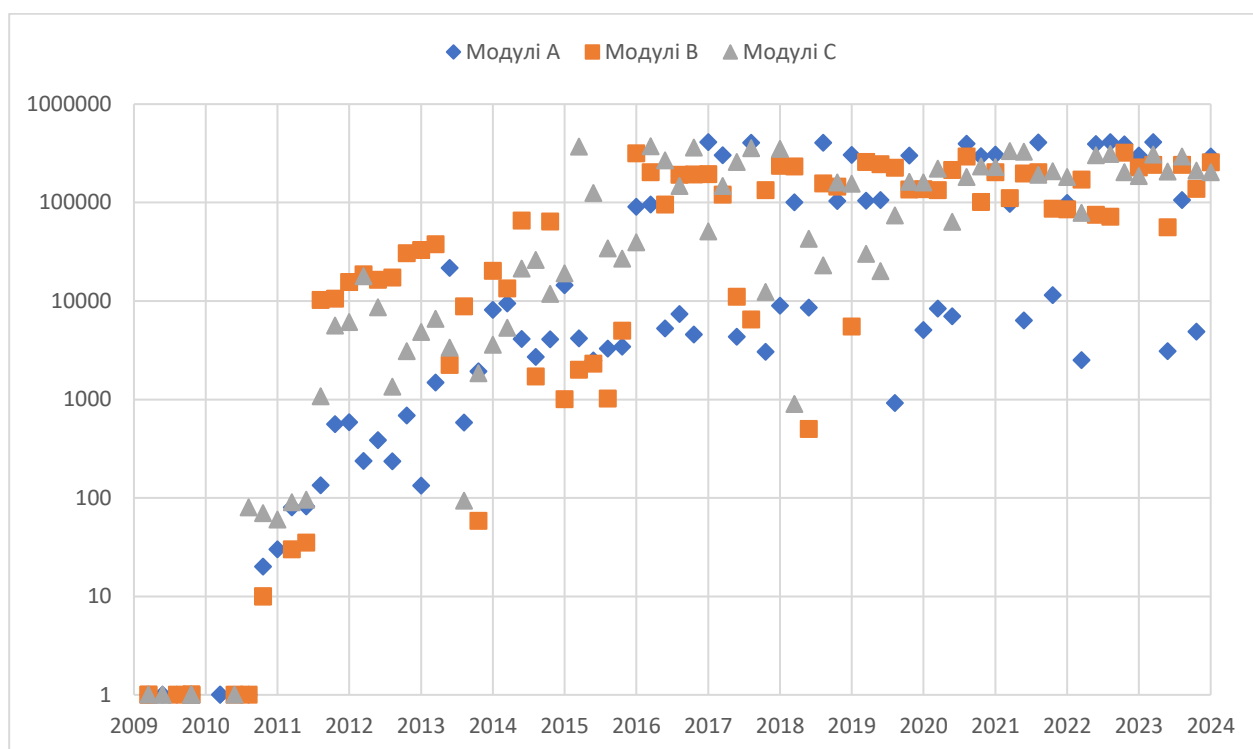


Рисунок 2.5 – залежність кількості RowHammer помилок від року виробництва DRAM чіпів.

Як видно з зібраних даних, усі модулі DRAM, виготовлені в період 2013–2014 років, демонструють вразливість до атак RowHammer. Це дозволяє зробити важливий висновок: поява таких вразливостей є явищем, тісно

пов'язаним із еволюцією більш прогресивних технологічних процесів виробництва пам'яті. Зменшення фізичних розмірів комірок пам'яті та відстаней між ними, що було продиктоване прагненням підвищити щільність зберігання даних і зменшити енергоспоживання, стало водночас джерелом нових ризиків. Цей факт демонструє, що розвиток технологій, спрямований на вдосконалення одних характеристик, може створювати потенційні загрози для інших аспектів, таких як безпека. Зменшення розмірів транзисторів, збільшення щільності розташування елементів і загальна оптимізація для підвищення продуктивності сприяли появі цієї вразливості, яка проявляється саме в більш просунутих поколіннях технологічних процесів. Такий зв'язок між рівнем технологічного прогресу та схильністю до нових типів атак підкреслює важливість вивчення і розуміння подібних тенденцій.

Інтерполяція цих даних у відповідні графіки може надати ширший контекст і дозволити виявити тренди в розвитку технологій пам'яті. Наприклад, аналіз частоти вразливостей у різні роки виробництва може показати, як зміни технологічних процесів (перехід на менші нанометри) вплинули на стійкість модулів до таких атак. Крім того, графіки можуть бути корисними для аналізу, чи є прогрес у зменшенні вразливостей у DRAM, створених після 2014 року, і чи впроваджені виробниками зміни ефективно справляються з цими загрозами. Таким чином, можна буде виявити певні закономірності, наприклад, залежність між поколінням DRAM та їхньою стійкістю до RowHammer. Додатково, графічне представлення таких даних дозволяє візуально порівняти вразливість модулів різних виробників. Це може бути корисним для вивчення того, які стратегії боротьби з RowHammer були впроваджені різними компаніями та наскільки вони були успішними. Зокрема, аналіз дозволяє зрозуміти, чи впливають на вразливість специфічні особливості архітектури чіпів, використовувані матеріали або алгоритми захисту, такі як Refresh Management або Error-Correcting Code (ECC).

Крім того, дані можуть допомогти оцінити, наскільки виробники приділяють увагу вирішенню цієї проблеми. Наприклад, якщо модулі певних років чи поколінь демонструють поступове зменшення вразливостей, це свідчить про те, що виробники активно вдосконалюють свої технології. У разі ж відсутності покращень можна зробити висновок, що проблема залишається актуальною і потребує більшого втручання з боку галузі.

Через сильне розкидування точок графіком лінійна інтерполяція не буде підходити, тому використаємо методи інтерполяції високого порядку. Метод інтерполяції високого порядку дозволяє наблизити функцію, значення якої відомі лише в дискретних точках, поліномом високого степеня. Метод Лагранжа є представником таких функцій. Нехай задано набір точок $(x_0, y_0), (x_1, y_1) \dots (x_n, y_n)$, де $y_i = f(x_i)$. Інтерполяційний поліном $P(x)$ визначається як:

$$P(x) = \sum_{i=0}^n y_i L_i(x) \quad (2.2)$$

де $L_i(x)$ — базові поліноми Лагранжа, які визначаються формулою:

$$L_i(x) = \prod_{\substack{0 \leq j \leq n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \quad (2.3)$$

Тут кожен базисний поліном $L_i(x)$ враховує всі точки, окрім x_i , і має значення 1 в точці x_i і значення 0 в усіх інших точках. Інтерполяція Лагранжа зазвичай дає хороші результати, якщо потрібно побудувати поліном для невеликої кількості точок, однак вона має високий ступінь полінома, який може бути обчислювально дорогим для великих наборів даних. На щастя нам потрібно вирахувати інтерполяцію лише один раз, тому цей недолік нам не перешкоджає.

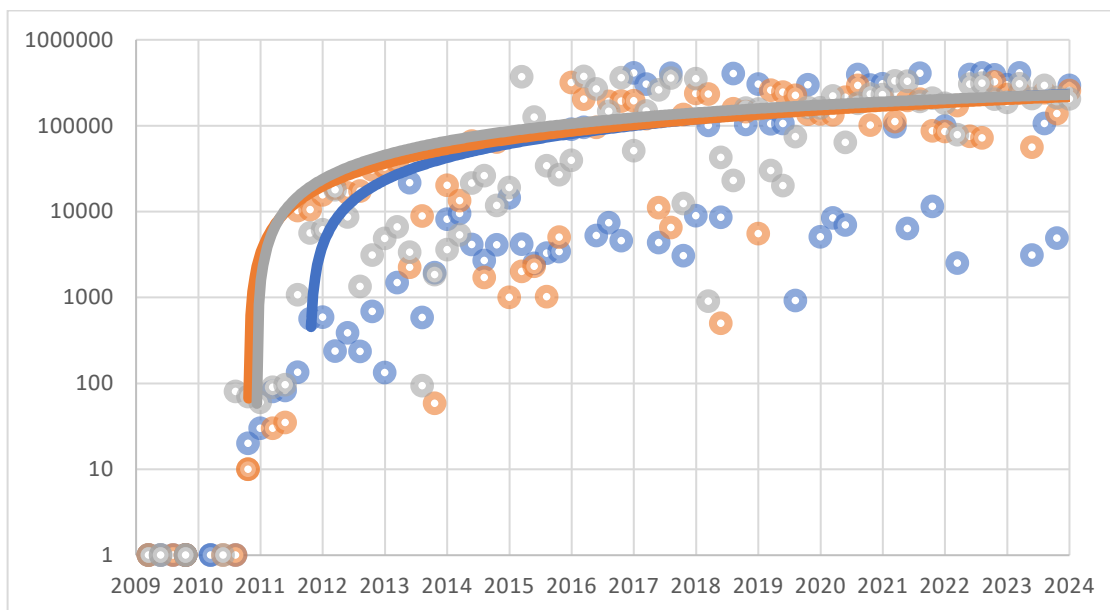


Рисунок 2.6 – Графіки тренду даних протестованих чіпів DRAM

Аналізуючи дані, можна помітити значний тренд на зростання кількості помилок у модулів пам'яті, виготовлених виробниками А та В. Цей феномен може бути пояснений кількома технічними й економічними чинниками. Одним із ключових є зменшення технологічного процесу, яке дозволяє збільшити густину пам'яті, тобто розмістити більше комірок у тому самому об'ємі. Проте зменшення розміру осередків пам'яті та зменшення відстані між ними сприяє зростанню взаємного впливу комірок, зокрема витіканню заряду. Це значно підвищує ризик виникнення помилок, зокрема через атаки типу RowHammer, які експлуатують саме ці фізичні обмеження.

Ще одним важливим чинником є прагнення виробників до здешевлення виробництва. У контексті зростання конкуренції на ринку DRAM це може призводити до використання менш якісних матеріалів, спрощення технологічних процесів або скорочення заходів безпеки під час розробки та виготовлення. Такий підхід, хоч і знижує витрати на виробництво, може сприяти появі більших вразливостей до специфічних атак, що й проявляється у підвищенні частоти помилок у тестованих модулях. У той же час модулі пам'яті виробника С демонструють тенденцію до покращення захисту від атак

типу RowHammer. Це можна частково пояснити більшим технологічним процесом, що використовується в їхньому виробництві. Збереження більшої відстані між комірками та менша щільність осередків дозволяють зменшити ризик неконтрольованого впливу між ними, що позитивно позначається на загальній стійкості модулів. У контексті боротьби з RowHammer це може свідчити про свідомий вибір на користь більш надійного, хоча й менш компактного дизайну, спрямованого на мінімізацію вразливостей.

При аналізі цих даних важливо враховувати різноманітність лінійок пам'яті, що випускаються різними виробниками. Різні серії модулів, навіть у межах одного виробника, можуть демонструвати абсолютно різні результати в тестах через різницю в конструкції, цільовій аудиторії та пріоритетах при розробці. Наприклад, серії для серверів або критично важливих застосунків можуть мати значно кращий захист, ніж масові споживчі модулі. Це потребує додаткової сегментації даних, яка дозволить виділити специфічні зони скупчення результатів і визначити характеристики цих зон для глибшого розуміння проблеми. Особливо важливим аспектом аналізу є врахування хронологічного чинника. Зусилля виробників із боротьби з атакою RowHammer досягли стадії масового впровадження в середині 2017 року, коли почали з'являтися помітні зміни в архітектурі модулів пам'яті та захисних механізмах. Це робить доцільним розділення даних на два часові відрізки: до 2017 року, коли захист лише почав формуватися, і після, коли виробники почали впроваджувати більш досконалі технології на рівні масового виробництва. Такий підхід дозволяє чіткіше ідентифікувати зміни в ефективності захисту та їхній зв'язок із еволюцією технологій пам'яті.

Тому перейдемо до кластеризації способом кластеризації нечітких К-середніх. У нечіткій кластеризації кожна точка може належати кільком кластерам одночасно, з певною ймовірністю, а не бути цілком прив'язаною до одного кластера. Метод нечітких k-середніх намагається вирішити ситуацію,

коли точки знаходяться близько до кількох центрів або є неоднозначними, замінюючи традиційну відстань на ймовірність, що може бути функцією відстані, наприклад, інвертованою відстанню. В цьому методі для кожної точки обчислюється ймовірність її належності до кожного з кластерів, а потім використовуються ці ймовірності для обчислення зваженого центроїду. Процеси ініціалізації, ітерацій та припинення є подібними до тих, що використовуються в стандартному алгоритмі k -середніх. Отже, отримані кластери мають розглядатись як ймовірнісні розподіли, а не як чітко визначені мітки. Важливо зазначити, що стандартний алгоритм k -середніх є окремим випадком нечітких k -середніх, коли ймовірність належності точки до конкретного кластера дорівнює 1, якщо точка найближча до центроїда, і 0 в усіх інших випадках.

Алгоритм нечітких k -середніх можна описати такими кроками:

1. Задаємо фіксовану кількість k -кластерів
2. Ініціалізуємо випадковим чином k до μ_k , пов'язане з кластерами, і обчислюємо ймовірність того, що кожна точка даних x_i є членом заданого кластера k
3. Перераховуємо центроїд кластера як зважений центроїд, враховуючи ймовірності членства всіх точок даних x_i :

$$\mu_k(n+1) = \frac{\sum_{x_i \in k} x_i \times P(\mu_k | x_i)^b}{\sum_{x_i \in k} P(\mu_k | x_i)^b} \quad (2.4)$$

$$P(\mu_k | x_i) = \left(\sum_{j=1}^k \left(\frac{d_{ik}}{d_{jk}} \right)^{\frac{2}{b-1}} \right)^{-1} \quad (2.5)$$

4. Продовжуємо таку ітерацію, доки матриця членства не збіжиться або поки не буде досягнуто вказаної користувачем кількості ітерацій.

При цьому змінна b є показником зважування, який визначає відносні значення вагів для розподілу і рівень нечіткості. Коли $b \rightarrow 1$, значення точок, що мінімізують функцію помилки у квадраті, стають краще розподіленими, в той час як $b \rightarrow \infty$ членство до кластера наближається до 1, що вважається найбільш нечітким станом. На сьогодні не існує теоретичних доказів, які б вказували на оптимальне значення b , проте емпірично корисні значення знаходяться в межах від 1 до 30, а в більшості досліджень значення $1.5 \leq b \leq 3.5$ показали хороші результати. Тож використаємо значення 2 для нашого випадку і проводимо розрахунки для кластеру точок до та після позначки 2017 року. Після цього візуалізуємо на одному графіку і отримаємо графік представлений на рисунку 2.7.

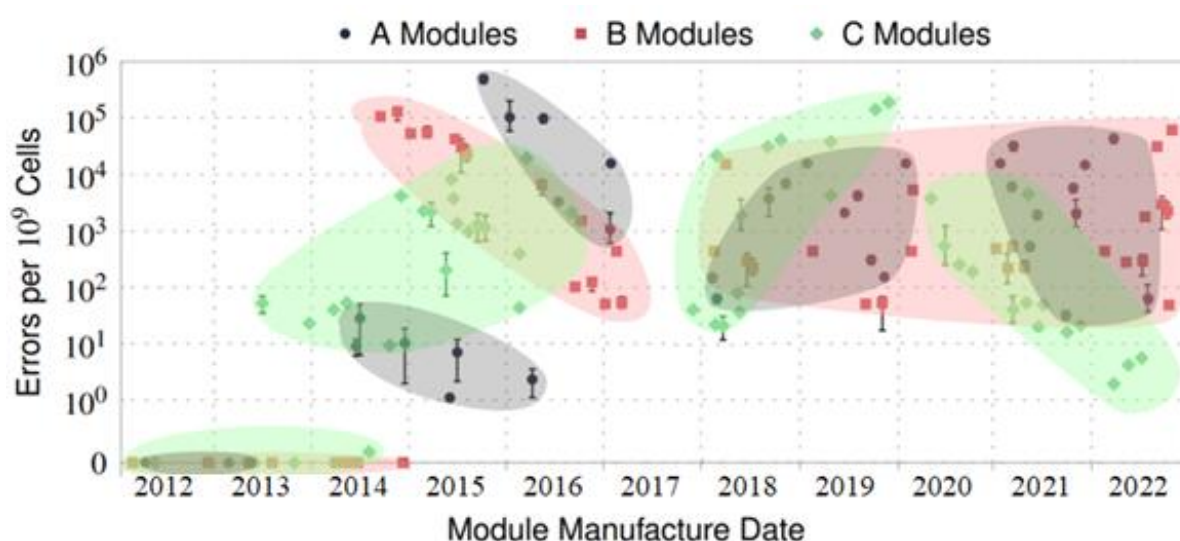


Рисунок 2.7 – Графіки кластерів вразливостей протестованих чіпів DRAM

У першому скупченні спостерігаємо, що всі модулі пам'яті, незалежно від виробника, не мають значної вразливості до атаки RowHammer, що пояснюється їхньою низькою густиною пам'яті. Цей фактор є критичним, оскільки низька густина означає, що між комірками пам'яті знаходиться

більший фізичний простір, що знижує ймовірність витікання заряду між сусідніми комірками та, відповідно, мінімізує шанси для атаки RowHammer бути ефективною. Модулі пам'яті з такою архітектурою, як правило, більш стійкі до електричних впливів, оскільки зменшене щільне розміщення елементів знижує ймовірність того, що одна комірка зможе мимовільно змінити значення сусідньої. Це означає, що ранні модулі пам'яті, виготовлені з менш щільними чіпами, природно мали більший захист від цього типу атак.

Що стосується виробника А, то його два основні сегменти — преміум лінійка і бюджетні модулі — демонструють різний рівень вразливості. Преміум модулі, ймовірно, мають більш інноваційні технології, які знижують ймовірність успішної атаки RowHammer. Можливо, вони використовують більш передові технології виробництва, що включають додаткові елементи захисту, такі як більш ефективне управління електричними полями або спеціалізовані алгоритми для зміни рівня напруги в комірках пам'яті. Це дозволяє преміум моделям значно знижувати вразливість до атак, що є важливим аспектом для виробників, орієнтованих на висококласних споживачів, де питання безпеки є надзвичайно важливим.

У той час як бюджетні модулі цього ж виробника не показують таких високих результатів, це може свідчити про те, що для здешевлення виробництва ці модулі використовують менш складні технології або знижені рівні захисту, що робить їх більш вразливими до RowHammer. Проте, незважаючи на це, обидва сегменти виробника А зараз перебувають на стабільному рівні, що дозволяє зробити висновок, що на даному етапі їх технології захисту від RowHammer досягли певного плато. Тренд вказує на незначне зростання вразливості, однак це може бути частиною природного розвитку технологій, оскільки з переходом до нових технологічних процесів можуть виникати нові вразливості, пов'язані з меншими розмірами чіпів та більшою щільністю пам'яті.

Виробник В демонструє схожі результати, маючи стабільні характеристики на рівні плато, що може свідчити про те, що обидва ці виробники досягли певного оптимуму в захисті від атак RowHammer, і тепер зміни відбуваються лише через поступову еволюцію технологій пам'яті, але без великих коливань. Як і у випадку з виробником А, цей тренд може змінитись при переході до нових технологій, які можуть принести з собою нові можливості або уразливості, пов'язані з підвищенням щільності пам'яті.

Модулі виробника С, навпаки, демонструють цікаву динаміку. Хоча в минулі роки вони мали тенденцію до зростання вразливості, на теперішній момент вони значно покращили свій захист від RowHammer. Цей тренд свідчить про те, що виробник С активно працює над покращенням технологій захисту і намагається скористатися новими методами або матеріалами для зменшення впливу атак. Враховуючи те, що їх модулі ближчі до сучасних розробок, можна припустити, що вони впроваджують більш ефективні рішення для забезпечення безпеки, що робить їх конкурентоспроможними на ринку.

2.2.2 Тестування додаткових методів захисту

Базовим методом програмного захисту є пришвидшення оновлення комірок в банках пам'яті. Це звісно несе додаткові витрати енергії, але це може бути дуже хорошою відправною точкою порівняння рівня захисту і відношення його до енерговитрат. Оновлення 2х – це спосіб, яке зазвичай реалізується на серверних чіпсетах від Intel після появи Sandy Bridge і є запропонованим за замовчуванням. Це зменшує час оновлення рядка контролером пам'яті з 64 мс до 32 мс і звужує потенційне вікно для проведення RowHammer або інших помилок спричинених невірним доступом до пам'яті.

Після цього можна зазначити те що представили виробники контролерів пам'яті, як-от Intel - pTRR (pseudo Target Row Refresh) [42], для зменшення впливу RowHammer. у майбутніх системах. На жаль, такі механізми оновлення

рядка жертви на основі контролера пам'яті не знають про фізичну суміжність рядків агресора та жертви в чіпі DRAM і, як такі, вони можуть не забезпечити повний захист від RowHammer. Що стосується DRAM, виробники DRAM запровадили механізми TRR (target row refresh) [10] і стверджували, що їхні нові мікросхеми DDR4 не вразливі до RowHammer [52] із захистом, який забезпечують ці механізми. TRR — це загальний термін, який використовується для механізмів, які оновлюють цільові рядки, до яких певним чином визначено частий доступ. На жаль, виробники DRAM не описували (і досі не описують), як їхні реалізації надійно запобігають RowHammer, і не розкривають, як їх реалізації працюють. Таким чином, приблизно в 2019-2020 роках було незрозуміло, чи можливі зміни бітів RowHammer у реальних мікросхемах DDR4 DRAM, але після представлення TRResspass стало зрозуміло що це можливо. TRR можна віднести до стандартних налаштувань контролерів, так як він є вбудованим в програмне забезпечення, але так як він є представником вибору рядка атакуючих, то цей спосіб представлятиме RowSampling. Метод RowSampling є одним з підходів до захисту від атак типу RowHammer, який використовує техніку вибірки рядків пам'яті для запобігання несанкціонованим змінам вмісту комірок пам'яті через агресивне активування сусідніх рядків. RowSampling метод полягає в регулярній перевірці або оновленні певних рядків пам'яті під час операцій зчитування чи запису на основі випадково згенерованих значень. Ідея методу в тому, щоб створити систему вибірки таких рядків, які будуть активно оновлюватися або вибиратися контролером пам'яті в процесі роботи системи. Це знижує ймовірність того, що рядки, які можуть стати жертвами атаки, залишатимуться без уваги і уразливими до RowHammer.

Контролер пам'яті вибирає рядки з певною ймовірністю (яка може бути змінною в залежності від стратегії або конкретних параметрів) і активно їх перезаписує або оновлює, таким чином запобігаючи накопиченню помилок через несанкціоновану активацію. Відповідно, рядки, які не вибираються,

отримують додатковий захист через процеси перезапису або регулярного вибору для оновлення, що сприяє зміцненню надійності системи пам'яті.

Один з головних аспектів цього методу — це його порівняна простота впровадження. У більшості випадків не потрібно перепроєктувати архітектуру пам'яті або використовувати складні додаткові механізми для забезпечення захисту. RowSampling добре працює в середовищах, де використовуються ресурси з низьким навантаженням, де частота оновлення рядків не викликає значних затримок. У порівнянні з іншими методами, такими як використання додаткових паритетних бітів або спеціалізованих контролерів пам'яті, RowSampling може вимагати мінімальних ресурсів для реалізації.

В залежності від частоти вибірки, цей метод може призвести до деяких накладних витрат, оскільки контролер пам'яті повинен активно оновлювати вибрані рядки. Це може уповільнити роботу системи, якщо вибірка відбувається надто часто.

2.3 Виявлення обмежень та вразливостей в наявних апаратних і програмних рішеннях

Незважаючи на впровадження численних методів захисту, як апаратних, так і програмних, більшість із них мають певні обмеження та не гарантують абсолютного захисту. Використання тестової платформи на основі FPGA (рисунок 2.8) дозволяє отримати глибоке розуміння ефективності існуючих механізмів захисту та визначити їхні потенційні слабкі місця.

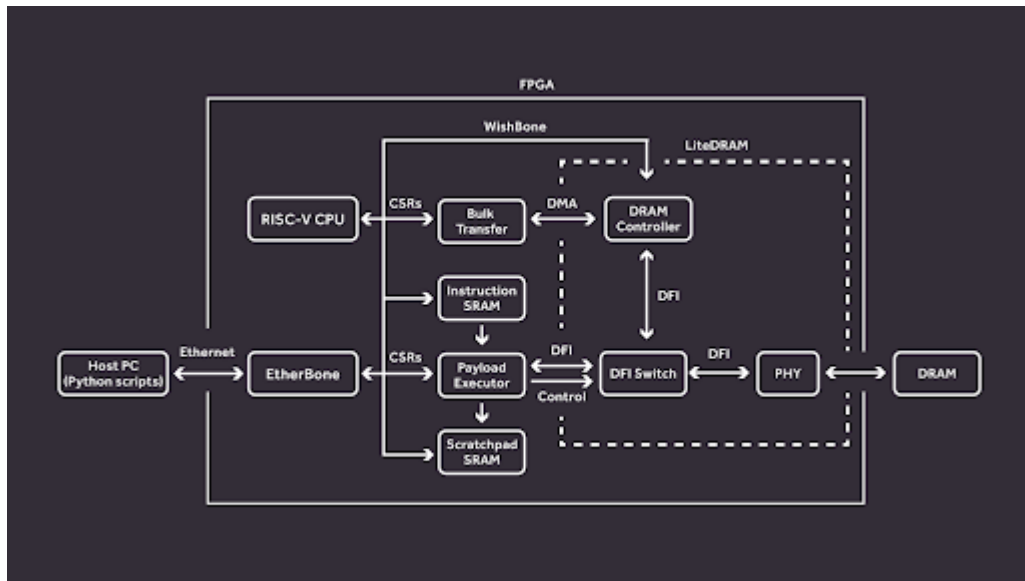


Рисунок 2.8 – Схема тестувальної системи чіпів DRAM

Для аналізу вразливостей ми використовували спеціально розроблену тестову платформу на базі FPGA, яка дозволяє проводити низькорівневі експерименти з DRAM-чіпами. Платформа підтримує можливість маніпулювання таймінгами пам'яті, реалізації алгоритмів агресивного доступу до рядків та збору статистики щодо бітових помилок. Дослідження проводилося на зразках DRAM-пам'яті DDR3 та DDR4 від різних виробників, що дозволило отримати узагальнені результати.

Виявлені обмеження апаратних методів захисту було сформовано за критеріями кількості знайдених помилок і кількості використаної енергії. В ідеальних умовах помилки, які можна виправити, мають бути виявлені та виправлені, тоді як виявлені помилки, які не можна виправити, мають призвести до збою процесу або системи. У цій конфігурації єдиний спосіб, яким зломисник може скомпрометувати систему, це запустити достатню кількість бітів у правильних позиціях, щоб функція ECC не виявила пошкодження. Таблиця 2.1 показує мінімальну кількість невідворотної зміни бітів, необхідних для бітів даних або бітів даних і контрольних бітів.

Таблиця 2.1 – Критичні умови втрати пам'яті для процесорів

Тип процесора	Патерн Роботи	Зміни бітів	Кількість бітів
AMD-1	[P1]	3-BF-16	3 Символи, 1 Контрольний
AMD-1	[P2]	4-BF-16	Мін. 2 символи
Intel-1	[P3]	4-BF-8	Мін. 2 символи
Intel-1	[P4]	2-BF-8	Мін. 2 символи

У більшості модулів DIMM, які вразливі до RowHammer, важко запустити такі численні зміни бітів поблизу один від одного. Однак набагато простіше викликати пошкодження на Intel-1, як обговорюватиметься далі. Як показано в таблиці 2.1, ми виявили, що в Intel-1 виявлені невіправні помилки не призводять до збою системи і навіть не повідомляються ОС. Схоже, що основною причиною є неналежна підтримка програмного забезпечення для контролера пам'яті в ОС, тобто драйвер звіту про помилки не розпізнає та не ініціалізує ресурси механізму звіту про помилки. Як наслідок, зловмисник може використовувати систему в її конфігурації за замовчуванням з меншою кількістю перевертань бітів, ніж необхідно, з ідеальною гарантією, наданою функцією ECC.

Якщо говорити про шаблони, які можна використовувати, то ми використовуємо Z3, розв'язувач обмежень, для розробки придатних для експлуатації шаблонів функцій ECC для AMD-1 і Intel-1. Для AMD-1 зловмиснику потрібні принаймні три зміни бітів у 16 байтах (тобто слово ECC), коли одне з збурень бітів знаходиться в керуючих бітах ([P1]). Інші два бітові спотворення мають бути націлені на два різні символи (тобто бути принаймні на 8 біт один від одного). При націлюванні лише на біти даних чотири змінених біта мають припадати на принаймні два різні символи в слові ECC ([P2]). Експерименти показали, що стандартні механізми ECC можуть ефективно

виправляти лише одиничні бітові помилки, тоді як атака RowHammer може призводити до множинних бітових змін у сусідніх комірках. Випробування також продемонстрували, що ECC, реалізований на рівні контролера пам'яті, не завжди здатний адекватно реагувати на аномальні патерни доступу, що дозволяє зломисникам обходити цей механізм захисту.

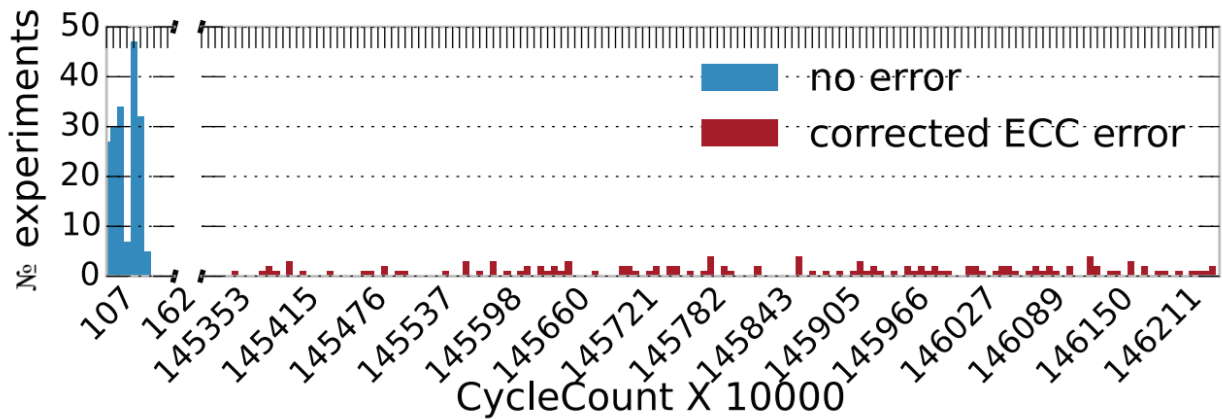


Рисунок 2.9 – Кількість ECC корекції відповідно до кількості атак

Ще одним апаратним підходом є збільшення частоти оновлення пам'яті, що зменшує ймовірність неконтрольованої зміни бітів. Проте цей метод має значні обмеження, оскільки призводить до зростання енергоспоживання та зниження загальної продуктивності системи. Тестування показало, що навіть при підвищенні частоти оновлення пам'яті в 2-3 рази, атака RowHammer все ще може спрацювати, хоча й потребує значно більшого часу.

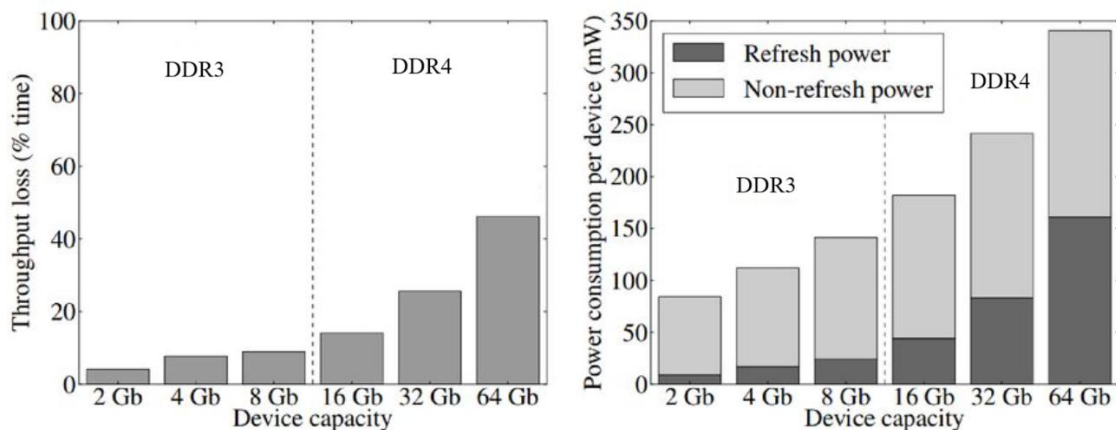


Рисунок 2.10 – Енерговитрати різних видів пам'яті

Як бачимо з рисунку 2.10. при збільшенні розміру пам'яті росте витрата енергії на її оновлення, при цьому збільшення швидкості в 2 рази не дає покращення виявлення помилок в 2 рази. Як бачимо з рисунку 2.11 при збільшенні частоти оновлення в 4 рази ймовірність захисту доходить до 90% для обох видів пам'яті DDR3 та DDR4. Це досі не дає гарантованого захисту при дуже значному збільшенні енерговитрат на пам'ять.

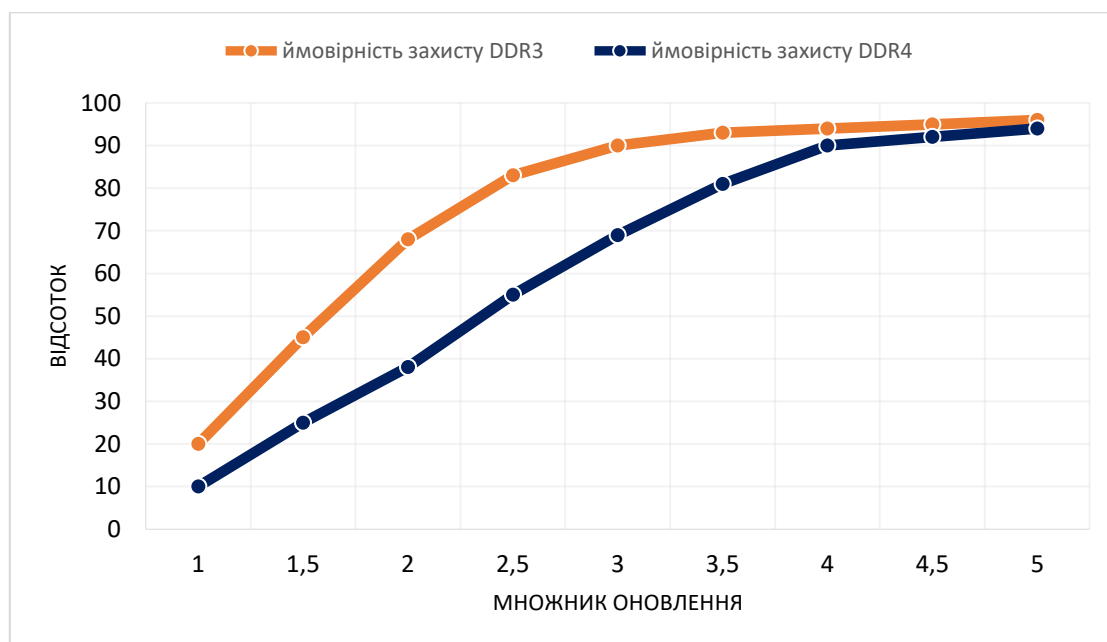


Рисунок 2.11 – Залежність захисту від атак відносно множника оновлення

Серед програмних механізмів захисту було протестовано алгоритми моніторингу аномальної активності пам'яті та обмеження доступу до певних областей. Аналіз показав, що більшість існуючих алгоритмів мають значні обмеження у виявленні складних варіантів RowHammer, особливо тих, що використовують розподілені патерни доступу або працюють у багатопоточному середовищі. Крім того, методи, що базуються на обмеженні доступу, можуть негативно впливати на продуктивність легітимних програм, особливо у випадку високонавантажених систем. Додатково, було досліджено ефективність програмних рішень, що використовують машинне навчання для виявлення RowHammer-подібних атак. Результати показали, що ці методи

можуть бути ефективними, але вони вимагають значних обчислювальних ресурсів та не завжди забезпечують необхідну точність при реальному навантаженні. Як бачимо з рисунку 2.12 при збільшенні кількості команд і двосторонньому підході можна досягти змін бітів обходячи всі наявні програмні методи захисту.

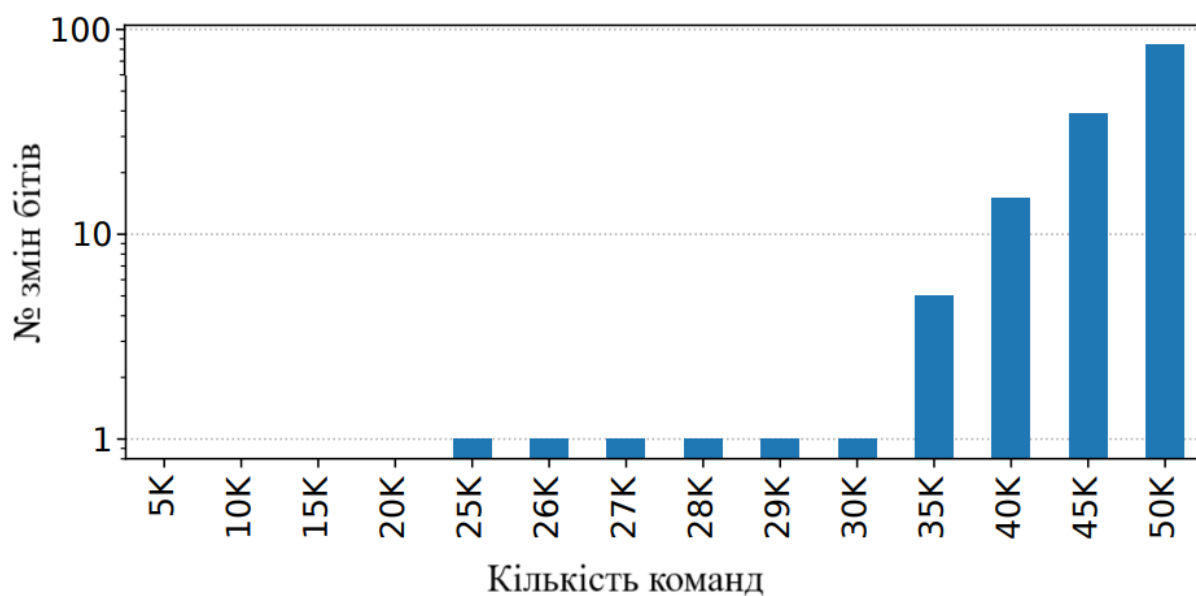


Рисунок 2.12 – Залежність зміни біті відносно кількості атак

На основі проведених експериментів було встановлено, що жоден із існуючих методів захисту не є абсолютно надійним проти атак типу RowHammer. Апаратні методи, такі як ECC та підвищене оновлення пам'яті, мають свої обмеження та не можуть повністю усунути загрозу. Програмні підходи, зокрема моніторинг аномальної активності та алгоритмічні обмеження, можуть бути ефективними лише в певних сценаріях, але мають значні недоліки у продуктивності та точності виявлення.

Таким чином, найбільш ефективною стратегією захисту є комбінований підхід, що поєднує апаратні та програмні методи, а також подальший розвиток адаптивних систем захисту, зокрема на основі машинного навчання. Подальші дослідження повинні бути спрямовані на розробку нових методик детектування

атак RowHammer, які забезпечуватимуть високу точність виявлення при мінімальному впливі на продуктивність системи.

Висновки до розділу 2

Отже у цьому розділі було створено платформу уніфікованого тестування засобів захисту пам'яті, що дозволяє проводити комплексний аналіз ефективності різних механізмів захисту від атак типу RowHammer на основі FPGA Xilinx ZCU208. Побудована універсальна програмна основа дала можливість стандартизувати підхід до тестування, що забезпечує порівнянність і уніфікованість отриманих результатів. Визначення критеріїв оцінювання ефективності захисту DRAM дозволило сформувати об'єктивні метрики для аналізу різних методів протидії атакам. Зокрема енерговитрата на комірку пам'яті, рівень збурень відносно джерела, радіус атаки, кількість АТС для зміни бітів та відповідно дозволена частота доступу при наявній атаці.

Процес збору та структуризації результатів тестування виявив низку обмежень існуючих апаратних і програмних рішень. Зокрема тестування показало, що сучасні чіпи DDR4 та DDR5 значно вразливі до атак двостороннього RowHammer та можуть легко бути скомпрометованими. Також, аналіз показав, що методи захисту в бюджетних лінійках всіх трьох протестованих виробників є недостатньо ефективними при роботі з сучасними DRAM-чіпами, що мають підвищену чутливість до паразитних збурень. Виявлені вразливості свідчать про необхідність подальшого вдосконалення механізмів захисту, зокрема розробки комбінованих підходів, що включають апаратні та програмні методи виявлення та пом'якшення впливу атак.

Отримані результати підтверджують, що існуючі рішення потребують доопрацювання та адаптації під новітні технологічні процеси у виробництві DRAM. Використання FPGA для тестування дозволило отримати детальні дані про роботу різних механізмів захисту та визначити їхні сильні та слабкі

сторони. Це створює основу для подальших досліджень і розробки більш надійних методів захисту пам'яті від атак RowHammer.

РОЗДІЛ 3.

РОЗРОБКА ПОКРАЩЕНИХ СИСТЕМ ЗАХИСТУ ПАМ'ЯТІ ТА ОЦІНКА ЇХ РОБОТИ

3.1 Розробка рішень оптимізації захисту на основі вибірки рядків

Захист на основі вибірки рядків (Row-Sampling) є одним із найперших і найпростіших методів протидії атакам типу RowHammer, що можна застосувати безпосередньо до контролера пам'яті. Його принцип роботи полягає в тому, що під час активації кожного рядка контролер генерує випадкове значення з певним відхиленням. З невеликою ймовірністю $p \ll 1$ цей рядок включається у вибірку та розглядається як потенційно атакуючий. У випадку, якщо рядок визначений як атакуючий, контролер виконує відповідні захисні дії, наприклад, оновлює сусідні рядки, які могли стати жертвами атаки. Такий підхід гарантує, що агресивний рядок не зможе уникнути виявлення та вибірки, завдяки чому атака RowHammer стає значно менш ефективною.

Цей метод захисту привернув увагу дослідників ще на ранніх етапах вивчення RowHammer-атак, що призвело до створення кількох його варіантів, таких як "імовірнісна активація сусідніх рядків" (PARA)[48] та "імовірнісна активація рядка" (PRA)[49]. Обидва ці підходи базуються на ідеї випадкової вибірки, але мають відмінності у способах реалізації та деталях адаптації до конкретних умов. Головною перевагою Row-Sampling є його простота, оскільки метод не вимагає складного зберігання станів або таблиць у пам'яті контролера, що значно відрізняє його від інших, більш ресурсомістких методів захисту, таких як використання лічильників або складних логічних операцій.

Серед значущих прикладів використання цього підходу можна згадати реалізацію вибірки рядків у старіших процесорах Intel для захисту DDR3

DRAM від RowHammer, яка отримала назву pTRR (probabilistic Target Row Refresh). Ця технологія була досить ефективною для DDR3, але зі зростанням частоти оновлення пам'яті DDR4 Intel вирішила відмовитися від підтримки pTRR, вважаючи, що стандартні характеристики DDR4 забезпечують достатній рівень захисту. Проте дослідження останніх років показали, що DDR4, як і раніше, вразлива до RowHammer-атак, причому новіші чіпи потребують ще меншої кількості операцій доступу до пам'яті, щоб спричинити зміну значень бітів.

Зважаючи на ці обставини, методи вибірки рядків знову набувають популярності як перспективний напрям для підвищення захисту DRAM. Виникає важливе питання: яке значення порогу p є оптимальним для забезпечення надійного рівня захисту? Відповідь на це питання має враховувати не тільки характеристики окремих модулів пам'яті чи поточну частоту оновлення, але й забезпечувати комплексний підхід, який враховує всю систему протягом усього терміну її експлуатації. Це потребує детального математичного моделювання, аналізу ризиків та глибокого розуміння змін у технологіях виробництва DRAM. Також, зважаючи на зростаючу складність і масштабність сучасних обчислювальних систем, важливо оцінювати ефективність методу Row-Sampling не лише з погляду безпеки, але й із погляду його впливу на продуктивність. Вибір надто високого значення p може збільшити накладні витрати на обробку, що знизить загальну пропускну здатність пам'яті. Тому основне завдання — знайти баланс між ефективністю захисту та впливом на продуктивність системи, враховуючи нові тренди в технологіях DRAM і вимоги сучасних обчислювальних навантажень.

Захисні механізми, створені для мінімізації RowHammer передбачають просту й уніфіковану модель DRAM. Кожен рядок після запису створює наводки в сусідніх рядах. Рівень спотворень на ряд жертви, як припускається, залежить виключно від його відстані від ряду агресора. Наприклад, ряд

агресора K впливає однаково на двох сусідніх жертв - рядки $K \pm 1$. При цьому K впливає $K \pm 2$ меншою мірою, ніж $K \pm 1$, $K \pm 3$ ще меншою, і так далі. А прискорення, з яким вплив наведень зменшується з відстанню, називається коефіцієнт послаблення (КП) а радіус дії (РД) вказує на відстань між рядом агресора та його найдальшою жертвою. Модуль DRAM із радіусом дії 2 означає що K збуджує тільки чотири ряди: $K \pm 1$ і $K \pm 2$.

Через це засоби захисту RowHammer, придатні для запису в контролер пам'яті, перш за все намагаються ідентифікувати ряди агресора. Їх мета полягає в тому, щоб ніколи не дозволяти йому отримувати більше активацій рядка, ніж фіксоване порогове значення, яке називається пороговим значенням RowHammer (A_{RH}), в межах інтервалу оновлення (64 мс у DDR4 та 32 мс у DDR5). Як тільки кількість активацій досягне A_{RH} , доступ блокується до наступного оновлення. Передбачається, що такий підхід нейтралізує все збурення, створене рядом агресора. Схеми вибірки рядків ж налаштовують поріг r так, щоб імовірність, що кількість запитів будь якого рядка дійде до A_{RH} буде дуже низькою. Проблема ж, що не дуже зрозуміло, якою має бути ця дуже низька ймовірність, хоча в деяких нещодавніх роботах описується значення 10–15 за годину безперервних записів [61]. Попередні схеми для Row Sampling [62], [63] припускали, що при досягнення A_{RH} контролер пам'яті також оновлює і рядків жертв. На жаль, внутрішня топологія рядків DRAM залишається комерційною таємницею постачальників DRAM. Тож контролер пам'яті нездатне ідентифікувати рядки жертв, на які впливає конкретний рядок агресора.

На практиці DRAM не поводитья так, як пропонує ця спрощена та уніфікована модель. Деякі рядки вимагають менше запитів, щоб викликати зміну бітів, ніж інші. Це відповідає необхідності мати індивідуальне A_{RH} для кожного рядка, а не єдине постійне значення для всієї DRAM у системі. Крім того, збурення DRAM не є рівномірними: деякі комірки мають більше шансів

бути зміненими, ніж інші, навіть якщо їх відстань до ряду агресорів однакова. Нарешті, радіус дії змінюється в залежності від ряду атакуючих; деякі ряди мають більший радіус дії, ніж інші. Саме тому справжня модель DRAM має налічувати всі ці змінні щоб аналізувати захист RowHammer з математичним підходом та реальними результатами в існуючих системах. При цьому така система може бути спрощена і до глобальних значень, але потрібно обирати найвищий поріг p з усіх можливих для чіпу пам'яті і найнижчий TH_{RH} після тестувань всіх рядків. Одна з перших робіт [59], присвячених Row-Sampling, містить виведення формули для обчислення ймовірності невдалої атаки RowHammer при заданих параметрах p і A_{RH} , а також вводить поняття "раунди" (позначаються як r). Раунди описують час, необхідний зловмиснику для досягнення порогу A_{RH} шляхом послідовної активації рядків, незалежно від тривалості вікна оновлення пам'яті.

$$P_{\text{невдачі}} = 1 - (1 - e^{-p \times A_{RH}})^k \quad (3.1)$$

На жаль, ця формула дуже применшує значення порогу вибірки. Його визначення ґрунтується на припущенні, що кожен із раундів є незалежним і жодні збурення не передаються від одного раунду до наступного. Але на практиці, атака, яка активує половину рядків наприкінці раунду, а іншу половину одразу на початку наступного, має високий шанс уникнути вибірки. Тобто ці події потрібно розглядати як взаємозалежні.

Для обчислення порогу в PARA [48] використовують такі формули:

$$P(e_N) = P(e_{N-1}) + p \left(1 - \frac{1}{2}p\right)^{A_{RH}} (1 - P(e_{N-A_{RH}-1})) \quad (3.2)$$

$$P(e_N) = 0 \text{ при } N < A_{RH} \quad (3.3)$$

Де e_N випадок успішної атаки. Умова 2 тут тривіальна і означає що атака не сталась при кількості запитів меншій, ніж та яка встановлена як порогова на контролері пам'яті DRAM. З цього випливає, що в такому випадку ймовірність

поломки RowHammer дорівнює нулю, тобто пам'ять було успішно захищено в даному раунді і до наступного всі значення в конденсаторах комірок пам'яті буде оновлено завдяки автоматичним процесам.

Як було вище описано, атака RowHammer вимагає виконання двох умов:

- A_{RH} активації рядків що не потрапили до вибірки
- відсутність автоматичного оновлення рядків жертви.

Так як ми не приймаємо за даність автооновлення рядків жертв, то можемо ще додати параметр $P(v_{TH})$ що показує таку ймовірність. Ця ймовірність пропорційна відношенню двох часових інтервалів: частина вікна оновлення, яка виходить за межі A_{RH} ($t_{атак}$) та часового інтервалу вікна оновлення ($t_{онов}$).

$$P(v_{TH}) = \frac{t_{онов} - t_{атак} \times A_{RH}}{t_{онов}} \quad (3.4)$$

Оскільки дві ймовірності є незалежними то ймовірність невдачі в даному випадку є їх добутком. Додатково формула ймовірності вдалої атаки RowHammer має враховувати всю систему (тобто не лише окремий блок пам'яті) і тривалість атаки. Таким чином, ми вводимо два додаткові параметри: загальна кількість блоків b у всій системі, які можуть бути одночасно атаковані RowHammer та A , загальна максимальна кількість активацій рядків у блоці протягом атаки. Отримавши ймовірність успішної атаки для окремого блоку, ми можемо масштабувати її до всієї системи, бо навіть один скомпрометований блок пам'яті вже є загрозою для нас. Таким чином, якщо успішну атаку на один блок позначити P_1 то для всієї системи формула стає $P_{системи} = 1 - (1 - P_1)^b$

При цьому, як було вказано вище, ймовірність цих двох родій незалежна, тому використовуючи формули 3.1, 3.2, 3.3, 3.4 і розписавши для незалежних вищеописаних ймовірностей отримаємо:

$$P_{системи} = 1 - (1 - P(e_A) \times P(v_{TH}))^b \quad (3.5)$$

Де $P(e_A)$ знаходиться з системи 3.2, 3.3 використовуючи настроювані значення, які можна буде підібрати для найкращої роботи, а $P(v_{TH})$ отримуємо з формули 3.4.

При цьому радіус дії не фігурує в формулі безпосередньо, але після калькуляції саме це порогове значення ми використовуємо для всіх рядків в глобально-зазначеному системою радіусі дії навколо атакуючого рядка. Результати імплементації такого методу можна бачити на рисунку 3.1

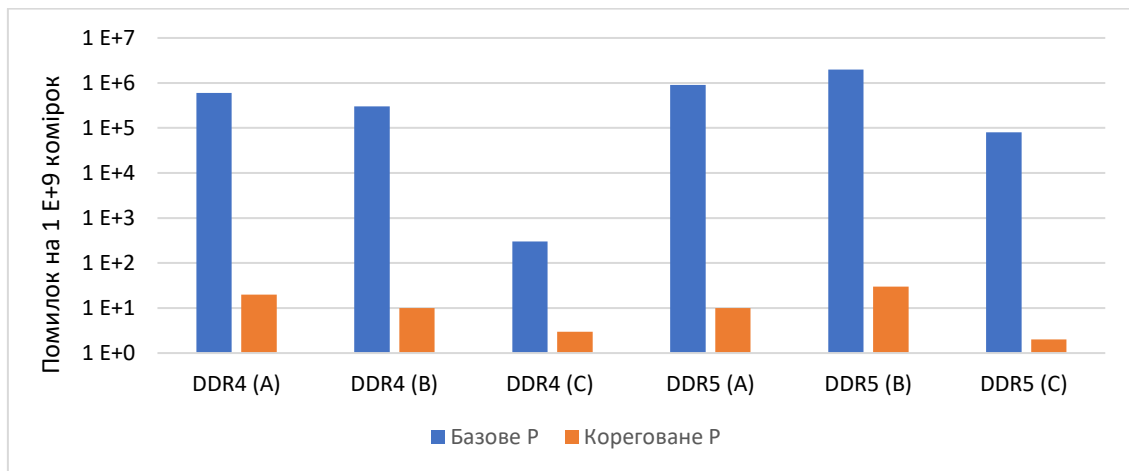


Рисунок 3.1 – Порівняння кількості RowHammer помилок для різних p

Як бачимо з порівняння ймовірність відмови Rowhammer для різних порогів p для конфігурацій А та В. Для частот дискретизації ми використовували зворотні величини ступеня двійки (тобто 1 з 32, 1 з 64), оскільки ми очікуємо, що такі частоти дискретизації будуть легкими реалізувати в контролері пам'яті. Однак наша формула та код можуть працювати з будь-якими значеннями частоти дискретизації. Ми також використовували низькі значення порогу, що відповідають останнім тенденціям, які показують, що нові комірки DRAM, що вимагають менше звернень до пам'яті, поки біти не почнуть змінюватись [5]. Графік ілюструє, що при правильній регуляції порогу вибірки залежно від апаратних конфігурацій, можливо значно підвищити захищеність системи від атак типу RowHammer. Наприклад для DDR5 розробника С поріг був підвищений достатньо, щоб

вдалось зменшити кількість спотворених бітів на 89.5%, при цьому не перевищивши вимоги по енергоефективності.

3.2 Зменшення витрати пам'яті в системах захисту на основі лічильників

Додатково на рівні методу вибірки рядків, що працює на ймовірнісному визначенні, ефективним підходом є використання захисних механізмів та методів, заснованих на лічильниках, які інтегруються в апаратну мікроархітектуру модулів пам'яті. Ці методи відстежують кількість активацій рядків пам'яті, що дозволяє визначити потенційно уразливі рядки до моменту, коли вони стають агресорами. Такий підхід забезпечує високий рівень захисту, проте має свої обмеження, зокрема, пов'язані з додатковими витратами на зберігання даних лічильників. Такі методи захисту зазвичай реалізуються на рівні банків (блоків) пам'яті.

У такій конфігурації кожен банк має свій набір лічильників, які зберігають інформацію про частоту активацій рядків у межах банку. Це дозволяє швидко реагувати на потенційні загрози, однак, знову ж таки, вимагає додаткових апаратних ресурсів для зберігання цих лічильників. Загальні накладні витрати на кремнієву площу при цьому зростають пропорційно до кількості банків у модулі пам'яті. Для великих конфігурацій DDR4 та DDR5 це може створювати значне навантаження на обмежені ресурси апаратної частини, особливо в системах, де економія простору та енергоефективність є критичними. До прикладу мобільні та низько рівневі пристрої.

Використання індивідуальних лічильників на рівні банку в методах захисту пам'яті від RowHammer супроводжується додатковими витратами часу, пов'язаними з їх роботою. Основна причина цих затримок полягає в необхідності регулярного оновлення значень лічильників, особливо під час інтенсивного доступу до пам'яті. У сучасних системах пам'яті DRAM доступ

до банків здійснюється паралельно для підвищення продуктивності. Але інтеграція лічильників створює додаткове навантаження на контролер пам'яті, оскільки потрібно виконувати додаткові операції підрахунку та перевірки кожного разу при активації рядка. Одним із ключових аспектів таких затрат є збільшення часу на обробку запитів до пам'яті. Контролер має не лише обробити стандартні команди читання та запису, а й оновити відповідні значення лічильників, а також виконати порівняння з пороговими значеннями, щоб визначити, чи потрібно вживати захисних заходів, таких як оновлення сусідніх рядків. Це може призводити до збільшення затримок, особливо у випадках, коли кілька банків одночасно активні й потребують синхронної обробки даних.

Ще одним фактором є обмеження, пов'язані з апаратною реалізацією лічильників. Для кожного банку необхідно виділяти частину ресурсів на зберігання лічильників, їхнє оновлення та логіку перевірки. Це створює додаткове навантаження на внутрішню шину пам'яті, яка обслуговує обмін даними між контролером і модулями DRAM. У високопродуктивних системах це може спричинити зниження загальної пропускної здатності через конкуренцію між стандартними операціями пам'яті та процесами обслуговування лічильників. Додатково через високу інтенсивність операцій із лічильниками контролер пам'яті може генерувати зайві команди оновлення сусідніх рядків навіть у випадках, коли це не є критично необхідним. Це збільшує кількість тактів, витрачених на захисні заходи, що в свою чергу знижує доступність пам'яті для виконання основних завдань. У багатозадачних середовищах такий підхід може створювати затримки в обслуговуванні інших процесів, які також використовують ресурси пам'яті.

Щоб зменшити витрати на апаратну реалізацію, це дослідження звертає увагу на можливість перенесення підрахунку значень доступу з рівня банку на рівень рангу. Тобто розширити рівень охоплення логічних зв'язків що

дозволить у цьому підході лічильникам охоплювати більші області пам'яті. У свою чергу це дозволить суттєво скоротити кількість таких лічильників, що вплине як на використання пам'яті загалом так і на швидкодію. Наприклад, один набір лічильників на ранг замінює кілька наборів для кожного банку в межах цього рангу. Такий перехід дозволяє зменшити вимоги до обсягу пам'яті, необхідної для зберігання лічильників, і одночасно знижує накладні витрати на площу і щільність пам'яті. Однак цей підхід вимагає ретельного аналізу впливу на ефективність захисту, оскільки більша деталізація може призвести до зниження чутливості до атак.

Зменшення деталізації підрахунку до рівня рангу дозволяє суттєво скоротити кількість лічильників і, відповідно, знизити апаратні витрати. Це особливо важливо для систем, де обмежені ресурси пам'яті та корисної обчислювальної площі є критичними, наприклад, вбудовані системи, сервери або пристрої з низьким енергоспоживанням. Однак така зміна деталізації підрахунку може вплинути на точність виявлення потенційних загроз, адже узагальнені дані на рівні рангу можуть не враховувати локальні аномалії, характерні для окремих банків або рядків пам'яті. Це вимагає використання додаткових механізмів контролю, таких як адаптивні алгоритми або розподілені системи моніторингу.

Такий підхід до деталізації також створює нові виклики для оптимізації параметрів лічильників. Наприклад, якщо інтенсивність використання пам'яті нерівномірно розподілена між банками всередині рангу, це може призводити до помилкових результатів. У таких випадках необхідно розробляти алгоритми, які динамічно враховують навантаження та коригують параметри роботи лічильників залежно від характеру доступу до пам'яті. Крім того, зменшення деталізації може бути ефективним тільки за умови, що інші механізми захисту, такі як періодичне оновлення рядків (refresh), працюють надійно й узгоджено з лічильниками. Ще одним аспектом є часові затримки, пов'язані з

масштабуванням підрахунку на рівень рангу. Через більший обсяг даних, які потрібно обробити, швидкість реагування на потенційні атаки може зменшитися, що підвищує ризик успішної реалізації RowHammer-атак. Для вирішення цієї проблеми можуть бути застосовані спеціалізовані апаратні прискорювачі або оптимізація структури лічильників, наприклад, використання багаторівневих схем зберігання, які поєднують швидкодію та економію ресурсів.

3.2.1 Деталізація підрахунку доступу на рівні банку та рангу

Як було показано вище, мета атак RowHammer полягає в тому, щоб надіслати достатню кількість команд запису (ACT) сусіднім рядкам жертви DRAM без прямого доступу до нього, щоб пошкодити, чи змінити біти в ньому. Через такі обмеження, як невідомість макету пам'яті, що тримається як виробнича таємниця, алгоритми на основі лічильників не можуть виміряти рівень збурення рядків «жертв», оскільки вони не знають, фізичне розташування того чи іншого рядка. Натомість вони намагаються виявити, коли рядки використовуються як агресори, підраховуючи їх активації. Будь який рядок пам'яті має бути класифікований механізмами захисту як рядок-агресора, перш ніж йому буде видано достатньо ACT, щоб пошкодити сусідній рядок жертви. Оскільки при двосторонньому RowHammer два сусіди рядки можуть використовуватися, щоб атакувати один, то при пороговому A_{RH} команд, потрібних для атаки один атакуючий рядок отримає $A_{aggr} = \frac{A_{RH}}{2}$ команд.

Загалом, витрати пам'яті, необхідні для гарантування виявлення атаки Row-Hammer, залежать від бітової ширини кожного лічильника та загальної кількості лічильників. Кількість бітів на лічильник S безпосередньо залежить від порогу виявлення, який розраховується з порогу атаки A_{RH} . В ідеалі, нам потрібно було б ввести лічильник взагалі для кожного рядка окремо. Однак мати один лічильник на рядок надто дорого в низькорівневому виконанні на

контролері (напр., типовий банк DDR4 має $2^{16} = 64\text{K}$ рядків, які вимагатимуть приблизно 1 МБ пам'яті лічильників на банк для T_{RH} з максимальним значенням 32768). Через ці обмеження, більшість існуючих механізмів виявлення на основі лічильників включають методи мінімізації їх кількості, при цьому все ще гарантуючи високий рівень виявлення. Параметрами цих методів є поріг виявлення і W - максимальна кількість команд до контролера, які можуть бути виконані протягом часу до наступного оновлення t_{REFW} .

Поріг виявлення визначає вразливість пам'яті до збурень і фіксується після виготовлення виробником. Звісно неможливо знайти таке значення для кожної комірки індивідуально, але як показують тестові дані цей поріг є середнім значенням для всіх рядків. Знову ж таки, точно цієї інформації знайти не можливо через комерційну таємницю виробників пам'яті. W визначається таймінгами DRAM і глибиною підрахункового таймера. На рівні банку активації рядків обмежені інтервалом між двома АСТ - t_{RC} , і періодичними оновленнями, що визначаються за допомогою t_{REFI} і t_{RFC} . Отже, W на рівні банку W_B можна розрахувати за формулою:

$$W_B = \left\lceil \frac{t_{REFW} \times \left(1 - \frac{t_{RFC}}{t_{REFI}}\right)}{t_{RC}} \right\rceil \quad (3.6)$$

Крім того, враховуючи глибину лічильника на рівні рангу, активації рядків не обмежуються лише виключно W_B банків. Кількість команд активації для рядка в ранзі обмежений чотирма разами протягом t_{FAW} . Як наслідок, не до всіх банків можна отримати доступ із максимальною частотою.

Технології DDR3 і DDR4 використовують механізм оновлення всіх банків, який оновлює всі банки одночасно під час виконання команди REF, утримуючи ранг зайнятим протягом t_{RFC} кожен t_{REFI} . Стандарт DDR5 представив однобанкову команду REF [65], що дозволяє оновлювати банки рангу індивідуально, для того щоб залишати доступними інші банки пам'яті, так як

$t_{RFC} \ll t_{REFI}$. Отже, значення W на рівні рангу W_P для пам'яті DDR3 і DDR4 можна розрахувати за формулою:

$$W_P = \left[\frac{t_{REFW} \times \left(1 - \frac{t_{RFC}}{t_{REFI}}\right)}{t_{FAW} \div 4} \right] \quad (3.7)$$

а W_P для пам'яті DDR5 набуває вигляду:

$$W_P = \left[\frac{t_{REFW}}{t_{FAW} \div 4} \right] \quad (3.8)$$

Тепер якщо ми візьмемо N_b як кількість банків на ранг, загальна кількість лічильників у ранзі для виявлення атак на рівні банку буде дорівнювати добутку N_b та кількості лічильників на банк. Якщо кількість лічильників пропорційна W , а ефективне значення такого порогу для суми всіх механізмів виявлення на рівні банку в ранзі є $N_b \times W_B$. Отже, коли $N_b \times W_B > W_P$, механізм визначення атаки на ранговому рівні має нижчу ефективність W ніж сума всіх механізмів виявлення на рівні банків пам'яті. Як наслідок, перехід до переміщення лічильників доступу на рівень рангу може зменшити кількість лічильників, необхідних для захисту пам'яті від атак RowHammer.

Для модулів пам'яті різного типу, протестованих у розділі 2, значення W_B , W_P наведені в таблиці 3.1. У цій таблиці погіршення рівня W розраховується за формулою 3.9:

$$W_{\text{погір}} = 1 - \frac{W_P}{W_B \times N_B} \quad (3.9)$$

Ця таблиця показує, що в міру того, як технологія розвивається і стандарт дозволяє використовувати більше банків на ранг, зменшення W під час переходу від рівня банку до рівня рангу деталізація підрахунку і погіршення порогу збільшується з 19% для DDR3 до 62% для DDR5.

Таблиця 3.1 – Теоретичне зменшення загальної кількості лічильників для DDR3, DDR4 і DDR5 при переході з банку до рангу

Пам'ять	W_B	W_P	N_B	$W_{\text{погір}}$
DDR3	1.25×10^6	8.15×10^6	8	19%
DDR4	1.33×10^6	11.3×10^6	16	47%
DDR5	6.61×10^5	8.00×10^6	32	62%

3.2.2 Перехід до лічильників рівня рангу

Для кількісної оцінки зменшення використання пам'яті в результаті знайденого і обрахованого W ми розглядаємо два нещодавно опубліковані контрзаходи, а саме Graphene [60] і BlockHamer [61]. Оскільки ці контрзаходи були призначені лише для пам'яті DDR4, результати в цьому розділі обмежуються DDR4.

Graphene зберігає адреси рядків і лічильники в асоціативній пам'яті (АП) і використовує алгоритм Місра-Гріса [62] для підрахунку лише АСТ на $N_{\text{записів}}$ для рядків банку, що були активовані найчастіше. Мінімальна кількість записів у АП відповідно до публікації розраховується за формулою:

$$N_{\text{записів}} = \left\lfloor \frac{W}{A_{RH} \div 4} \right\rfloor \quad (3.10)$$

Запис у АП при цьому містить пару ключ-значення, де ключ є адресою рядка, а значенням є лічильник, плюс один біт переповнення для лічильника. Цей біт переповнення використовується як тригер для виявлення, коли жертва була активована занадто багато разів, і її сусідам потрібно оновити свої значення. Лічильник має бути здатний зберігати значення до $A_{\text{aggr}} / 2 - 1$. Для типового порогу збурення $T_{RH} = 32768$ [9], це означає максимальне значення на

лічильник $A_{\text{aggr}} / 2 - 1 = 8191 = 2^{13} - 1$, отже, лічильник 13 біт. Адресна частина запису повинна мати можливість ідентифікувати кожен рядок. Враховуючи деталізацію підрахунку на рівні банку, як запропоновано в оригінальній публікації, і банки DDR4 із 65536 рядками, адресна частина запису матиме всі 16 біт. Таким чином для загального запису однієї комірки захисних даних потрібно: $C = 16 + 13 + 1 = 30$ бітів. Відповідно до формули 3.10 з $W = W_B$, прототип банку пам'яті DDR4 з таким захистом потребуватиме 162 записи в АП. Таким чином, загальний розмір АП становить $16 \text{ банків} \times 162 \text{ записи} \times 30 \text{ бітів} = 9.49 \text{ KiB}$ на ранг (5.06 KiB для ключів, 4.43 KiB для решти даних).

З іншого боку, враховуючи деталізацію підрахунку на рівні рангу, записи вимагатимуть більше місця для відповідності унікальній адресі рядка. Оскільки ранг DDR4 може містити до 16 банків, адреса рядка має бути на 4 біти більшою за еквівалент на рівні банку, щоб однозначно ідентифікувати кожен рядок. У результаті загальний розмір запису стає $C' = 20 + 13 + 1 = 34$ біти. Відповідно до формули 3.10 з $W = W_R$, прототип захищеного рангу пам'яті DDR4 матиме 1394 записи в АП. Таким чином, загальний розмір АП становитиме $1394 \text{ записи} \times 34 \text{ біти} = 5.71 \text{ KiB}$ на ранг (3.36 KiB для ключів, 2.35 KiB для решти даних).

Для одного рангу додатковий обсяг пам'яті, який вимагає Graphene з глибиною підрахунку доступів на рівні рангу, на 40% менший, ніж така ж сума для 16 додаткових незалежних банків, необхідних для Graphene з глибиною підрахунку доступів на рівні банку.

BlockHammer з іншого боку використовує фільтри Блума з підрахунком (ФБП) [71], щоб підрахувати кількість АСТ для кожного рядка. ФБП працює, пов'язуючи кожне можливе вхідне значення з унікальним набором k лічильників із загальної кількості M лічильників, використовуючи хеш-функцію. Коли рядок активовано, ФБП використовує хеш-функцію для визначення асоціації k лічильників та збільшує їх. Коли рядок активований і всі

пов'язані з ним лічильники перевищують попередньо визначене порогове значення H_{BL} , механізм розглядає ряд як агресора. Він передає команду процесору, щоб уповільнити процес, який активував рядок-агресора востаннє, щоб дочекатись доки рядок не оновиться. Кожен t_{REFW} усі лічильники ФБП скидаються для врахування періодичності оновлення рядків пам'яті. Щоб не пропустити атаку, яка не була б націлена на рядок, синхронно оновлений зі скиданням усіх лічильників, Block-Hammer використовує два ФБП. Вони по черзі оновлюються кожен раз $t_{REFW} / 2$, і обидва збільшують своє значення при АСТ, але одночасно фігурує в розрахунках лише один. Принцип роботи цього механізму показано на рисунку 3.2.

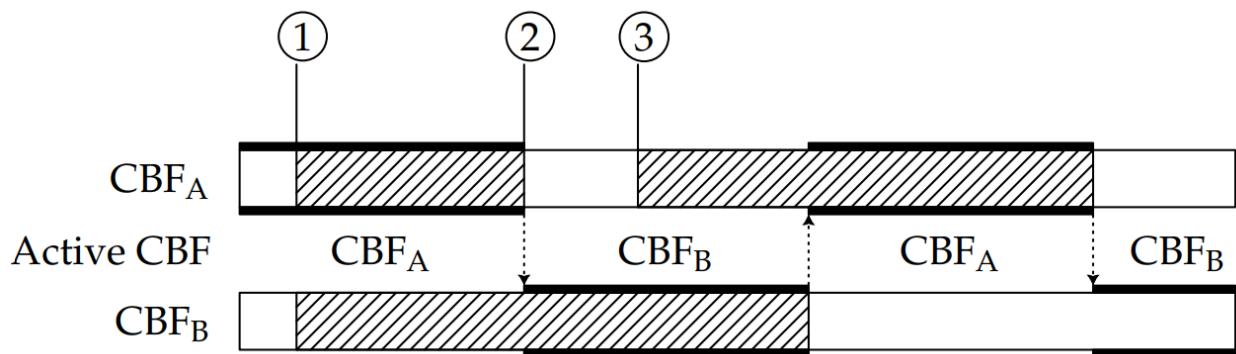


Рисунок 3.2 – Чергування ФБП на BlockHammer.

Коли активний ФБП виявляє агресора (1 на рисунку 3.2), він заносить рядок до чорного списку. Кожен $t_{REFW} / 2$, активний ФБП очищається і деактивується та обчислення переходять на інший ФБП (2 на рисунку 3.2). Якщо неактивний ФБП виявляє агресора (3 на рисунку 3.2), цей результат не враховується до його наступної активації.

Цей механізм використовує скорочений набір лічильників для підрахунку доступу. Оскільки лічильники є спільними для кількох рядків в банку, механізм виявлення має бути досить витривалим і розбірливим, щоб мати змогу відрізнити рядки-агресори від інших звичайних рядків, до яких здійснюється нечастий доступ. Для зрозумілого і репрезентативного способу оцінити

здатність BlockHammer відрізнити рядки-агресори від доброякісних рядів, ми вводимо змінну рівень шуму - n . Рівень шуму - це середнє значення, досягнуте лічильниками після W АСТ рівномірно розподілених по всіх рядках розглянутої в даний момент пам'яті. Він повинен бути значно нижчим за поріг виявлення H_{BL} щоб лічильники, пов'язані з рядками-агресорами, виділялися серед лічильників, збільшених безпечними доступами до пам'яті. Таким чином значення n буде розраховуватися за наступною формулою:

$$n = \frac{W \times k}{m} \quad (3.11)$$

Реалізація на рівні банку, запропонована авторами BlockHammer, включає $M = 1024$ лічильників на ФБП, $k = 3$, і $H_{BL} = 8192$. Для розглянутої DDR4 с $W_B = 1.33 \times 10^6$, маємо рівень шуму $n \approx 3896$. Це означає, що під час звичайної роботи, яка потребує багато пам'яті, більшість лічильників, швидше за все, досягнуть цього значення. Механізм може легко розрізнити агресорські ряди, кількість доступу до котрих буде досягати значення H_{BL} , що приблизно вдвічі перевищує знайдене значення значення n .

Розглядаючи реалізацію перенесення даних лічильників до рангового рівня, рівень шуму повинен набувати ідентичного значення, щоб забезпечити ідентичний рівень захисту. Так можна зйти $W_P = 11.3 \times 10^6 \approx 8.5 \times W_B$. При цьому ми обрали $M = 8192$ і $k = 3$. Також для простоти ми вирішили зберегти константу k , оскільки це і так вже дуже невелике число. Ми вибрали M як найближчий ступінь двійки, щоб зберегти подібне значення шуму. З цими значеннями ми маємо рівень шуму $n \approx 4138$.

Кількість бітів на лічильник визначається лише з H_{BL} , тому незмінною залишається деталізація підрахунку на рівні банку та рівні рангу. Оскільки стандарт DDR4 має 16 банків на ранг, а ФБП на рівні рангу має у 8 разів більше лічильників порівняно з ФБП на рівні банку, загальний розмір, зайнятий

лічильниками для рівня деталізації підрахунку на рівні рангу, вдвічі менший за ідентичні підрахунки але вже на рівні банку.

Очевидно, що перенесення підрахункових лічильників доступу з рівня банку на рівень рангу в пам'яті типу DDR4 дозволяє значно зменшити обсяг додаткової пам'яті, необхідної для реалізації захисних механізмів, таких як Graphene і BlockHammer. Зокрема, економія може досягати 40–50%, що є суттєвим показником для зниження апаратних витрат і підвищення ефективності використання ресурсу. Такий підхід відкриває перспективи для оптимізації захисту без втрати ефективності, оскільки на рівні рангу можливе більш широке використання загальних ресурсів для кількох банків, що мінімізує дублювання даних у пам'яті. Також важливо підмітити, що методології, спочатку розроблені для DDR4, є достатньо універсальними, аби їх можна було адаптувати до інших поколінь пам'яті DDR, таких як DDR3 і DDR5. Перенесення лічильників на рівень рангу не лише забезпечує економію пам'яті, але й дозволяє стандартизувати підходи до захисту між різними поколіннями DRAM. Це особливо важливо в умовах швидкого розвитку пам'яті, оскільки виробники прагнуть досягти високої щільності розташування комірок, що збільшує ризик уразливостей, зокрема до атак RowHammer.

Розрахунки в таблиці 3.1 демонструють, як зменшення обсягу додаткової пам'яті прямо залежить від коефіцієнта W , що характеризує кількість лічильників доступу для кожного банку чи рангу. Для новіших модулів пам'яті, таких як DDR5, зниження значення W дозволяє не лише зменшити обсяг додаткової пам'яті, але й покращити масштабованість системи. Це має вирішальне значення для збереження конкурентоспроможності виробників DRAM, оскільки технології пам'яті стають дедалі складнішими. У таблиці 3.2 представлені прогнозовані обсяги економії пам'яті для DDR3, DDR4 і DDR5 при переході на підрахункові лічильники на рівні рангу для Graphene і BlockHammer. Згідно з цими даними, адаптація захисних механізмів до рівня

рангу забезпечує істотну оптимізацію без значних змін у загальній архітектурі контролера пам'яті. Крім того, ці результати демонструють, що універсальний підхід до зниження обсягів пам'яті може стати ключовим фактором у розробці майбутніх поколінь DRAM, де високі щільності та енергетична ефективність будуть основними вимогами ринку.

Таблиця 3.2 – Зменшення загальної кількості зайнятої лічильниками пам'яті для різних типів захисту DDR3, DDR4 і DDR5

		Рівень банку	Рівень рангу	Зменшення зайнятої пам'яті
DDR3	Загальний $W (8 \times W_B, W_P)$	10×10^6	8.15×10^6	19%
	Graphene АП	4.45KiB	4.00KiB	11.2%
	BlockHammer ФБП	26KiB	26KiB	0%
DDR4	Загальний $W (16 \times W_B, W_P)$	21.28×10^6	11.3×10^6	47%
	Graphene АП	9.61KiB	5.79KiB	40%
	BlockHammer ФБП	52KiB	26KiB	50%
DDR5	Загальний $W (32 \times W_B, W_P)$	21.15×10^6	8×10^6	62%
	Graphene АП	9.38KiB	4.05KiB	57%
	BlockHammer ФБП	52KiB	19.5KiB	62.5%

Одним із ключових питань, яке постає при зміні деталізації підрахунку з рівня банку на рівень рангу, є здатність контрзаходу ефективно витримувати скорочену затримку між двома активаціями рядків (АСТ). У прототипі пам'яті DDR4 на рівні банку контролер пам'яті здатний видавати одну активацію кожні 45 нс. Цього часу достатньо для того, щоб більшість лічильників виконували всі необхідні операції. Проте, на рівні рангу середня затримка між активаціями зменшується до значення $t_{FAW}/4 \approx 5.42$ нс. Це створює значні виклики для

реалізації ефективних механізмів захисту, зокрема для тих, що базуються на підрахункових лічильниках.

Наприклад, BlockHammer демонструє затримку всього 1 нс перед видачею кожної активації, що дозволяє йому функціонувати навіть за умов скороченої затримки між двома активаціями на рівні рангу. Утім, інші механізми, зокрема ті, що використовують апаратну підтримку пошуку (АП), як це робить Graphene, стикаються зі значними труднощами. Швидкість роботи таких механізмів помітно знижується із збільшенням їхнього масштабу та навантаження на лінії електропередач. Через це поточні проєкти АП можуть виявитися нездатними підтримувати необхідний рівень швидкодії для реалізації захисту на рівні рангу. Аналогічні проблеми були зазначені у попередніх дослідженнях, зокрема в пропозиції CAT-TWO, де автори прямо вказували, що знижена частота роботи АП обмежує їхню ефективність при переході до підрахунків на рівні рангу.

Попри ці труднощі, існує значна перспектива розвитку майбутніх проєктів АП, які зможуть забезпечити достатню швидкість пошуку. Одним із потенційних рішень є використання конвеєрної архітектури для АП. У конвеєрних схемах затримки пошуку не впливають на час видачі активації, оскільки всі операції виконуються паралельно. Це означає, що пропускна здатність залишається ключовим параметром, а не затримка обробки. Крім того, конвеєрна архітектура дозволяє поступово зменшувати кількість активних записів під час пошуку, що суттєво знижує навантаження на лінії електропередач і збільшує пропускну здатність.

Завдяки цим перевагам конвеєрні АП можуть значно зменшити обмеження на швидкість роботи механізмів виявлення RowHammer. Навіть за умови збільшення складності апаратної реалізації, це рішення здатне забезпечити як високу швидкодію, так і надійність, необхідні для ефективного захисту в сучасних і майбутніх системах пам'яті. Таким чином, інновації у проєктуванні

АП відкривають нові можливості для адаптації механізмів захисту RowHammer до умов, які вимагають більш високої продуктивності та меншої затримки.

3.3 Модель захисту на основі машинного навчання

Використання машинного навчання (ML) для виявлення атак RowHammer є достатньо свіжим підходом, який лише починає активно досліджуватися науковою спільнотою. Цей метод пропонує адаптивність і можливість аналізувати складні взаємозв'язки між параметрами доступу до пам'яті, що важко виявити традиційними засобами. Основна ідея полягає в тому, щоб використовувати алгоритми ML для аналізу великих обсягів даних, які генеруються під час роботи DRAM, з метою виявлення шаблонів або аномалій, характерних для RowHammer-атак. Машинне навчання може бути корисним для створення систем виявлення на основі моніторингу поведінки пам'яті в реальному часі. Алгоритми, такі як нейронні мережі, деревоподібні моделі рішень або методи кластеризації, можуть навчатися на великих наборах даних, які включають як звичайні операції доступу до пам'яті, так і дії, що імітують атаки RowHammer. Після тренування такі системи здатні автоматично класифікувати доступи до пам'яті як потенційно небезпечні або безпечні, навіть якщо атака маскується під звичайну активність.

Попри значний потенціал, технологія ML для виявлення RowHammer поки що перебуває на ранніх стадіях розвитку. Вона стикається з низкою викликів, таких як необхідність обробки великих обсягів даних у реальному часі, обмеженість ресурсів обчислювальної системи та складність налаштування моделей для роботи в різних апаратних середовищах. Крім того, точність моделей може залежати від якості й різноманітності навчальних даних, що потребує ретельного збору та обробки інформації. Іншим важливим аспектом є адаптивність системи виявлення. Сучасні атаки RowHammer постійно еволюціонують, і зловмисники використовують нові техніки для обходу

захисту. Це означає, що системи машинного навчання повинні бути здатні оновлюватися і навчатися на нових даних для збереження своєї ефективності. Розробка таких динамічних рішень потребує як технічних інновацій, так і ресурсів для інтеграції в існуючі системи.

Наскільки нам відомо, це перша пропозиція механізму виявлення RowHammer на основі ML, яка спрямована на апаратну реалізацію. В публікації на схожу тему професор Чакрабарті та ін. [45] запропонував механізм захисту на основі програмного забезпечення, який відстежує рівень пропусків LLC для виявлення підозрілих процесів. Потім він записує банк DRAM і доступ до рядків у цьому процесі та використовує згортову нейронну мережу (CNN) для класифікації шаблон доступу від процесу атаки чи ні. Незважаючи на те, що це рішення не потребує модифікації апаратного забезпечення, для виявлення процесу атаки після її початку потрібно багато часу (в середньому 1,5 с). Однак атаки RowHammer повинні виконувати зміну бітів менш ніж $t_{REFW} = 64$ мс, а процес, який уже має доступ до агресорів, близьких до бажаної жертви, може створити біт-фліп менш ніж за 10 мс [4]. Тому, вимагаючи 1,5 с для виявлення процесів атаки за шаблоном доступу до DRAM здається недостатнім.

3.3.1 Вибір моделі машинного навчання

Машинне навчання (ML) — це підгалузь штучного інтелекту, яка фокусується на створенні алгоритмів і моделей, здатних навчатися з даних і покращувати свої результати без необхідності жорсткого програмування. У цьому процесі комп'ютерна система аналізує дані, виявляє закономірності та використовує їх для прогнозування або прийняття рішень. Види різних моделей машинного навчання можна побачити на рисунку 3.3.

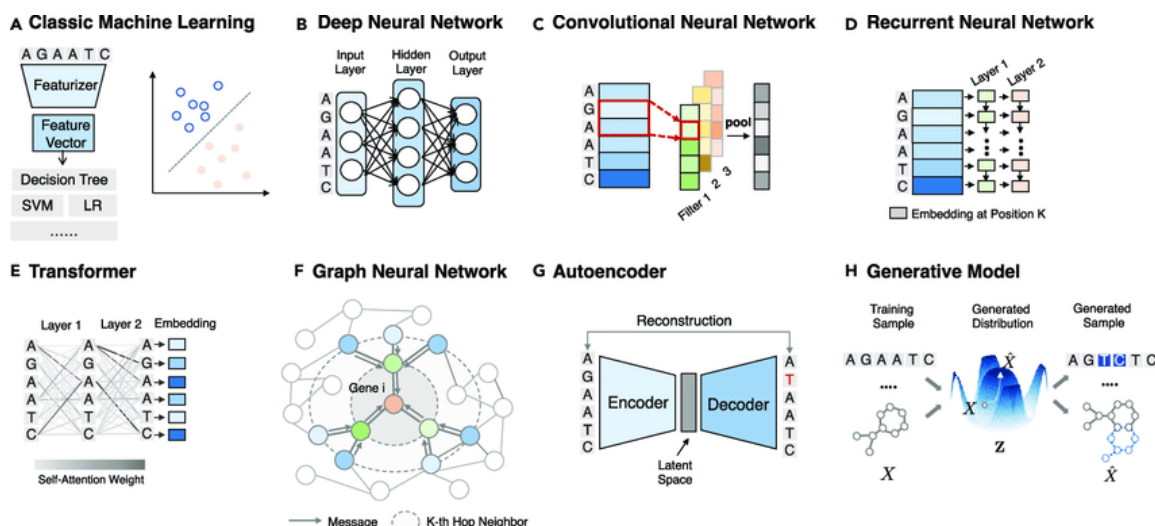


Рисунок 3.3 – Різні моделі машинного навчання

Основою машинного навчання є дані. Для тренування моделей необхідно зібрати релевантний і якісний набір даних, який відповідає задачі. Дані можуть бути різного типу:

- числові (наприклад, вікові значення),
- текстові (наприклад, коментарі в соціальних мережах),
- зображення,
- відео,
- часові ряди тощо.

Дані можуть бути як структурованими (наприклад, таблиці), так і неструктурованими (наприклад, текст або зображення). Для якісного навчання важливо, щоб зібрані дані були репрезентативними та охоплювали всі можливі сценарії використання моделі. Сирі дані часто містять помилки, відсутні значення, дублікатні записи або нерелевантну інформацію. Щоб модель могла працювати ефективно, дані проходять кілька етапів попередньої обробки. Спочатку очищення даних, видалення помилок, заповнення пропущених значень, видалення дублікатів. Після цього йде етап перетворення даних, наприклад, масштабування числових значень, кодування категоріальних змінних або нормалізація. Після цих етапів дані можуть бути використані для

навчання обраних моделей. Дані поділяються на навчальну вибірку (для тренування моделі), валідаційну вибірку (для налаштування параметрів) та тестову вибірку (для перевірки точності моделі).

Навчання — це процес, під час якого алгоритм аналізує навчальну вибірку і налаштовує свої параметри для мінімізації помилки. Наприклад, у лінійній регресії модель шукає таку лінію, яка найкраще описує залежність між змінними. Навчання відбувається за допомогою ітераційних методів. Алгоритм обчислює помилку (наприклад, різницю між передбаченими та реальними значеннями) і коригує свої параметри, щоб зменшити цю помилку. Цей процес повторюється до досягнення бажаного рівня точності.

Існує кілька основних типів машинного навчання:

- Кероване навчання (Supervised Learning): Модель навчається на даних з відомими мітками. Наприклад, у задачі класифікації модель аналізує дані та визначає, до якої категорії належить новий об'єкт. Приклади алгоритмів: лінійна регресія, дерева рішень, SVM, нейронні мережі.
- Некероване навчання (Unsupervised Learning): Модель працює з даними без міток і намагається знайти закономірності, наприклад, групувати дані в кластери. Приклади алгоритмів: k-середні, ієрархічна кластеризація, автоенкодери.
- Навчання з підкріпленням (Reinforcement Learning): Модель навчається взаємодіяти з середовищем і отримувати винагороду за правильні дії. Цей підхід часто використовується в робототехніці та іграх.

Лінійна регресія є однією з базових моделей машинного навчання. Вона використовується для прогнозування значення y на основі змінної x .

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b \quad (3.12)$$

де:

- w_i — ваги або коефіцієнти,
- b — зміщення (bias),
- x_i — вхідні змінні (ознаки).

Мета полягає в мінімізації функції втрат, наприклад, середньоквадратичної помилки (MSE):

$$MSE = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2 \quad (3.13)$$

де:

- \hat{y}_i — передбачене значення,
- y_i — реальне значення,
- N — кількість зразків даних.

Оптимізація і навчання в даній моделі виконується методом градієнтного спуску. Це метод оптимізації, який використовується для знаходження мінімуму функції. Кожна ітерація оновлює параметри моделі w у напрямку протилежному до градієнта функції втрат:

$$w := w - \eta \frac{\partial L}{\partial w} \quad (3.14)$$

- η — швидкість навчання,
- $\frac{\partial L}{\partial w}$ — часткова похідна функції втрат L за параметром w .

Говорячи про некероване навчання можна розглянути основний метод К-середніх (K-means) що використовується як основний для такого типу мереж машинного навчання. К-середні — це алгоритм кластеризації, який групує точки даних у k кластерів. Основна ідея полягає у мінімізації відстані від кожної точки до центроїда її кластера.

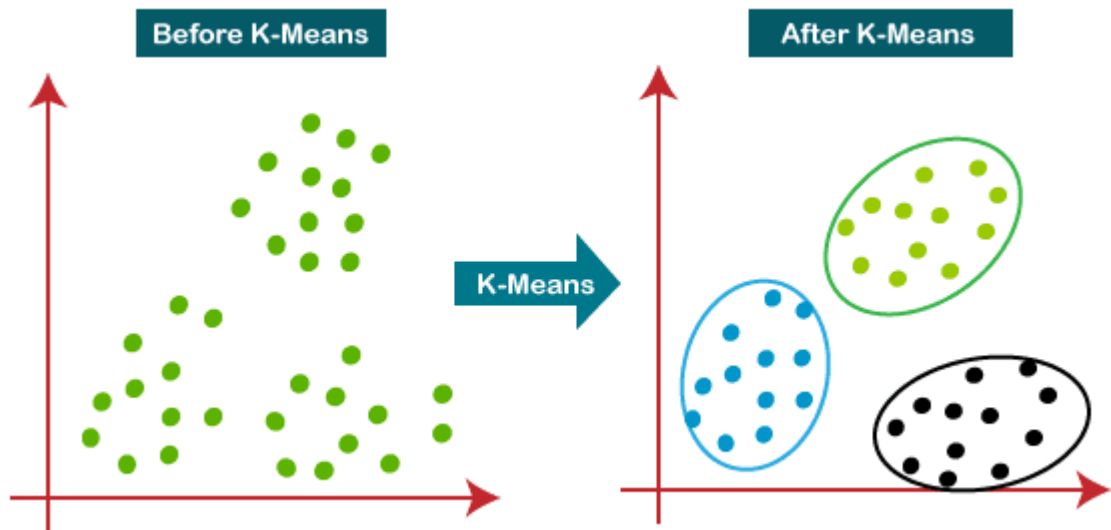


Рисунок 3.4 – Алгоритм роботи K-Means

Формула даного алгоритму має вигляд:

$$\arg \min_C \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2 \quad (3.15)$$

- C_i — набір точок у кластері i ,
- μ_i — центр кластера i ,
- $\|x - \mu_i\|^2$ — квадратична відстань між точкою x і центром кластера μ_i .

Цей алгоритм ітеративно призначає кожну точку до найближчого центроїда а після цього оновлює центроїди як середнє точок у кожному кластері.

У навчанні ж з підкріпленням агент навчається через взаємодію із середовищем. Агент отримує винагороду R_t за дії a_t , які він виконує в стані s_t . Мета агента, що навчається при цьому — максимізувати накопичену винагороду. При цьому розробник має оптимально розділити нагороди як за основну так і за допоміжні цілі, щоб контролювати навчання моделі.

$$G_t = R_t + \gamma R_t + 1 + \gamma^2 R_{t+2} + \dots \quad (3.16)$$

де γ — коефіцієнт дисконтування, який зменшує важливість майбутніх винагород. Політика $\pi(s,a)$ визначає, яку дію a агент виконає в стані s . Для її оптимізації використовуються методи, наприклад, Q-навчання:

$$Q(s, a) := Q(s, a) + \alpha \left[R_t + \gamma \max_a Q(s', a') - Q(s, a) \right] \quad (3.17)$$

де:

- $Q(s,a)$ — функція оцінки стану і дії,
- α — швидкість навчання.

Після навчання модель перевіряється на тестових даних, які вона ще не бачила. Це дозволяє оцінити її узагальнюючу здатність, тобто те, наскільки добре вона працюватиме з новими, невідомими даними.

Основні метрики для оцінки залежать від задачі:

- Для класифікації: точність (accuracy), повнота (recall), F1-міра.
- Для регресії: середньоквадратична помилка (MSE), середня абсолютна помилка (MAE).

Першою моделлю яку ми використаємо для створення моделі визначення загрози RowHammer буде модель з довго-короткочасною пам'яттю. Long Short-Term Memory (LSTM). Такий тип мереж є спеціальним типом рекурентної нейронної мережі (RNN), розробленим для роботи з послідовними даними. Її ключова особливість — здатність зберігати та обробляти важливу інформацію протягом тривалих часових проміжків, долаючи проблему затухаючих градієнтів, яка характерна для стандартних RNN.

Основний принцип роботи LSTM полягає в тому, щоб вирішувати, яку інформацію зберігати, яку забувати, а яку оновлювати на кожному кроці обробки послідовності. Для цього модель використовує комірки пам'яті, які функціонують як спеціалізовані накопичувачі даних. Доступ до цих комірок

контролюється через набір гейтів (воріт): ворота забування, ворота введення та ворота виходу. Ворота забування вирішують, яку частину попередньої інформації слід видалити, а яку зберегти, залежно від поточного контексту. Ворота введення визначають, яку нову інформацію додати до комірки пам'яті, тоді як ворота виходу контролюють, яка інформація буде передана далі на наступний крок. Такий підхід дозволяє LSTM ефективно працювати з задачами, де важливо враховувати контекст на великій відстані в послідовності. Наприклад, у задачах обробки природної мови це може бути врахування взаємозв'язків між словами, які знаходяться на початку та в кінці речення. Подібні довгострокові залежності є важливими також у задачах аналізу часових рядів, таких як прогнозування ринкових тенденцій або розпізнавання патернів у потоках даних.

LSTM моделі мають значну стійкість до втрати важливої інформації завдяки своїй архітектурі. Водночас ця ж архітектура ускладнює їхню реалізацію, вимагаючи більших обчислювальних ресурсів порівняно з простими RNN. Незважаючи на це, LSTM залишається популярною моделлю для роботи з послідовними даними, оскільки її переваги в точності й здатності зберігати довгострокові залежності переважають недоліки.

Кожен блок LSTM (представлений на рисунку 3.5) складається з трьох основних компонентів:

1. Вхід (input gate): визначає, яку нову інформацію додати до "пам'яті".
2. Вихід (output gate): контролює, яка частина інформації з пам'яті стане виходом.
3. Вхід забуття (forget gate): вирішує, яку інформацію видалити з пам'яті.

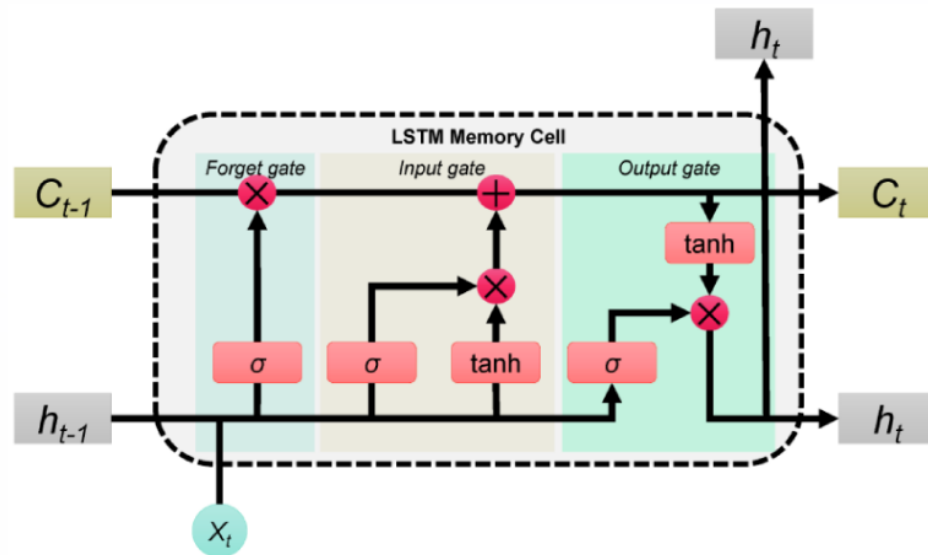


Рисунок 3.5 – Будова блоку пам'яті моделі LSTM

На кожному кроці t LSTM отримує вхід:

- x_t — вектор вхідних даних у момент часу t ,
- h_{t-1} — вектор виходу попереднього стану,
- c_{t-1} — стан пам'яті з попереднього кроку.

Вхід забуття працює першим і обробляє дані на предмет їх запам'ятовування:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (3.18)$$

- f_t — коефіцієнт, що визначає, яку частину старого стану пам'яті c_{t-1} слід забути,
- σ — сигмоїдна функція активації,
- W_f і b_f — ваги і зміщення.

Після цього система переходить на вхід захоплюючи з собою дані з минулих ітерацій:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (3.19)$$

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (3.20)$$

- i_t — коефіцієнт, який визначає, скільки нової інформації додати до стану пам'яті,
- \tilde{c}_t — кандидат на новий стан пам'яті.

Після цього стан пам'яті оновлюється з урахуванням теперішнього і минулого стану:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \quad (3.21)$$

- c_t — оновлений стан пам'яті, який є сумішшю старого стану c_{t-1} і нового внеску \tilde{c}_t .

І передаємо знайдену інформацію на вихід нейронної мережі:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (3.22)$$

$$h_t = o_t \cdot \tanh(c_t) \quad (3.23)$$

- o_t — коефіцієнт, який визначає, яку частину оновленого стану c_t слід винести на вихід,
- h_t — вихідний стан LSTM на поточному кроці.

LSTM навчається за допомогою зворотного поширення похибки через час (Backpropagation Through Time, BPTT). Функція втрат (наприклад, MSE або крос-ентропія) мінімізується з використанням градієнтного спуску.

Другою моделлю для імплементації буде багаторівневий перцептрон. Multi-Layer Perceptron (MLP) — це один із основних типів нейронних мереж, що складається з кількох шарів нейронів, які взаємодіють між собою через зважені з'єднання. MLP належить до категорії глибоких нейронних мереж і широко застосовується для вирішення різноманітних задач, таких як класифікація, регресія та інші проблеми машинного навчання.

Основна структура MLP включає три типи шарів: вхідний шар, один або кілька прихованих шарів, та вихідний шар. Вхідний шар отримує дані, які потім передаються через нейрони прихованих шарів, де вони обробляються і перетворюються в нові представлення. Вихідний шар генерує результат обробки, який може бути ймовірністю для класифікації або прогнозом для задач регресії. Процес навчання MLP зазвичай здійснюється за допомогою алгоритму зворотного поширення помилки (backpropagation). Під час навчання мережа намагається мінімізувати функцію втрат, що вимірює різницю між передбаченням моделі і фактичними значеннями. Для цього модель коригує ваги з'єднань між нейронами, що дозволяє мережі покращити свої прогнози на основі вхідних даних.

Приховані шари в MLP використовують нелінійні активаційні функції, такі як ReLU (Rectified Linear Unit), сигмоїда або тангенс гіперболи (tanh). Це дозволяє мережі моделювати складні й нелінійні залежності у даних, що робить її дуже потужною для вирішення складних задач. MLP можуть мати один або кілька прихованих шарів, що робить їх глибокими нейронними мережами (deep neural networks). Однак MLP мають деякі обмеження. Зокрема, вони не так добре підходять для роботи з даними, що мають просторову або часову структуру, як зображення або текст. У таких випадках більш ефективними можуть бути спеціалізовані архітектури, як-от згорткові нейронні мережі (CNN) для зображень або рекурентні нейронні мережі (RNN) для послідовностей. Незважаючи на це, MLP все ще є потужним і універсальним інструментом для багатьох завдань машинного навчання, де немає таких специфічних вимог до структури даних.

Для кожного нейрона в мережі, лінійна комбінація вхідних значень можна виразити через формулу 3.12. Після того як обчислено лінійне значення нейрона, воно пропускається через активаційну функцію, щоб отримати вихід

a_j . Активаційна функція може бути різною залежно від задачі. Однією з найбільш популярних є сигмоїда, яка виражається формулою:

$$a_j = \sigma(z_j) = \frac{1}{1+e^{-z_j}} \quad (3.24)$$

Інші активаційні функції, що широко використовуються в MLP, включають ReLU (Rectified Linear Unit), яка має вигляд:

$$a_j = \max(0, z_j) \quad (3.25)$$

або тангенс гіперболічний (tanh):

$$a_j = \tanh(z_j) = \frac{e^{z_j} - e^{-z_j}}{e^{z_j} + e^{-z_j}} \quad (3.26)$$

Основний алгоритм навчання MLP — це зворотне поширення помилки, що дає змогу коригувати ваги нейронної мережі для мінімізації помилки на виході. При навчанні моделі використовується функція втрат (loss function), яка для задачі регресії може бути визначена як середньоквадратична помилка за формулою 3.13. Також оскільки MLP використовує зворотне поширення помилки, то необхідно обчислювати похідні функції втрат по відношенню до ваг і зсувів. Вони використовуються для оновлення параметрів формулу 3.14. Цей процес повторюється для кожної ітерації навчання, що дозволяє моделі поступово покращувати свої прогнози.

Таким чином, в MLP кожен нейрон виконує лінійне перетворення вхідних даних, а потім застосовує активаційну функцію для отримання результату. Зворотне поширення дозволяє коригувати ваги для мінімізації помилки. Математична складова MLP складається з лінійних трансформацій, активаційних функцій та використання градієнтного спуску для оптимізації мережі. При цьому такий тип мереж є досить простим, що збільшує швидкість роботи, що є дуже важливим в низькорівневих імплементаціях.

Останнім екземпляром який ми використаємо для навчання будуть згорткові нейронні мережі. Convolutional Neural Networks (CNNs) є одним з найбільш популярних типів нейронних мереж, що спеціалізуються на обробці та аналізі структурованих даних, таких як зображення, відео чи аудіо. Вони зазвичай використовуються для задач комп'ютерного зору, включаючи класифікацію зображень, розпізнавання об'єктів, сегментацію, а також інші задачі, пов'язані з аналізом візуальних даних. Основною особливістю CNN є застосування операцій згортки (convolution), що дозволяє ефективно видобувати ознаки з даних. Операція згортки полягає в застосуванні фільтрів (ядра) до вхідних даних або попередньому шару. Кожне ядро проходить по даним, виділяючи локальні ознаки, такі як краї в зображеннях, текстури або інші специфічні патерни даних. Операція згортки виконується таким чином:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n) \quad (3.27)$$

- $S(i, j)$ — результат згортки на позиції (i, j) ,
- $I(m, n)$ — дані вхідного кластера,
- $K(i, j)$ — ядро згортки,
- $*$ — операція згортки (добуток і сумування).

В результаті кожен фільтр створює новий блупринт (активацію), що містить ознаки, виявлені цим фільтром. Після згортки результат передається через активаційну функцію, щоб додати нелінійність в мережу. Зазвичай використовується функція ReLU (Rectified Linear Unit), представлена формулою 3.25. Це дозволяє моделі навчатися складним, нелінійним патернам, які є необхідними для вирішення більшості реальних задач.

Для зменшення розмірів даних і зменшення обчислювальної складності застосовується підвибірка (pooling). Один з найпоширеніших методів — це максимальна підвибірка (max pooling), де з кожного підблоку вибирається

максимальне значення $\text{MaxPool}(x) = \max(x)$. Підвибірка зменшує просторові розміри активацій, що допомагає знизити кількість параметрів і запобігає перенаванчання моделі.

Після кількох шарів згортки та підвибірки, отримані ознаки передаються в один або кілька шарів з повним з'єднанням (fully connected layer), де кожен нейрон зв'язаний з усіма нейронами попереднього шару. Це дозволяє здійснити фінальне класифікаційне рішення на основі ознак, витягнутих на попередніх етапах. Вихідний шар зазвичай є шаром, що відповідає за класифікацію або регресію. Для задачі класифікації зазвичай використовується сигмоїд (формула 3.24) або софтмакс для перетворення результатів у ймовірності.

$$\text{Softmax}(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} \quad (3.28)$$

де z_j — вихід нейрону j на вихідному шарі, k — усі можливі класи передбачені для класифікації даною моделлю.

До особливостей CNN можна віднести автоматичне видобування ознак - CNN можуть автоматично видобувати ознаки з вхідних даних без необхідності ручного вибору, що робить їх дуже потужними для обробки комплексних наборів даних. Це дозволяє нейронній мережі самостійно навчатися і адаптуватися до специфічних особливостей візуальних патернів. Фільтри в шарах згортки здатні виявляти локальні ознаки (наприклад, краї, текстури), і саме ці ознаки використовуються для більш високорівневого аналізу. Вони також зберігають просторову інформацію, що важливо для розпізнавання об'єктів у зображеннях. Завдяки тому, що фільтри можуть переміщатися по зображенню, CNN можуть захоплювати локальні патерни на різних позиціях у зображенні, що робить мережу інваріантною до зміщення об'єкта.

Шар підвибірки і застосування фільтрів допомагають зменшити розміри даних і кількість параметрів, необхідних для навчання моделі, що знижує ризик

перенавчання і покращує загальну продуктивність. Також при цьому мережа може масштабуватись для роботи з великими обсягами даних, зокрема в реальному часі. Оскільки CNN самостійно видобуває важливі ознаки, це значно спрощує процес створення моделі.

До недоліків CNN можна віднести великі обчислювальні витрати. Навчання CNN вимагає значних обчислювальних ресурсів, особливо для великих мереж і великих наборів даних. Для досягнення високої точності CNN також потребують великих обсягів навчальних даних. Тому важливо мати великі датасети для навчання і тестування. Але не дуже великими бо дана модель може бути схильна до перенавчання. Якщо даних недостатньо, або якщо мережа занадто велика для наявного набору даних, CNN може перенавчатися, що знижує її здатність до узагальнення.

3.3.1 Використання для захисту та порівняння результатів мереж

Створення механізму захисту буде ділитися на чотири етапи:

1. *Вибір особливостей*: вибір списку апаратних подій для відстеження нашою обраною моделлю;
2. *Симуляція*: створення вибірки даних шляхом моделювання та моніторингу системи при атаці, в якій буде реалізовано цей механізм;
3. *Навчання та тестування*: згенеровані дані поділяються на набір для навчання та набір для тестування та використовуються для навчання моделі ML та її оцінки.
4. *Апаратна інтеграція*: впровадження алгоритму виявлення в систему.
5. *Широке тестування*: тестування системи на даних датасету атак на пам'ять різних виробників сформованого в розділі 2

Тож для коректної імплементації системи машинного навчання ми маємо визначити головні параметри для відслідковування. Коли виконується атака RowHammer, вона має певний вплив на мікроархітектурні події. Список подій, які відстежуються для виявлення, повинен дозволяти механізму розрізняти атаки та безпечну поведінку системи. Хоча різноманітність відстежуваних подій важлива для належного виявлення атак і зниження частоти хибних спрацьовувань, вибір надлишкових подій може зробити реалізацію дорожчою з точки зору пам'яті, споживання енергії та/або часу виявлення без суттєвих переваг для точності виявлення.

Мета атаки RowHammer — змінити значення бітів в основній пам'яті шляхом швидкої та багаторазової активації рядків DRAM. Таким чином, атака спричинить багато конфліктів для рядків у банках DRAM і дуже мало звернень до рядків напряду (row hits), це і буде першими подіями, які механізм буде відстежувати. Щоб отримати доступ до основної пам'яті, атака повинна обійти всі рівні кешу, очищаючи при цьому рядки агресора. Кеш першого рівня (L1) розділений на L1-I для інструкцій та L1-D для даних. Сигнатурний патерн атаки RowHammer є дуже коротким за кількістю операцій. Коли його інструкції зберігаються в L1-I, він навряд чи генеруватиме багато промахів у цьому кеші так як буде часто звертатись по тій самій адресі. Отже, L1-I матиме більше потраплянь по кешу, ніж промахів, а L1-D матиме більше промахів кешу так як даних така атака не несе. Тому потрібно відстежувати потрапляння по кешу L1-I та промахи кешу L1-D. Активність інших рівнів кешу може бути частково врахована з промахів кешу в L1 і активності основної пам'яті. Тому відстеження звернень до кешу та промахів кешу на інших рівнях є зайвим. Тож резюмуючи можна сказати, що були вибрані такі характеристики: влучання L1-I, промахи L1-I, влучення L1-D, промахи L1-D, влучення в рядки і промахи в рядки.

Використаємо gem5 [91] для симуляції атаки та налаштуємо архітектуру, на якій працюватиме механізм. Симулятор має бути налаштований для

реєстрації вибраних функцій під час моделювання програм. У той час як події, пов'язані з кеш-пам'яттю, можуть реєструватися gem5 безпосередньо, події, пов'язані з буфером рядків, не реєструються Ramulator, який використовується для моделювання основної пам'яті. Модуль Memory-Corruption, представлений у додаткових функціях можна використовувати для реєстрації звернень рядків і конфліктів рядків. Також паралельно будуть працювати різні програми для запуску, включно з програмами атаки та тестами продуктивності пам'яті, щоб підкреслити компоненти, на які також буде спрямована атака. Ці програми виконуються послідовно на одному змодельованому процесорі або паралельно з використанням кількох змодельованих процесорів.

Зібрані дані для навчання та тестування витягуються з вихідних файлів gem5 в які було додано кастомні дані з незалежних тестів на реальних чіпах DRAM. Зареєстровані події перетворюються на зразки фіксованої тривалості та групуються у вікна (буфери фіксованої довжини), позначені як атака або відсутність атаки. Два послідовних вікна частково перекриваються: друга половина в n -го вікна повторюється для першої половини $(n + 1)$ -го вікна атаки. Коефіцієнт перекриття між двома послідовними вікнами можна змінювати за бажанням. Потім випадковим чином вибираються вікна для набору даних для навчання або для набору даних для тестування. Кілька моделей ML навчаються та тестуються за допомогою схожих наборів даних, щоб оцінити і рівень відносну швидкість навчання. Щоб вибрати модель для кінцевої реалізації, потрібно враховувати точність, використання пам'яті та час висновку під час роботи в програмному забезпеченні. Також важливо розуміти наскільки буде реально зібрати навчальну вибірку для тої чи іншої мережі, так як велика кількість даних можуть просто бути недоступна через важкість її збору. Також це може бути не дуже комерційно вигідно, що теж є важливим фактором при оцінці нейронних мереж що будуть використовуватись у великих масштабах.

Апаратна інтеграція. Коли модель ML налаштована та здатна класифікувати вікна подій як такі, що походять від атаки чи ні, її можна реалізувати як механізм онлайн-виявлення в архітектурі системи. Інтегрований механізм показано на рисунку 3.6. Реалізований механізм можна розглядати як три основні складові:

1. апаратні лічильники
2. буфери семплів (проміжний 2a і фінальний 2b)
3. нейронна мережа

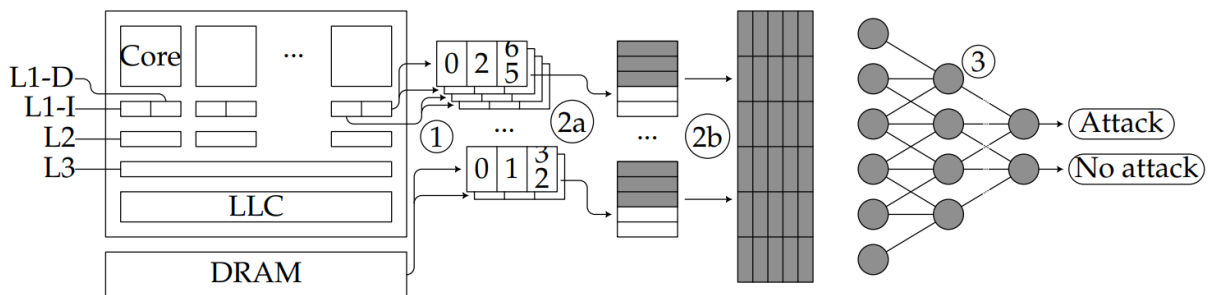


Рисунок 3.6 – Інтеграція механізму виявлення RowHammer в DDR на основі машинного навчання

Апаратні лічильники безпосередньо інтегровані в архітектуру та підраховують вибрані події. Кожне ядро має окремий набір лічильників для підрахунку подій попадань і промахів на L1-I та L1-D. Лічильники регулярно зчитуються, копіюються в буфери та очищаються. Після того, як буфери заповнені, вони копіюються у вхідний буфер нейронної мережі та частково очищаються відповідно до коефіцієнта перекриття між послідовними вікнами. Реалізована нейронна мережа обробляє свій вхідний буфер і класифікує процес як процес нормальної поведінки або ненормальної.

Для спрощення навчання та тестування моделі машинного навчання все буде виконуватись безпосередньо на одному портативному комп'ютері з

процесором Intel® Core™ i7-8565U @ 1,80 ГГц, 16 ГБ оперативної пам'яті під керуванням Windows 10 версії 19042. Симуляція системи за допомогою gem5 буде виконуватися на сервері. Кілька системних архітектур використовуються для створення наборів даних, щоб збільшити різноманітність вхідних параметрів та сценаріїв. Конфігурації включають один або два ЦП, що працюють на частоті 1 ГГц, з використанням включених класів ЦП TimingSimpleCPU, які враховують таймінги кожної інструкції, або DerivO3CPU, який, крім того, може виконувати позачергове виконання команд атаки. Система використовує два 32 КБ кешу L1 на ЦП (один L1-D і один L1-I), один глобальний кеш L2 512 КБ і, нарешті, DDR4 DRAM на 2400 МГц, який обробляє Ramulator [65] як основну пам'ять із обмеженням на зберігання 4 ГБ.

Щоб згенерувати набори даних, ми вибрали дві різні програми для запуску на змодельованих системах. Перша програма - це тест STEAM, який вимагає великої кількості пам'яті [101], через те що його якраз і зроблено для тестування продуктивності пам'яті. Друга програма — це комбінація атаки RowHammer із довільним доступом до пам'яті. Атака RowHammer буде представляти собою цикл викликів пам'яті з лістингу 1.1, а випадкові звернення до пам'яті записані в лістингу 3.1. У цьому коді функція rand() вибирає випадкову адресу в межах а 218- масиву байтів. Обидва цикли виконуються по черзі кілька разів із випадковою тривалістю (де максимальна тривалість у 5 разів перевищує мінімальну тривалість), причому кожен з них займає приблизно 50% загального часу виконання. І тест STEAM, і програма для атаки/довільного доступу є програмами, які потребують багато пам'яті, тому модель машинного навчання повинна буде розпізнавати шаблони атак, а не лише програми, які потребують багато пам'яті.

```
rand_loop:
    mov rand(), %eax ; put a random address into %eax
    mov (%eax), %ebx ; read value at address %eax
    jmp rand_loop    ; restart the loop
```

Лістинг 3.1 - Цикл довільного доступу до пам'яті на x86 архітектурі

Для файлу журналу моделювання ми генеруємо зразки по 100 нс кожен, які групуємо у 100 10-мікросекундних вікна із 50% перекриттям між послідовними вікнами доступу: останні 50 зразків записаних вікон доступу повторно використовуються як перші 50 зразків наступного вікна. Програмний лічильник на виході використовується для класифікації кожного вікна як таке, що виконує атаку і є загрозою, чи ні.

Як було описано вище випробовуються три різні моделі ML: Довга короткочасна пам'ять (LSTM), Багатошаровий перцептрон (MLP) і згортова нейронна мережа (CNN).

Моделі машинного навчання будуються за допомогою Keras [103], мовою Python, представленими в лістингу 3.2, 3.3 і 3.4. У цих списках `n_timesteps = 100` кількість зразків на вікно атаки, і `n_features = 6` це кількість різних функцій, зареєстрованих `gem5` і використаних як вхідні дані. Усі три моделі беруть `n_timesteps × n_features` зразків як буфер даних на вхід і має 2 виходи ("відбувається атака" і "атака відсутня").

```

1 model = Sequential()
2 model.add(LSTM(100, input_shape=(n_timesteps,n_features)))
3 model.add(Dropout(0.5))
4 model.add(Dense(100, activation='relu'))
5 model.add(Dense(n_outputs, activation='softmax'))
6 model.compile(loss='categorical_crossentropy',
7               optimizer='adam', metrics=['accuracy'])

```

Лістинг 3.2 - Код Python для створення моделі LSTM

```

1 model = Sequential()
2 model.add(Permute((2,1), input_shape=(n_timesteps, n_features)))
3 model.add(Dense(n_timesteps // 2))
4 model.add(Flatten())
5 model.add(Dense(128, activation="relu"))
6 model.add(Dense(n_outputs, activation="softmax"))
7 model.compile(loss='categorical_crossentropy',
8               optimizer='adam', metrics=['accuracy'])

```

Лістинг 3.3 - Код Python для створення моделі MLP

```

1 model = Sequential()
2 model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
3                 input_shape=(n_timesteps, n_features)))
4 model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
5 model.add(Dropout(0.5))
6 model.add(MaxPooling1D(pool_size=2))
7 model.add(Flatten())
8 model.add(Dense(100, activation='relu'))
9 model.add(Dense(n_outputs, activation='softmax'))
10 model.compile(loss='categorical_crossentropy',
11              optimizer='adam', metrics=['accuracy'])

```

Лістинг 3.4 - Код Python для створення моделі CNN

Навчання відбувається в двох режимах, що є ідентичними до роботи в реальних комп'ютерних системах. Перевірка також відбувається за допомогою наборів даних, створених за допомогою симуляції архітектури в умовах кількох видів навантаження пам'яті DDR:

- Ізольоване виконання: одночасно виконується лише одна програма;
- Паралельне виконання: дві програми виконуються паралельно на двох змодельованих ядрах.

Щоб розширити набори даних, обидві конфігурації запускалися як на центральних процесорах з послідовною архітектурою, так і на процесорах з позапорядковим виконанням команд. Це дозволяє оцінити ефективність моделей машинного навчання в різних середовищах, а також врахувати вплив архітектурних особливостей апаратного забезпечення на результати. Точність

різних моделей, отриманих під час цього аналізу, наведено в таблиці 3.3. Ці результати допомагають ідентифікувати найефективніші моделі для конкретних умов обчислень і розширених наборів даних.

Час обробки, зазначений у таблиці, відображає продуктивність програмного забезпечення моделей машинного навчання. Крім того, аналіз часу обробки дозволяє виявити потенційні вузькі місця, пов'язані з виконанням моделей. Наприклад, для конфігурацій із позапорядковим виконанням команд часто спостерігається менший час обробки, оскільки такі архітектури дозволяють оптимізувати виконання завдань завдяки більш ефективному управлінню потоками інструкцій. Цей фактор є важливим для врахування під час вибору платформи для впровадження моделі машинного навчання.

Таблиця 3.3 – Точність і час категоризації ML моделей

Тип моделі	Точність (%)	FP (%)	FN (%)	Час на виконання
LSMP (ізолюване)	99,9447	0,0527	0,0026	236 мкс
LSMP (паралельна)	99,8861	0,0902	0,0237	246 мкс
MLP (ізолювана)	99,9824	0,0167	0,0009	7,5 мкс
MLP (паралельна)	99,7675	0,2183	0,0142	6,9 мкс
CNN (ізолювана)	99,9851	0,0140	0,0009	40 мкс
CNN (паралельна)	99,9715	0,0285	0	53 мкс

Таким чином, отримані результати не лише оцінюють точність і швидкодню різних моделей, але й підкреслюють важливість адаптації моделей до конкретного апаратного забезпечення. Це особливо актуально для реальних систем, де затримка обробки та точність можуть впливати на продуктивність і кінцевий досвід користувачів. Це підтвердження концепції демонструє, що

машинне навчання можна використовувати для виявлення атак RowHammer із трасування апаратних подій з високою точністю. Тим не менш, це рішення вимагає важливої додаткової накладної області пам'яті для зберігання вхідного буфера та моделі машинного навчання, і не гарантує виявлення всіх 100% атак. Щоб зробити його конкурентоспроможним у порівнянні з найсучаснішими алгоритмічними механізмами виявлення RowHammer, моделі машинного навчання та трасування необхідно додатково оптимізувати, щоб зменшити необхідну додаткову пам'ять на зберігання нейромережі для збереження напрямку на контролері пам'яті.

Цікаво, що в майбутньому рішення на основі машинного навчання можуть стати більш привабливими, ніж рішення на основі лічильників. Дійсно, алгоритмічні контрзаходи на основі лічильників використовують кількість лічильників, обернено пропорційну порогу збурення комірок пам'яті. Оскільки фізичне наближення між комірками стає все менше з роками [75], ця кількість лічильників буде продовжувати пропорційно зростати. А з іншого боку, контрзаходи на основі машинного навчання не будуть масштабуватися обернено пропорційно зниженню порогу активацій збурення RowHammer.

3.3.2 Порівняння зайнятої пам'яті

Так як дані мережі мають використовуватись в низькорівневих системах важливо розуміти їх фізичні розміри в еквіваленті пам'яті. Чим меншим буде розмір отриманої моделі, тим краще. Звісно також потрібно враховувати ціну її розширення, тобто чи система буде гарно розширюваною в умовах збільшення футпринту пам'яті. Обсяг зайнятої пам'яті визначається кількістю параметрів моделі та тим, як ці параметри зберігаються. Параметри нейромережі зберігаються у вигляді чисел із певною точністю, зазвичай 32-бітні (4 байти) або 16-бітні (2 байти) значення з плаваючою комою.

Загальний обсяг пам'яті, який використовує неймережа, можна обчислити за формулою:

$$\text{Зайнята пам'ять (MB)} = \frac{N_{\text{Parameters}} \times B_{\text{Parameter}}}{1024^2} \quad (3.29)$$

- $N_{\text{Parameters}}$ — загальна кількість параметрів у моделі, розрахована як сума параметрів кожного шару.
- $B_{\text{Parameter}}$ — кількість байтів, необхідна для зберігання одного параметра (4 байти для float-precision 32 або 2 байти для float-precision 16).

При цьому для розрахування кількості параметрів використаємо формулу:

$$N_{\text{Parameters}} = \text{depth}_n \times (\text{kernel}_{\text{width}} \times \text{kernel}_{\text{height}}) \times \text{depth}_{n-1} + \text{depth} \quad (3.30)$$

Тут:

- depth_n — кількість фільтрів у поточному шарі.
- depth_{n-1} — кількість фільтрів у попередньому шарі (або кількість каналів вхідних даних).
- $\text{kernel}_{\text{width}}$ та $\text{kernel}_{\text{height}}$ — розмір ядра згортки.
- Доданок depth враховує значення зсуву (bias) для кожного фільтра.

Розмір пам'яті значною мірою залежить від обраного формату чисел. Використання 32-бітних чисел із плаваючою комою (FP32) є стандартним у багатьох неймережах, але для зменшення обсягу пам'яті можна перейти на 16-бітні значення (FP16). Наприклад, якщо неймережа має 1 мільйон параметрів, пам'ять для FP32 буде:

$$\text{Memory}_{\text{FP32}} = 1,000,000 \cdot 410242 \approx 3.81 \text{ MB}$$

тоді як для FP16:

$$Memory_{FP16} = 1,000,000 \cdot 210242 \approx 1.91 \text{ MB}$$

Також окрім параметрів моделі, на зайняту пам'ять також впливають:

- Використання тимчасових буферів для зберігання проміжних результатів обчислень.
- Градієнти під час навчання, які також зберігаються у форматі FP32 або FP16.
- Пам'ять для входів і виходів, яка залежить від розміру вхідних даних та обраної архітектури.

Розрахуємо футпринт для першої мережі архітектури LSTM, яка працює з рекурентними обчисленнями. Для кількості параметрів тут використовують формулу:

$$N_{Parameters(LSTM)} = 4 \times (n_{timesteps} \times (n_{features} + n_{timesteps}) + n_{timesteps})$$

Таким чином отримаємо:

$$\begin{aligned} N1_{Parameters(LSTM)} &= 4 \times (100 \times (6 + 100) + 100) = \\ &= 4 \times (100 \times 106 + 100) = 4 \times 10700 = 42800 \end{aligned}$$

Додамо параметри повнозв'язних шарів:

$$N2_{Parameters(LSTM)} = 100 \times 100 + 100 + 100 \times 2 + 2 = 10202$$

Сумарно:

$$N_{Parameters(LSTM)} = N1 + N2 = 42800 + 10202 = 53002$$

Далі знайдемо футпринт для багатошарового перцептрона. У цій архітектурі відсутні згорткові шари. Параметри обчислюються тільки для повнозв'язних шарів:

Dense-шар 1:

$$N1_{Parameters(PCT)} = 100 \times (100 + 1) = 100 \times 101 = 10100$$

Dense-шар 2:

$$N2_{Parameters(PCT)} = 128 \times (100 + 1) = 128 \times 101 = 12928$$

Dense-шар 3:

$$N3_{Parameters(PCT)} = 128 \times (2 + 1) = 128 \times 3 = 384$$

Сумарно:

$$N_{Parameters(PCT)} = 10100 + 12928 + 384 = 23412$$

Для фінального обчислення CNN нейромережі спочатку потрібно обчислити розмір згорткових шарів. Застосуємо відповідну формулу для обчислення параметрів 3.30.

Conv1D, Шар 1:

$$\begin{aligned} N1_{Parameters(CNN)} &= 64 \times (3 \times 6) + 64 = 64 \times 18 + 64 = \\ &= 1152 + 64 = 1216 \end{aligned}$$

Conv1D, Шар 2:

$$\begin{aligned} N2_{Parameters(CNN)} &= 64 \times (3 \times 64) + 64 = 64 \times 192 + 64 = \\ &= 12288 + 64 = 12352 \end{aligned}$$

Dense Шар після Flatten шарів:

$$N3_{Parameters(CNN)} = 100 \times (flattened_{size} + 1)$$

Якщо вхідні дані $n_{timesteps}=100$ після MaxPooling (з $pool_size=2$) залишиться 50, тобто:

$$flattened_size = 50 \times 64 = 3200$$

$$N4_{Parameters(CNN)} = 100 \times (3200 + 1) = 100 \times 3201 = 320100$$

Вихідний шар:

$$N5_{Parameters(CNN)} = 100 \times (2 + 1) = 100 \times 3 = 300$$

Сумарно:

$$N_{Parameters(CNN)} = 1216 + 12352 + 320100 + 300 = 333968$$

Тож фінальне значення зайнятої пам'яті представленими неймережами можна побачити в таблиці 3.4. Як бачимо персептрон є найменшою мережею, при цьому тримаючи друге місце за точністю визначення атаки.

Таблиця 3.4 – Пам'ять зайнята ML моделями

Тип моделі	Кількість параметрів	Зайнята пам'ять (КБ)
LTSM	53002	207.38
Perceptron	23412	91.45
CNN	333968	1304.56

3.4 Модель захисту на основі частотного масиву

Допоки захисні механізми від RowHammer на основі лічильників надають найкращу гарантію захисту. Ці рішення використовують лічильники активації рядків для виявлення рядків-агресорів. На відміну від імовірнісних рішень, такі методи можуть запропонувати гарантований захист від змін бітів, коли агресор завжди буде виявлений до того, як атака пошкодить важливі дані.

Для врахування періодичних оновлень рядків DRAM лічильники в усіх методах захисту необхідно періодично скидати. Однак таке періодичне оновлення всіх рядків DRAM не відбувається одночасно, як це стається з їх лічильниками. Оновлення розподіляються короткими шматочками t_{RFC} кожен

t_{REFI} . Але оскільки контролер пам'яті не знає, який рядок зараз оновлюється, скидання лічильника не можна синхронізувати з власне оновленням рядка, доступ до якого він моніторить. У результаті більшість існуючих методів виявлення та захисту на основі лічильників скидають усі лічильники одночасно, кожен раз t_{REFW} . Це має певні прямі наслідки для розробки контрзаходів.

Оскільки більшість лічильників не скидається синхронно з оновленням рядка, агресор може використовувати цю інформацію для націлювання на рядки, чії лічильники будуть скинуті в середині атаки, поки рядок не оновлено, що призведе до атаки, яка не буде засвідчена контрзаходом. Рисунок 3.7 ілюструє атаку на рядок, який не оновлюється синхронно зі скиданням лічильників. На цьому малюнку оновлення в розглянутому рядку не синхронізовані зі скиданням лічильників. Таким чином, коли лічильники скидаються, фактична кількість АСТ рядка перевищує значення лічильника. Якщо під час атаки цей рядок використовується як агресор, коли фактичний підрахунок АСТ рядка перевищує A_{RH} , значення лічильника буде нижчим від фактичного підрахунку, оскільки його було скинуто під час атаки. У цьому випадку атака не буде виявлена. Існуючі механізми використовують кілька методів, щоб врахувати цей факт. Наприклад, BlockHammer [57] вирішує подвоїти весь механізм і по черзі скидати лічильники, а Graphene [60] ділить поріг виявлення на 2, отже, використовуючи вдвічі більше початкової кількості лічильників

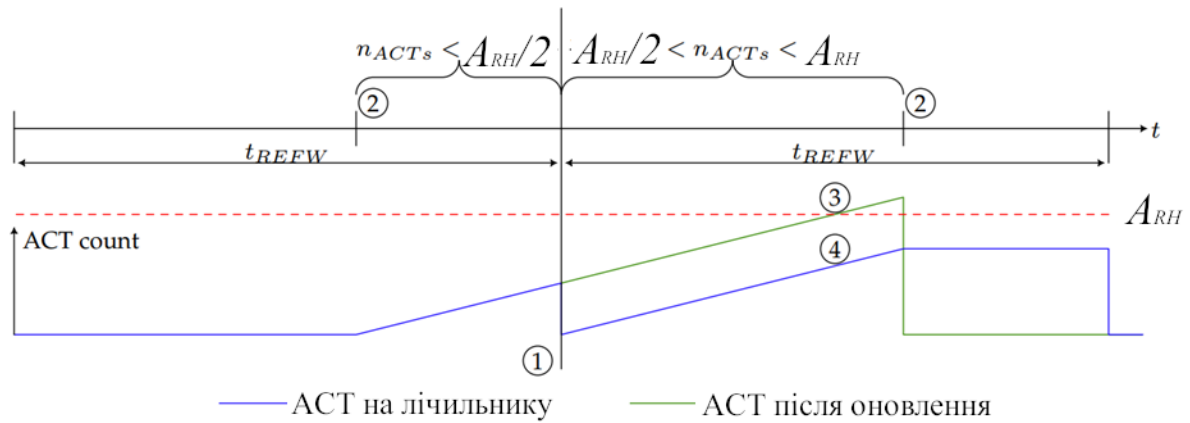


Рисунок 3.7 – Еволюція лічильника доступу при несинхронному оновленні

Запропонований механізм виявлення на основі частотного масиву базується на поєднанні лічильників АСТ та оцінки їх частоти. Щоб спростити демонстрацію, ми припускаємо, що для виконання 2 АСТ завжди знадобиться якийсь фіксований час, навіть якщо між ними видається періодична команда REF, тобто, ми будемо розглядати лише час, протягом якого на пам'ять можуть бути видані команди активації. Нехай в банку пам'яті можуть бути зроблені N_C команд активації протягом t_{REFW} . З урахуванням порогу збурення T_{RH} , оскільки два сусідні рядки жертви можуть бути використані для його пошкодження, необхідна кількість АСТ на рядок, щоб викликати зміну бітів, становить

$$A_{RH} = \frac{T_{RH}}{2} \quad (3.31)$$

Тоді кількість рядків використаних в атаці можна знайти як:

$$R_A = \frac{N_C}{A_{RH}} \quad (3.32)$$

Тому середній період між двома АСТ на агресора не може перевищувати $P = R_A \times t_{RC}$ для успішної атаки. Іншими словами, якщо середній період АСТ рядка перевищує P , він не є агресором, оскільки не може завершити атаку протягом вікна оновлення (t_{REFW}).

Щоб відстежувати лише потенційного агресора, ми можемо відстежувати лише рядки, які активуються досить часто, щоб бути агресорами. Відстежувані рядки будуть збережені в таблиці. Записи таблиці складаються з пари ключ-значення, де ключ — це ідентифікатор рядка, а значення — час закінчення терміну дії. Щоб відстежувати лише часто активовані рядки, ми можемо виконати такі дії:

1. Коли рядку видається АСТ вперше, виділяємо для нього запис у таблиці. Час видалення встановлюємо як $t + P$, де t це поточний час.
2. Якщо рядок знову активовано до видалення, збільшуємо час видалення на P . Таким чином в загальному вигляді його буде встановлено на $t_0 + n \times P$, де t_0 це час першого доступу до рядка в цьому t_{REFW} , а n це кількість разів, коли він був активований після кроку 1.
3. Після досягнення часу видалення запис можна прибирати з таблиці. Нові команди АСТ у цьому рядку розпочнуться знову на кроці 1.

Використовуючи цей метод, лише ті рядки, які активуються принаймні один раз за кожний P в середньому буде збережено в таблиці. Однак відстеження всіх часових позначок в таблиці для перевірки прострочених значень займе надто багато часу. Замість постійного моніторингу нові записи спочатку намагатимуться замінити прострочені.

Ця техніка може відстежувати рядки, які активуються досить часто, щоб стати частиною атаки. Еволюція одного запису в таблиці проілюстрована на рисунку 3.7. На цьому малюнку чорна лінія позначає час, що залишився до того, як таблиця забуде запис. АСТ позначаються червоними колами, а періоди, коли рядок не відстежується механізмом, позначаються сірими прямокутниками.

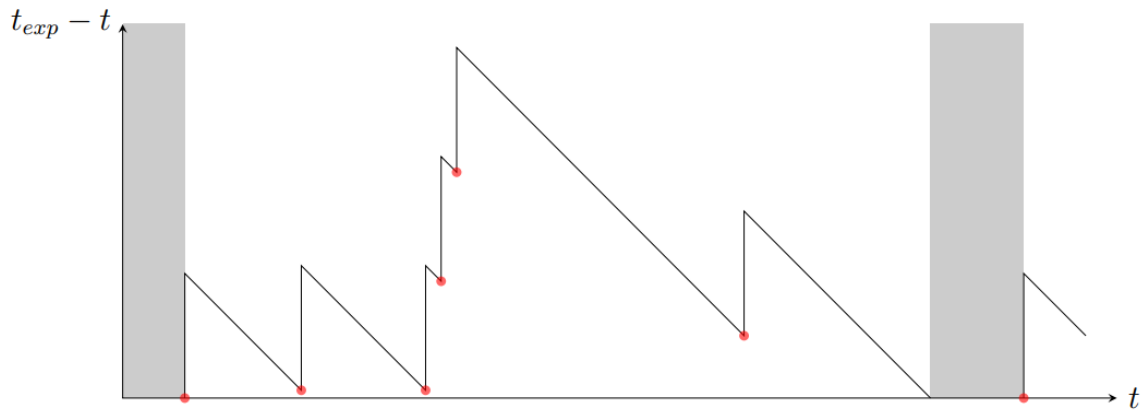


Рисунок 3.8 – Еволюція лічильника часу в таблиці доступу при атаці

Механізм здатний відстежувати всі ряди, які активуються досить часто, щоб бути потенційними агресорами. Окрім часу видалення, до запису таблиці додається лічильник, щоб вести підрахунок кількості активацій, які було видано для рядка з моменту його додавання до таблиці. Цей лічильник ініціалізується одиницею, коли запис виділено, і збільшується, коли рядку видається АСТ. Коли запис забувається, а потім використовується повторно, лічильник перезавантажується з 1. Прості атаки можна виявити лише за допомогою цього лічильника: коли лічильник наближається до порогу виявлення ($\approx A_{RH}$) ми можемо вважати ряд агресором і запобігти збуренню даних будь-яким способом, наприклад оновленням рядків чи обмеженням доступу до пам'яті атакуючим процесом.

Після того, як рядок вважається агресором, його запис можна очистити, щоб звільнити його. Це працює для атак RowHammer, які регулярно видають АСТ своїм рядкам-агресорам, зберігаючи запис у таблиці, збільшуючи лічильник для кожного АСТ, доки він не досягне порогу виявлення.

В цього простого підходу є дві проблеми які потрібно додатково вирішити. По-перше, агресор може заблокувати запис в масив (тобто зберігаючи його активним, тому недоступним для нових записів) протягом тривалого часу, видаючи велику кількість АСТ, не досягаючи при цьому порогу виявлення. Тоді

агресор може використати час, коли перший запис заблоковано, щоб заблокувати другий запис і так далі, що зрештою вимагає дуже великої кількості лічильників і моніторингових механізмів, щоб уникнути блокування всіх записів агресором.

Щоб вирішити цю проблему, ми можемо змінити те, як ми оцінюємо рядки як агресорів. Замість того, щоб покладатися лише на значення лічильника, ми можемо інтегрувати частоту активації у формулу враховуючи C значення лічильника, t_{exp} час видалення запису, t поточний час і P максимальна затримка між АСТ рядів-агресорів, потенціал ряду як агресора для атаки RowHammer f_{RH} розраховується за формулою

$$f_{RH} = C \times \frac{t_{exp} - t}{P} \quad (3.33)$$

Зміну цього значення відносно лічильника проілюстрована на рисунку 3.8. Поріг для f_{RH} при якому розглядаємо ряд як агресора $f_{RH} = A_{RH}$. Якщо рядку агресора видається АСТ через регулярний інтервал $\Delta t \in [2t_{RC}; P)$, значення f_{RH} можна обчислити після АСТ за формулою:

$$f_{RH} = c \frac{t_0 + cP - (t_0 + (c - 1)\Delta t)}{P} = c \frac{c(P - \Delta t) + \Delta t}{P} \quad (3.34)$$

де t_0 це час першої команди активації, виданої рядку. Якщо атака видає АСТ рядку якомога повільніше ($\Delta t \rightarrow P$), f_{RH} досягає A_{RH} для $c = c_{max} \approx A_{RH}$. І навпаки, якщо атака видає АСТ якнайшвидше ($\Delta t \rightarrow 0$), f_{RH} досягає порогу A_{RH} для $c = c_{min} \approx \sqrt{A_{RH}}$. Точне значення c_{min} можна обчислити як найменше ціле значення C що задовольняє нерівність $f_{RH} \geq A_{RH}$, з $\Delta t = \Delta t_{min} = 2 \times t_{RC}$.

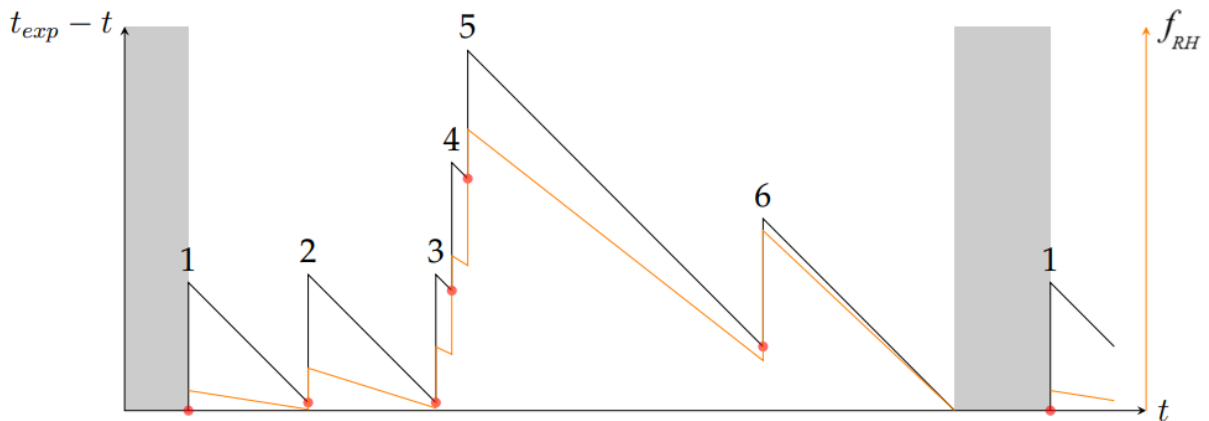


Рисунок 3.9 – Зміна значень лічильника часу та частоти в таблиці доступу при атаці RowHammer

Другою проблемою, яка може виникнути, є атаки при розділенні, як проілюстровані на рисунку 3.9. Як показано, агресор зазвичай виконує повільні активації, доки значення не наблизиться до ліміту, чекає, поки запис зникне з таблиці, а потім видає решту АСТ для атаки протягом часу, що залишився циклу оновлення t_{REFW} , щоб завершити атаку непомітно. Агресор повинен переконатися, що між зникненням запису з таблиці та завершенням циклу оновлення пройшло достатньо часу, щоб видати всі АСТ, що залишилися, щоб завершити атаку.

Щоб усунути цю проблему, P можна збільшити та скористатися перевагою способу обчислення f_{RH} . Чим більше АТС виконано в першій частині атаки, тим менше часу має агресор у другій частині. Враховуючи, що виконання ще одного запиту до пам'яті у першій частині займе час, співрозмірний кільком АСТ у другій частині, перша частина не повинна видати занадто багато АСТ, щоб дати другій частині достатньо часу для виконання решти АСТ. Щоб уникнути виявлення на другій частині атаки, вона повинна складатися з менше ніж c_{min} АСТ. Для захисту від атак розділення значення P має бути встановлено таким чином, щоб після завершення $A_{RH} - c_{min} + 1$ АСТ для першої частини та

очікування видалення запису часу, що залишився, було недостатньо для видачі решти $c_{min} - 1$ АСТ. Нове значення P має задовольняти нерівність

$$P (A_{RH} - c_{min} + 1) + \Delta t_{min} (c_{min} - 2) \geq t_{REFW} \quad (3.35)$$

Отже, мінімальне значення можна розрахувати за формулою

$$P = \frac{t_{REFW} - \Delta t_{min} (c_{min} - 2)}{A_{RH} - c_{min} + 1} \quad (3.36)$$

Також варто зазначити, що значення c_{min} є найменшим значенням лічильника, для якого обчислюється найменше значення, для якого $f_{RH} \geq A_{RH}$. Оскільки значення c_{min} і P залежать одне від одного, їх можна визначати ітераційно, спочатку обчислюючи P за допомогою

$$P_0 = N_{agg} \times t_{RC} = \frac{W \times t_{RC}}{A_{RH}} \quad (3.37)$$

а потім ітеративно обчислюючи c_{min} і P , поки значення c_{min} не стабілізується.

Висновки до розділу 3

Отже в даному розділі було змодельовано та проаналізовано три моделі для виявлення та захисту від атак типу RowHammer. Перша з них передбачала зменшення деталізації підрахунку на рівні рангу, що дозволило суттєво оптимізувати використання пам'яті, зменшивши її зайнятість лічильниками до 67% у DDR5. Це рішення дозволяє ефективніше використовувати ресурси контролера пам'яті, не знижуючи точність виявлення потенційних загроз. Крім того, така оптимізація сприяє зменшенню енергоспоживання та покращенню загальної продуктивності системи.

Друга модель базувалася на застосуванні методів машинного навчання для виявлення атак, зокрема використання LSTM, MLP та CNN моделей. Було визначено, що найкращі результати продемонструвала модель MLP, яка досягла точності 99,7% при часі виявлення 6,9 мкс. Це свідчить про високу ефективність використання штучного інтелекту для запобігання атакам на пам'ять в реальному часі. Використання MLP також дозволяє системі адаптуватися до нових загроз завдяки можливості перенавчання на актуальних наборах даних, що значно підвищує її надійність у довгостроковій перспективі. Недоліком такого підходу є потреба в зборі і навчанні нейромережі на великій кількості даних відповідного типу пам'яті.

Третя модель базувалася на методі виявлення на основі частотного масиву в Content Addressable Memory (CAM), що працював шляхом аналізу порогових значень кількості доступів до пам'яті, які можуть спричинити зміни бітів, а також контролю частоти цих доступів. Даний метод представляє вирішення проблеми присутньої в Graphene і BlockHammer, а саме несинхронне оновлення лічильників. Алгоритм передбачає запис в масив рядків до яких відбувся доступ та час коли дані з нього можна видаляти, базуючись на порогових значеннях для зміни бітів в представленому типі пам'яті. Крім того, такий підхід дозволяє значно покращити швидкість детектування атак, оскільки обробка відбувається без необхідності тривалих обчислень, що характерно для методів на основі штучного інтелекту. Недоліком ж такого підходу є необхідність у точних даних щодо атак на конкретний тип представленої пам'яті, тобто при виході нових версій пам'яті потрібен час на дослідження перед імплементацією.

Отримані результати підтверджують, що запропоновані моделі мають значний потенціал для підвищення ефективності захисту пам'яті від атак RowHammer, дозволяючи оптимізувати використання ресурсів та покращити швидкість і точність детектування атак. Кожна з них має свої унікальні

переваги, що дає змогу адаптувати їх для використання в різних умовах. Подальші дослідження можуть бути зосереджені на комбінуванні цих моделей, щоб досягти ще вищої ефективності, а також на інтеграції розглянутих підходів у реальні апаратні та програмні системи. Таким чином, представлена робота робить вагомий внесок у розробку нових методів захисту пам'яті від атак RowHammer та їхню практичну реалізацію в сучасних обчислювальних системах.

РОЗДІЛ 4.

ПРАКТИЧНЕ ЗАСТОСУВАННЯ РОЗРОБЛЕНИХ ЗАСОБІВ ЗАХИСТУ ОБЧИСЛЮВАЛЬНОЇ СИСТЕМИ ВІД АТАК НА ПАМ'ЯТЬ

4.1 Формування основних характеристик тестованої обчислювальної системи.

Розробляючи техніку запобігання RowHammer на реальних системах, важливо правильно оцінити її за багатьма пунктами. Це дозволить порівнювати її з існуючими пропозиціями, покращувати або виправляти недоліки, якщо необхідно, і налаштовувати деякі параметри для оптимізації. Щоб оцінити успішність захисту системи та вартість зменшення продуктивності, пропозиції щодо захисту мають використовувати платформи оцінки, які генерують такі показники, як частота виявлення хибно-позитивних (FP) і хибно-негативних (FN) результатів, а також деякі специфічні метрики, притаманні лише атакам типу RowHammer, як наприклад кількість додаткових оновлень пам'яті, кількість заміन рядків або кількість виявлених агресорів.

Навіть якщо пропозиції щодо захисту на основі програмного забезпечення не потребують жодних модифікацій апаратного забезпечення для захисту системи, генерація цих показників все одно вимагає розробки платформи оцінки, яка містить необхідні інструменти для їх моніторингу. Деяка інформація про пам'ять не відома процесору, і якщо атаці вдасться пошкодити один біт у пам'яті, немає гарантії, що зміна та спотворення бітів буде визначено під час генерації метрики оцінки. З іншого боку, апаратні пропозиції щодо захисту та зменшення наслідків завжди потребуватимуть реалізації в архітектурі, що підлягає оцінці вже накладних витрат на розробку та модифікацію я вже існуючих так і майбутніх модулів пам'яті.

Щоб належним чином оцінити пропозиції щодо захисту, платформа оцінки повинна мати можливість виконувати сучасні атаки, які вимагають повністю робочої операційної системи і програмами в ній (щоб тестувати наприклад атаки, які можуть виходити із пісочниці веб-браузера) під час виконання також потрібно передавати всю необхідну інформацію в середовищі, близькому до споживчих систем.

Розробка та виготовлення нових інтегральних схем для кожної ітерації під час розробки методів захисту від RowHammer матиме величезні фінансові та часові витрати. В тестуванні модулів пам'яті у великих масштабах ми покладались на FPGA [93], щоб емулювати контролер пам'яті, впроваджуючи атакуювальні алгоритми, щоб перевірити їх на існуючих сучасних системах та пам'яті DRAM. Також це має перевагу в тому, що такий етап дуже точно відповідає реальним системам, водночас пропонуючи високу модульність і низький цикл розробки для створення методів захисту. Однак як програмна реалізація в реальних системах, так і апаратна реалізація на FPGA стикаються з тими ж обмеженнями. По-перше, внутрішнє розташування банків DRAM невідоме контролеру пам'яті. Пропозиції щодо захисту, які використовують оновлення постраждалих рядків, не можуть захистити пам'ять, оскільки вони не можуть знати, які сусідні рядки потрібно оновити. По-друге, необхідність тестування на існуючих модулях пам'яті є проблемою при розгляді майбутніх технологій пам'яті, таких як DDR5, або нових енергонезалежних пам'яті, які потенційно чутливі до різновидів атаки RowHammer [62, 63].

Останнім рішенням для цього є використання симуляторів, де ми можемо симулювати сучасні або майбутні архітектури з актуальними чіпами пам'яті або з майбутніми технологіями. Сучасні пропозиції щодо захисту від атак напам'ять використовують різні інструменти моделювання для оцінки ефективності. Наприклад: Graphene [60], TWiCe [64], ProHIT [49] використовують симулятор мікроархітектури x86 McSimA+ [73]; BlockHammer

[57] і PARA [15] використовують симулятор синхронізації DRAM Ramulator [65]; MRLoc [50] використовує симулятор модульної архітектури gem5 [66].

Також крім симуляцій важливо зробити фізичну платформу, яка покаже результати в реальному світі. Тестова система базується на платформі розробки ARTY Z7 [103]. Ця плата містить систему на кристалів (SoC) Zynq-7000 від Xilinx, який містить двоядерний процесор 650 МГц ARM Cortex-A9 разом із програмованою вентиляною матрицею Xilinx 7-ї серії (FPGA). Поєднання процесора та FPGA дозволяє створювати спеціалізовані периферійні пристрої та спеціальні логічні схеми для оптимізації доступу до пам'яті. Архітектура системи з процесором і периферійними пристроями, інтегрованими в FPGA, проілюстрована на рисунку 4.1.

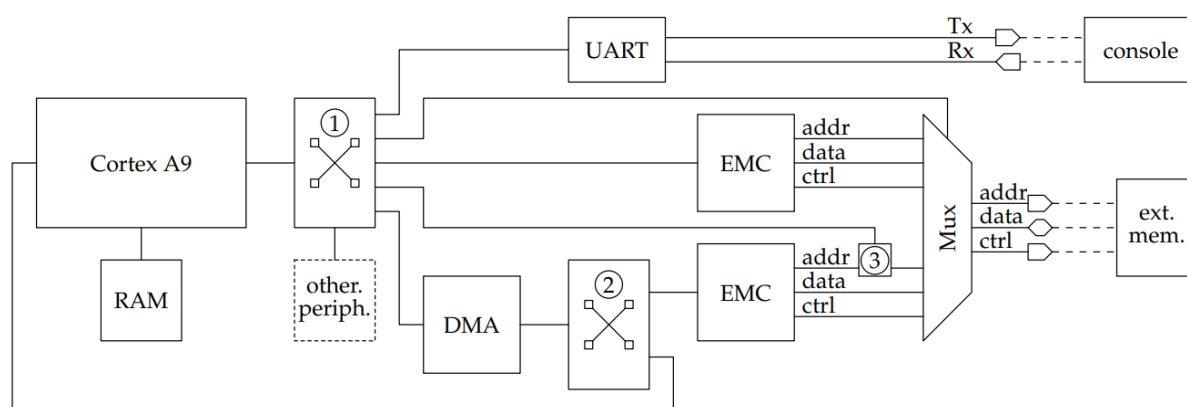


Рисунок 4.1 – Схема платформи тестування на основі FPGA

Процесор має спеціальний порт для зв'язку з оперативною пам'яттю та спілкується з периферійними пристроями через шину AXI при інтеграції більшої їх кількості. Для підключення одного головного пристрою до кількох підлеглих пристроїв ми використовуємо поперечну панель AXI.

Доступ до зовнішньої пам'яті від процесора вимагає використання периферійного контролера зовнішньої пам'яті (EMC) для керування контактами адреси, даних і керування зовнішньою пам'яттю. Цей EMC необхідно налаштувати під час проектування логічної схеми з таймінгами

пам'яті показаних в таблиці 1.1 для доступу до неї з максимальною швидкістю. Після експериментів ми помітили, що мінімальна затримка, якої може досягти ЕМС між двома послідовними зверненнями до зовнішньої пам'яті, становить 50 нс. Хоча це вище, ніж мінімальні 45 нс, зазначені в даташитах пам'яті, різниця досить мала, щоб вважати цю конфігурацію і її результати достовірними.

Доступ до пам'яті з боку процесора може сповільнюватися двома факторами. По-перше, процесор обмежений у кількості звернень до пам'яті, які він може виконати за допомогою однієї інструкції. Після завершення серії звернень до пам'яті для однієї інструкції процесор повинен отримати та декодувати наступну інструкцію з пам'яті. Ця операція займає деякий час, протягом якого процесор не звертається до перевіреної пам'яті. Крім того, програма використовуватиме інструкції, пов'язані з циклом (зменшення лічильника та перехід до початку циклу), щоб повторювати звернення до пам'яті велику кількість разів. Ці інструкції займуть деякий додатковий час для процесора що він використає для отримання та декодування, протягом якого процесор не матиме доступу до перевіреної пам'яті.

З цих причин платформа інтегрує периферійний пристрій прямого доступу до пам'яті (DMA) у FPGA, який виконуватиме велику кількість звернень до пам'яті без прямих команд від процесора. Процесор втручається лише один раз наприкінці кожного циклу, щоб перезапустити його, якщо це необхідно, обмежуючи вплив отримання і декодування інструкцій на процесі. DMA діє як підлеглий пристрій на шині AXI, підключений до процесора, і як контролер до шини, яка з'єднує його з пам'яттю, на якій він працює. Тому це має спеціальну паралельну AXI для зв'язку з оперативною пам'яттю, який з'єднує пристрій з виділеним ЕМС і Cortex для доступу до RAM.

Однак DMA обмежений у шаблонах адрес, які він може використовувати під час доступу до пам'яті. Простий DMA, інтегрований у FPGA, здатний лише

виконувати доступи для читання або запису на послідовних адресах пам'яті. Щоб оцінити чутливість до кількешарових атак на пам'ять, платформа повинна мати можливість виконувати повторні доступи до однієї адреси або кількох адрес. Щоб полегшити розробку платформи, замість того, щоб модифікувати DMA, ми реалізуємо матрицю, яка множить вектор адреси на виході щоб дозволити такій моделі доступу до пам'яті EMC після DMA, що генерує новий вектор адреси для атаки. Цей кінцевий вектор можна обчислити за формулою:

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}_{pins} = \begin{bmatrix} m_{0,0} & m_{0,1} & \dots & m_{0,n} \\ m_{1,0} & m_{1,1} & \dots & m_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n-1,0} & m_{n-1,1} & \dots & m_{n-1,n} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ 1 \end{bmatrix}_{EMC}. \quad (4.1)$$

Матриця складається лише з 1 і 0, а додатковий крок множення матриці замінюється порозрядною бінарною операцією АБО. Додаткова 1 додається в кінці вектора вхідної адреси для більшого контролю над модифікацією адреси. Ця матриця адрес реалізована як незалежний периферійний пристрій, редагований процесором.

Нарешті, підключення пам'яті лише до DMA унеможливить прямий доступ процесора до неї. Прямий доступ можна використовувати для виконання простих завдань, таких як пошук пошкоджень або ініціалізація пам'яті. Таким чином, периферійний пристрій мультиплексора вставляється для підключення EMC процесора та EMC DMA (за якою йде матриця адреси) до контактів DRAM. Це дозволяє процесору вільно перемикатися між доступами з DMA і доступами з процесора, а також налаштовувати два EMC з різними параметрами синхронізації. EMC DMA може бути налаштований для оптимізації швидкості атаки RowHammer та на рівень пошкодження записаних або зчитаних значень. Додатково, EMC, до якого звертається процесор, можна

налаштувати з більш повільними параметрами синхронізації, щоб забезпечити точність і неспотвореність значень, записаних або зчитаних у цю конфігурацію.

Для тестування DRAM на предмет захисту від RowHammer критично важливим є потужний і сучасний центральний процесор. Це обумовлено необхідністю проведення складних операцій, таких як швидке генерування високочастотних доступів до пам'яті, що є основою атаки RowHammer. Процесори з високою тактовою частотою (не менше 3.5 ГГц) і великою кількістю ядер (6–8 або більше) забезпечують можливість багатопоточності, що необхідно для паралельного виконання тестів. Наразі це є звичайною частиною сучасних комп'ютерних систем тому ми припускаємо це як базову точку піддослідної системи. Центральним процесором виступить Intel Core i7-13700K, який має 16 ядер (8 продуктивних і 8 енергоефективних), підтримує частоту до 5.4 ГГц і забезпечує потрібну для аналізу багатопоточну обробку даних. Материнська плата ASUS Pro WS W480-ACE забезпечує підтримку усіх стандартів пам'яті, відповідно і DDR5, можливість регулювання таймінгів і частоти DRAM через BIOS, а також має кілька слотів DIMM для тестування різних модулів пам'яті. Для тестування буде використовуватись кілька комплектів оперативної пам'яті:

- Kingston HyperX Fury DDR3 4 Гб з частотою 1600 МГц;
- Corsair Vengeance DDR3 8 Гб з частотою 1866 МГц;
- Crucial Ballistix DDR4 16 Гб з частотою 2666 МГц;
- Corsair Vengeance LPX 16 Гб (2 x 8 Гб) із частотою 3200 МГц;
- Kingston Fury Beast DDR5 16 Гб (2 x 8 Гб) із частотою 5200 МГц;

Так як основним елементом системи є модулі DRAM, які будуть піддаватися тестуванню для отримання репрезентативних результатів буде використано кілька модулів DRAM від різних виробників і поколінь (DDR3, DDR4 і DDR5). Додатково модулі мають різні ємності (4 Гб, 8 Гб, 16 Гб) і були виготовлені на різних техпроцесах, таких як 20 нм, 14 нм і 10 нм для Kingston

Fury Beast. Також важливо зазначити що система здатна підтримувати повний доступ до інформації SPD (Serial Presence Detect), що дозволяє зчитувати інформацію про модуль пам'яті.

Додатково не менш важливим компонентом є програмне забезпечення, яке дозволяє створювати специфічні сценарії для тестування DRAM на вразливість до RowHammer. Будуть застосовані описані вище gem3, спеціалізовані інструменти, такі як Memtest86 та деякі кастомні утиліти на основі Python і C, які генерують часті активації рядків пам'яті. Для аналізу результатів використаємо статистичні бібліотеки та платформи (Python із NumPy і Matplotlib).

4.2 Імплементация захисних моделей

Для імплементации скористаємось вбудованим модулем тестування кешу `cache_test_physaddr`. Цей підпрограмний каталог містить програму, яка вибирає набори ділянок пам'яті, які проектується на той самий кеш L3, встановлений на 2-ядерних ЦП типу Sandy Bridge. Він перевіряє, чи дійсно місцеположення зіставляються з тим самим набором кешу, визначаючи час доступу до нього і виводячи ці дані у файл CSV для відображення.

```
def test():
    cache = CacheBitPLRU(4, randomise=False)
    assert_eq(cache.mru_state(), '0000')

    def check(addr, is_miss, state):
        assert_eq(cache.lookup(addr), is_miss)
        assert_eq(cache.mru_state(), state)

    check(0, True, '1000')
    check(1, True, '1100')
    check(2, True, '1110')
    check(3, True, '0001')
```

Лістинг 4.1 – Тестування на кеш промахи

Отже, спершу ми налаштуємо систему для проведення тестів. Використовуємо Linux як базову операційну систему, оскільки вона дозволяє нам безпосередньо працювати з фізичною пам'яттю. Завантажуємо систему в режимі суперкористувача (root), адже тільки так можна отримати повний контроль над ресурсами пам'яті. Далі підключаємо потрібні інструменти — наприклад, mmap або спеціалізовані драйвери, які дають змогу мапувати фізичну пам'ять і отримувати до неї прямий доступ. Далі переходимо до отримання фізичних адрес пам'яті. Оскільки сучасні системи працюють із віртуальною пам'яттю, потрібно перетворити віртуальні адреси на фізичні. Це робиться через зчитування таблиць сторінок, які зберігаються в операційній системі. Ми обираємо адреси, які відповідають рядкам пам'яті, що знаходяться поруч один із одним, бо саме ці рядки будуть "агресорами" в атаці. Ретельно перевіряємо, щоб вони були коректно вирівняні, щоб створити оптимальні умови для RowHammer. Після запуску тестів отримуємо графік представлений на рисунку 4.2.

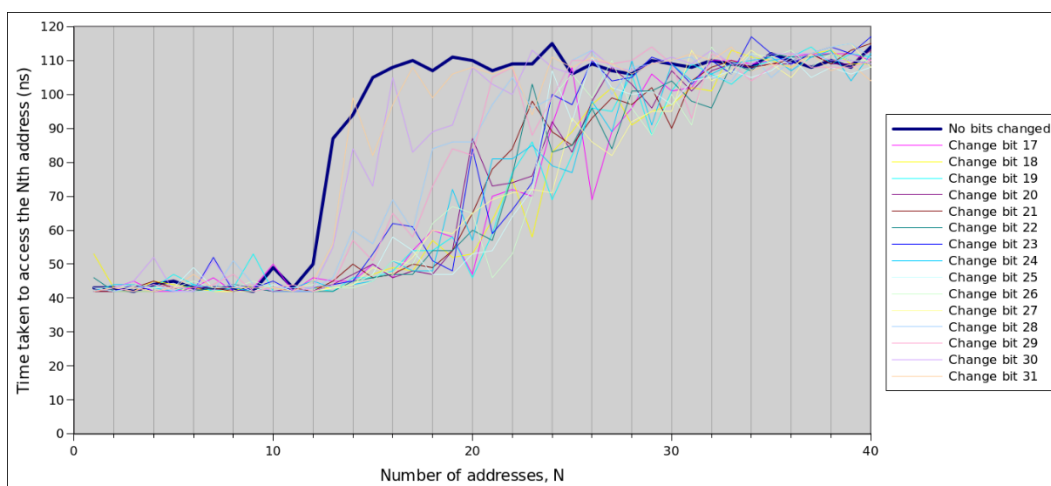


Рисунок 4.2 – Графік отримання кеш промахів на тестовій системі

Даний графік показує схильність до кеш промахів в кеші нашого процесора, що допоможе в подальшому з імплементацією алгоритму атаки. Після визначення адреси, створюємо шаблон доступу до пам'яті. Наша мета — багаторазово й безперервно звертатися до цих "агресорських" рядків. Це

робиться за допомогою циклів, які виконуються з мінімальними затримками. Для цього використаємо фрагмент коду представлений в лістингу 1.1, який максимально швидко читає й записує дані в комірки для атаки. Це дозволяє створити умови для витоків заряду в сусідніх рядках пам'яті жертви.

Щоб перевірити, чи викликала атака зміни в пам'яті, додаємо механізм моніторингу. Перед початком тесту ми записуємо відомі значення в рядок "жертви", а потім регулярно читаємо їх і порівнюємо зі збереженими. Якщо якісь біти змінили значення, це означає, що атака RowHammer успішно викликала помилки. Всі ці зміни ми фіксуємо в окремий журнал для подальшого аналізу. Зрештою, ми підсумовуємо результати. Записуємо частоту помилок, їхні позиції та інші деталі, щоб побачити, наскільки вразливий конкретний модуль DRAM. Потім ми можемо змінювати інтенсивність тесту або налаштування системи, щоб дослідити, як це впливає на рівень помилок. Такий підхід дає змогу не тільки оцінити вразливість, а й краще зрозуміти, як саме поводить себе пам'ять під час атаки.

Якщо ж говорити про імплементацію захисту на частотних масивах, то перш за все потрібно оцінити глибину дискретизації лічильників. Для реалізації в лічильниках час видалення i , отже, період повинні бути перетворені в дискретні значення. Дискретизувати змінні, пов'язані з часом P і Δt нам допоможе змінна часу t_{ATC} у тиках процесора на одну на АСТ. Наприклад, а t_{ATC} що рівне 0.25 означає, що лічильник налаштований таким чином, що пам'ять може отримати 4 команди активації за один i той же тік. Δt_{xe} в такому представленні буде дорівнювати $d_{min} \times t_{ATC}$, де d_{min} це максимальна кількість рядків, які можна використовувати одночасно без уповільнення швидкої атаки. При цьому значення t_{ATC} визначить точність таймера. При зменшенні t_{ATC} , точність таймера знижується, що призводить до меншого розміру запису в масиві (менше бітів, необхідних для збереження t_{exp}). Однак це також може призвести до збільшення кількості записів. Значення t_{ATC} можна оптимізувати

для вибору найкращого компромісу між розміром запису та кількістю записів, залежно від того, як таблиця буде реалізована.

Для хорошої деталізації підрахунку на рівні банку, $d_{min} = 2$: 2 ряди використовуються по черзі, щоб уникнути потрапляння в рядки. На рівні рангу можна використовувати декілька банків одночасно. Коли один банк обробляє АСТ, рядки в інших банках можуть отримувати АСТ паралельно, що призводить до більшого значення для d_{min} . як P має задовольняти нерівність 3.35, формула для обчислення першого значення та ітераційна формула стають наступними:

$$P_0 = \frac{W \times t_{ACT}}{A_{RH}}, \quad (4.2)$$

$$P = \left\lceil t_{ACT} \times \frac{W - d_{min} \times (c_{min} - 2)}{A_{RH} - c_{min} + 1} \right\rceil \quad (4.3)$$

При використанні дискретного значення для P виникає нова проблема. Оскільки таймер може не синхронізуватися з АСТ, нові записи можуть бути забуті до того, як вони фактично досягнуть початкового (недискретного) P часу. Щоб уникнути цієї проблеми, нові записи матимуть свої t_{exp} встановлені на $t + P + 1$ замість $t + P$ (де t це поточний тик), щоб переконатися, що всі записи залишаються в таблиці правильний проміжок часу. Як наслідок, якщо t_0 є тіком першої команди активації рядка, тоді $t_{exp} = t_0 + c \times P + 1$.

C_{min} при цьому обчислюється як найменше ціле число, яке задовольняє нерівність

$$c_{min} \times \frac{c_{min} \times (P - d_{min} \times t_{ACT}) + d_{min} \times t_{ACT} + 1}{P} \geq A_{RH}. \quad (4.4)$$

Механізм реалізує таблицю з двома адресованими вказівниками на пам'ять та одним масивом. Перший CAM CAM_R містить ідентифікатори рядків, другий CAM_T – час видалення, а масив CNT містить значення лічильників. Причина використання CAM для ідентифікаторів рядків і часу видалення

полягає в тому, що обидва вони потребують пошуку, коли видається АСТ. SAM_R використовується для пошуку, якщо в активованому рядку вже є запис, а SAM_T використовується для пошуку записів для заміни або видалення. Однак значення лічильника в CNT використовуються лише для розрахунку f_{RH} . Запис у механізмі виявлення має однаковий індекс для всіх масивів.

З точки зору алгоритму, у таблиці реалізовано такі функції:

- SAM_R реалізує функції пошук(*search(рядок)*), встановлення(*set(індекс, рядок)*).
- SAM_T реалізує функції пошуку, видалення і встановлення по часу *searchExpired(t)*, *searchAvailable(t)*, *get(індекс)*, *set(індекс, t)*.
- Додаткова функція видалення (*delete(індекс)*), який видаляє запис з усіх трьох компонентів таблиці.

Лістинг 4.1 показує, як працює механізм. У цьому алгоритмі t_{max} це максимальне значення, яке можна досягти t , $t_{cycle} = t_{max} / n_{циклів}$ – тривалість циклів обслуговування, тобто, кількість тактів між кожною перевіркою пам'яті для видалення прострочених записів. Функція *init* встановлює початкове значення t при запуску. Функція *onAST* викликається кожного разу, коли АСТ видається до вказаного рядка пам'яті, а функція *timer* викликається кожен такт для приросту t .

```

1  √ Function init:
2      t = 0
3  √ Function isRowhammer(c, texp):
4      f_RH = c * (texp-t) / P
5      return f_RH > T_RH
6
7  √ Function onACT(row):
8      index = CAMR.search(row)
9  √      if index >= 0 then
10         texp = CAMT.get(index)
11  √         c = CNT[index]
12  √         if texp < t then
13             texp = t + P + 1
14             c = 1
15  √         else
16             texp = texp + P
17             c = c + 1
18  √         if c >= cmin and isRowhammer(c, texp) then
19             detected(row)
20             delete(index)
21  √         else
22             CAMT.set(index, texp)
23             CNT[index] = c
24  √     else
25         index = CAMT.searchAvailable(t)
26         CAMR.set(index, row)
27         CAMT.set(index, t + P + 1)
28         CNT[index] = 1
29  √ Function timer:
30     t = t + 1 mod tmax
31  √     if t mod tcycle = 0 then
32  √         foreach index in CAMT.searchExpired(t) do
33             delete(index)

```

Лістинг 4.1 – Глобальний алгоритм роботи з масивами частоти

Існує максимум $1/t_{\text{ATC}}$ команд активації на кожному такті, що можуть збільшувати t_{exp} на P для рядів з доступом. У той же час значення $t_{\text{exp}} - t$ кожного запису зменшується на 1. Отже, на кожному тикі сума всіх $t_{\text{exp}} - t$ зменшується

на $n_{\text{записи}}$ і може збільшитися на $\frac{P}{t_{\text{ATC}}}$. Коли $n_{\text{записи}} < \frac{P}{t_{\text{ATC}}}$, сума всього $t_{\text{exp}} - t$ може збільшуватися швидше, ніж зменшуватися. І навпаки, коли $n_{\text{записи}} > \frac{P}{t_{\text{ATC}}}$, сума всіх $t_{\text{exp}} - t$ може тільки зменшуватися. Отже, для звичайної програми, що потребує великого обсягу пам'яті, яка використовує кеш, щоб уникнути повторних АСТ на кілька рядків, кількість записів стабілізуватиметься саме собою навколо $\frac{P}{t_{\text{ATC}}}$. Однак нападник може змусити механізм використовувати все більше і більше записів шляхом тимчасового блокування деяких рядків. Обчислення мінімальної кількості записів можна здійснити шляхом імітації атаки, яка намагатиметься використати якомога більше записів. Алгоритм імітації такої атаки написано в Лістингу 4.2

```

1  read t_ACT
2  read W
3  read d_min
4  read Hcf first
5  P = W*t_ACT / (A_RH)
6  repeat
7  c_min = findCmin(P, d_min)
8  P = t_ACT * (W-d_min*(c_min-2)) / A_RH-c_min+1 m
9  until c_min is fixed
10 n_entries = d_min
11 t_stop = (c_min - 1) * (P - d_min * t_ACT) + 1
12 while t_stop > P + 1 do
13     n_ACT = (t_stop-d_min*t_ACT -1) / P
14     t_stop = min(n_ACT * (P - d_min * t_ACT) + 1, t_stop - n_ACT * d_min * t_ACT)
15     n_entries = n_entries + d_min
16 n_entries = n_entries + t_stop/t_ACT

```

Лістинг 4.2 – Імітація атаки з максимальною кількістю записів

Сам алгоритм ділиться на кілька сегментів. Перший сегмент, від рядка 5 до рядка 9, використовується для повторного визначення значень P і c_{min} . Функція `findCmin` знаходить значення c_{min} за допомогою рівняння 4.4. Ітерація припиняється, коли значення стабілізуються. Це має статися дуже швидко, оскільки обидва значення лише незначно відрізняються і вони обидва округлені до цілих значень. Наступний сегмент у рядках 10 і 11 імітує перший d_{min} АСТs,

встановлюючи перше значення для часу до найранішого закінчення t_{stop} і ініціалізація кількості записів $n_{записи}$. Третій сегмент, від рядка 12 до рядка 16, є ядром алгоритму. Рядки виділені групами d_{min} , і видали достатньо АТС, щоб мати мінімальний вплив t_{stop} . після цього t_{stop} перераховується як час до найранішого видалення. Якщо рядків було активовано більше, ніж потрібно, щоб не впливати t_{stop} , найперші рядки будуть стерті механізмом раніше, ніж цей, і буде менше часу для активації інших рядків, що призведе до використання меншої кількості записів. Коли $t_{stop} \leq P + 1$, цикл більше не потрібен, як t_{stop}/t_{ATC} рядків можна відправити по одній АТС без впливу t_{stop} .

Як тільки перший найраніший час закінчення t_{stop} досягнуто, всі інші АСТ замінюють лише існуючі записи, які поступово стиратимуться. Як пояснювалося раніше, якщо в таблиці більше ніж P записів, сума усіх $t_{docvid} - t$ записів у таблиці може лише зменшуватися, що призводить до швидкого видалення значень. Остаточне значення $n_{записи}$ у цьому алгоритмі визначається кількість записів таблиці, необхідних для уникнення випадку, коли рядок не має місця в таблиці, коли видається АСТ. Однак, як зазначалося раніше, кількість використаних записів рідко перевищуватиме $P \div t_{ATC}$ під час нормальної роботи. Щоб зменшити споживання енергії, механізм міг би реалізувати необхідні схеми для відключення великих частин масиву, коли вони не використовуються.

4.3 Тестування отриманих моделей захисту

При тестуванні моделей машинного навчання важливо зрозуміти чи оптимальну кількість нейронів використано для того чи іншого варіанту тестування. Це важливо для того щоб мінімізувати час на проходження даних крізь нейромережу і визначення результату чи відбувається атака.

Так як найкращою в розмірності часу та точності була обрана модель мультирівневого персептрона MLP то потрібно знайти чи найоптимальніша

конфігурація була використана. Для цього візьмемо відповідні розмірності шарів і почнемо їх варіювати відповідно в кожній ітерації навчання. Так ми зможемо знайти зміну точності в залежності від зміни розмірності, при цьому залишивши всі інші значення константними. Першим параметром є перший Dense шар після вхідного, що має розмірність від 1 до `n_timesteps`. Змодельовані результати навчання в таких межах можемо бачити на рисунку.

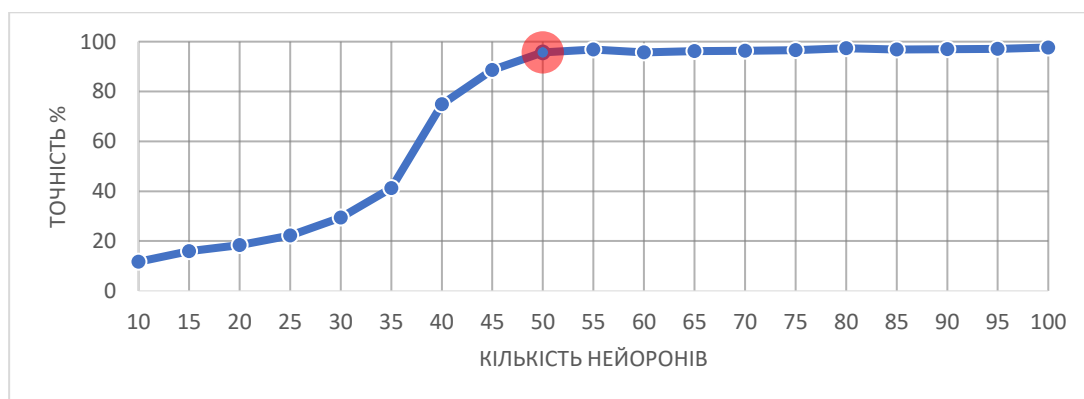


Рисунок 4.3 – Точність моделі MLP при різній розмірності першого шару

Як бачимо при значенні $n_timesteps / 2$ точність не покращується, тож це значення буде використано для подальших тренувань та імплементацій. Після цього слідує другий шар мережі після Flatten функції з relu видом активації. Тут розмірність не прив'язана до вхідної розмірності а скоріше до вихідної. Як зазначає автор статті про оптимізацію MLP систем [95] значення ьтакого шару краще обирати як один зі степенів вихідного шару, тобто степені двійки в нашому випадку. Таке порівняння після навчання можна бачити на рисунку 4.4.

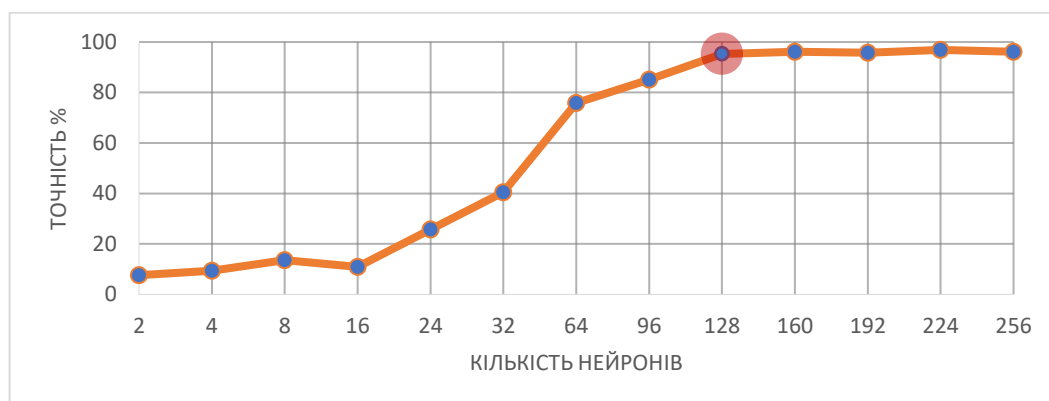


Рисунок 4.4 – Точність моделі MLP при різній розмірності другого шару

Тепер оцінивши розмірність мережі потрібно оцінити її накладані витрати на пам'ять та енергостоживання. Щодо першого накладна пам'ять для такої мережі була розрахована як 0.086 Мб що є дуже невеликим розміром, можливим для імплементації навіть на контролері DRAM. Але контролер має обмежену політику по використанню потужності тож виміряємо які саме створює наша неймережа. Ми вимірюємо енергоспоживання DRAM в нашому ПК за допомогою кількох підходів, поєднуючи програмні та апаратні інструменти, щоб отримати точні й надійні результати.

Для початку ми використовуємо програмні інструменти, такі як Open Hardware Monitor, щоб отримати загальну інформацію про потужність, яку споживає вся система. Ця програма дозволяє нам відслідковувати температури, напруги та загальну потужність комп'ютера, що дає розуміння про те, як працюють компоненти в системі. Однак ці інструменти вимірюють споживану потужність на рівні всього ПК, тому для точнішої оцінки енергоспоживання саме DRAM ми комбінуємо їх із більш спеціалізованими підходами. Ми також використовуємо зовнішній енергометри, підключаючи їх між блоком живлення та комп'ютером для вимірювання загальної потужності, яку споживає вся система. Ці пристрої допомагають відстежувати споживану потужність під час виконання різних навантажень. Але оскільки енергометри вимірюють потужність всієї системи, нам потрібно спеціально спостерігати за змінами потужності, щоб зробити висновки щодо енергоспоживання DRAM. Для ще більш детального вимірювання енергоспоживання DRAM, ми використовуємо спеціалізовані датчики енергоспоживання, які можуть бути підключені через інтерфейси, як PMBus або I2C. Ці датчики дають змогу точно вимірювати енергоспоживання на рівні окремих чіпів DRAM, дозволяючи детально вивчити поведінку пам'яті під час виконання різних задач.

По завершенню тестів ми використовуємо програмне забезпечення для аналізу отриманих даних. Так як в тестованій системі стоїть процесор від Intel,

через Intel VTune Profiler ми можемо відстежити, як різні операції з пам'яттю (запис, читання, оновлення) впливають на енергоспоживання. Це дає нам змогу не лише виміряти потужність, але й оптимізувати її в рамках конкретних задач, зменшуючи споживання енергії під час роботи системи.

Загалом, вимірювання енергоспоживання DRAM в нашому ПК є комбінованим підходом, результати якого продемонстровано на рисунку 4.5. Тут ми бачимо тести зроблені для пам'яті типу DDR3 та DDR4.

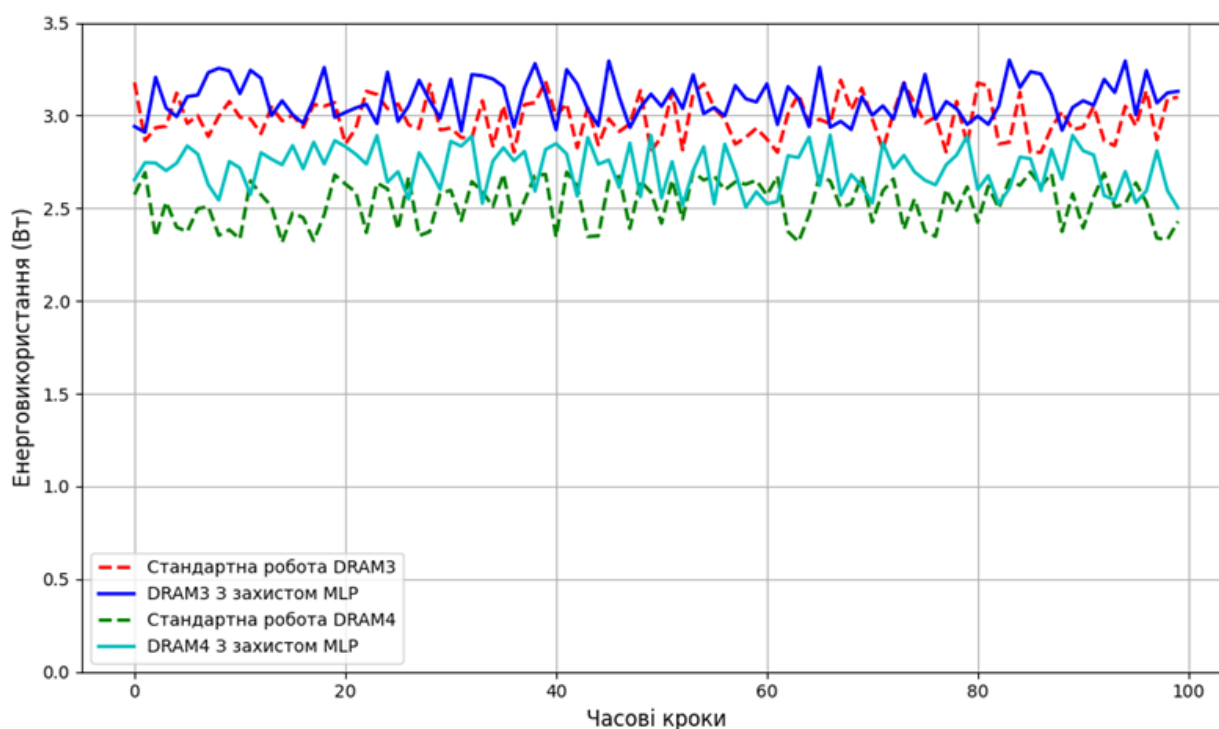


Рисунок 4.5 – Додаткове енерговикористання захисту MLP для DRAM

Середнє накладне навантаження на енергоспоживання склало 0.14 Вт що є достатнім рівнем для мінімального впливу на систему. При цьому важливо розуміти, що це є локалізованим значенням на яке впливає багато шумів та наведень і ця цифра є в районі похибки для сучасних обчислювальних систем.

Тестування імплементованих нейромереж представлено в Таблиці 4.1. Тут ми бачимо різницю між тренуванням на датасеті відповідної пам'яті та загальному датасеті доступів. Різниця обумовлена різною частотою і

відповідно різними параметрами порогових значень, що мають бути відповідними до типу пам'яті.

Таблиця 4.1 – виявлення на основі нейромережі відповідно до типу пам'яті та типі датасету

Тип пам'яті	Частота пам'яті, МГц	Виявлення атаки (Локальне тренування)	Виявлення атаки (Загальна мережа)
DDR3	1600	99.62%	92.34%
DDR3	1866	99.11%	91.14%
DDR4	2666	99.52%	89.38%
DDR4	3200	99.51%	90.41%
DDR5	5200	99.37%	88.95%

Розглядаючи ж типову імплементацію для захисту на основі лічильників можна обрати вже досліджену DDR4. Використовуючи параметри синхронізації, наведені в таблиці 1.1 ,і $A_{RH} = 16384$. Для впровадження на рівні банку відповідні параметри часу є такими: $N_C = 1.33 \times 10^6$, $d_{min} = 2$.

Для простоти виберемо $t_{ATC} = 1$, тобто, один тік за АСТ. Алгоритм показаний в лістингу 4.2 дає нам значення $P = 83$, $c_{min} = 130$ і $n_{записи} = 447$. Зверніть увагу, що в цій конфігурації значення P і c_{min} достатньо малі, щоб їх можна було визначити лише за одну ітерацію. Максимальне значення, якого може досягнути лічильник, $c_{max} = 16189$

Зі значеннями P і c_{min} , ми можемо порахувати $(t_{exp} - t)_{макс} = 10452$, і також розмір таймера для різних значень $n_{циклів}$. Для $n_{циклів} = 2$, таймер повинен утримувати $2 \times (t_{exp} - t)_{макс} = 20904$, що вимагає принаймні 15 біт. для $n_{циклів} = 3$, таймер мав би утримувати лише до $1.5 \times (t_{exp} - t)_{макс} = 15678$, що вимагає 14 біт. Розглядаючи типовий DDR4 з $2^{16} = 65536$ рядів, для $n_{циклів} = 2$, кожен запис таблиці складатиметься з 16(ідентифікатор рядка)+15(час до видалення)+14(лічильник АСТ) = 45біт. Таблиця має $n_{записи} = 447$ записів, загальний розмір таблиці становитиме $447 \times 45 = 19.6Kib$ (включаючи як CAM, так і масив CNT).

Таблиця 4.2 – параметри алгоритму для різних типів пам'яті

DDR тип	A_{RH} , тис	t_{REFW} , мс	N_C , млн	t_{ACT}	Лічильники	Розмір масивів CAM _P +CAM _T	Розмір масиву CNT
DDR3	69,2	64	1,25	1	114	3.35 Kib	1.91 Kib
DDR4	16	64	1,33	1/12	465	12.2 Kib	6.41 Kib
DDR5	4,8	32	0,661	1/28	701	17.6 Kib	8.25 Kib

Таблиця 4.2 містить перелік розмірів алгоритму для DDR3, DDR4 і DDR5. Параметри синхронізації, які використовуються для створення цієї таблиці, наведені в таблиці 1.1. Значення A_{RH} для DDR3 є першим розглянутим значенням для цього покоління DDR [24]. Значення A_{RH} для DDR5 ще не визначено, тож ми знайшли для цього значення практично з тестової вибірки розділу 2.

Для порівняння, реалізація Graphene на рівні банку для розглянутої DDR4 буде використовувати відповідно CAM 4,8 Кіб, а BlockHammer для тієї ж

конфігурації використовуватиме масив лічильників 26 Кіб. Тобто така реалізація не тільки не матиме недоліків з нерівномірним оновленням лічильників, а і займатиме значно менше місця 31.5 % у першому і 24.5 % у другому випадку відповідно.

Крім того, такий підхід дозволяє значно покращити швидкість детектування атак, оскільки обробка відбувається без необхідності тривалих обчислень, що характерно для методів на основі штучного інтелекту. Недоліком ж такого підходу є необхідність у точних даних щодо атак на конкретний тип представленої пам'яті, тобто при виході нових версій пам'яті потрібен час на дослідження перед імплементацією.

Висновки до розділу 4

У цьому розділі було здійснено імплементацію моделей захисту у реальну систему, що дозволило оцінити їхню ефективність у практичному застосуванні. На початковому етапі були визначені характеристики тестової системи, включаючи вибір процесора, материнської плати та SSD, а також використані модулі пам'яті DDR3, DDR4 і DDR5 для проведення тестування. Такий підхід дозволив оцінити роботу механізмів захисту на різних поколіннях пам'яті та визначити їхню ефективність у реальних умовах експлуатації.

Було реалізовано детектор атак на основі нейромережі MLP, який продемонстрував високий відсоток виявлення атак при навчанні на локально згенерованих даних. Це свідчить про те, що адаптація моделі під конкретне обладнання та умови експлуатації значно підвищує її ефективність. Однак було також встановлено, що хоча метод машинного навчання забезпечує високу точність, він не гарантує повного захисту від усіх можливих варіантів атак, що є його основним обмеженням.

Крім того, впроваджено модель детектування атак на основі частотного масиву, який також продемонструвала відмінні результати. Ця модель захисту виявилася більш жорсткою щодо запобігання атакам, оскільки вона покладається на чітко визначені порогові значення кількості доступів до пам'яті. Також зазначена модель виявилась більш ефективною у використанні пам'яті порівняно з існуючими рішеннями, такими як Graphene і BlockHammer, зменшуючи використання пам'яті на лічильники на 31% та 24,5% відповідно. Це забезпечує ефективний захист, проте його основним недоліком є необхідність заздалегідь визначати ці порогові значення, що може обмежити його ефективність при зміні характеристик нових модулів пам'яті.

Таким чином, отримані результати свідчать про те, що обидві продемонстровані моделі захисту мають свої переваги та недоліки. Модель на основі машинного навчання є гнучкою і добре адаптується до нових типів пам'яті, але не дає абсолютного захисту, тоді як модель частотного аналізу забезпечує повний захист, але значною мірою залежить від наперед визначених параметрів. Подальші дослідження можуть бути спрямовані на поєднання цих підходів до захисту, що дозволить досягти балансу між точністю, швидкістю виявлення та універсальністю захисту для різних видів пам'яті.

ВИСНОВКИ

Отже у ході дисертаційного дослідження було проведено всебічний аналіз роботи оперативної пам'яті, її архітектурних особливостей і механізмів, а також детально розглянуто атаку RowHammer, яка здатна змінювати значення бітів у пам'яті без доступу до них. Було продемонстровано небезпеку цієї атаки та її актуальність, оскільки збільшення густини комірок пам'яті робить сучасні модулі більш вразливими до фізичних атак на рівні апаратної реалізації та доступу. Було наголошено, що на даний момент подібні атаки неможливо відрізнити від звичайної роботи системи, що робить їх особливо небезпечними. Крім того, було проведено аналіз існуючих механізмів захисту та показано їхні недоліки, серед яких значне використання енергії, неможливість роботи в реальному часі або недостатня ефективність у нових поколіннях пам'яті.

Розроблено та створено тестову платформу на основі FPGA для тестування чіпів пам'яті на вразливість до атаки RowHammer. Було підготовлено універсальну програмну оболонку, що підтримує різні типи DDR-пам'яті та дозволяє проводити широкомасштабні експерименти. У межах цього розділу було проведено тестування 254 плат пам'яті різних поколінь на предмет зміни бітів під впливом частих доступів до сусідніх рядків. Вперше було зібрано сучасні дані щодо рівня їхнього захисту та структуровано інформацію для подальшого аналізу. Тестування продемонструвало, що сучасні чіпи DDR4 і DDR5 є особливо вразливими до атак двостороннього RowHammer, що дозволяє зловмисникам легко їх компрометувати. Аналіз отриманих даних також показав, що засоби захисту, реалізовані в бюджетних серіях усіх трьох протестованих виробників, виявилися недостатньо ефективними для сучасних DRAM-чіпів, які мають підвищену чутливість до паразитних збурень. Виявлені вразливості підкреслюють необхідність подальшого вдосконалення механізмів захисту, зокрема шляхом розробки комбінованих підходів, що поєднують апаратні та програмні методи для виявлення атак і мінімізації їхнього впливу.

На основі отриманих даних було проведено оцінку патернів доступу та виявлено закономірності, що можуть бути використані для покращення існуючих методів захисту. Також сформульовано та продемонстровано методологію збору даних щодо захисту нових систем пам'яті від атак типу RowHammer.

Продemonстровано три моделі виявлення та захисту від атак типу RowHammer. Перша з них передбачає зменшення деталізації підрахунку на рівні рангу, що дозволило скоротити зайняту пам'ять лічильниками до 67% у DDR5, роблячи метод більш ефективним для сучасних систем. Друга модель захисту базується на використанні методів машинного навчання, зокрема моделей LSTM, MLP та CNN. Було визначено, що найкращим варіантом є модель MLP, яка досягла 99,7% точності виявлення атак при часі виявлення в 6,9 мкс. Третя модель ґрунтується на методі виявлення атак за допомогою частотного масиву в Content Addressable Memory (CAM), що використовує визначені порогові значення кількості доступів до пам'яті. Дана модель захисту продемонструвала зниження споживання пам'яті на 31% у порівнянні з Graphene та на 24,5% у порівнянні з BlockHammer, що свідчить про його високу ефективність при мінімальних витратах ресурсів.

Також показано результати імплементації запропонованих моделей захисних механізмів у реальну систему. На початковому етапі були сформовані характеристики тестового середовища, зокрема обрано процесор, материнську плату, SSD та модулі пам'яті DDR3, DDR4 і DDR5. Було проведено імплементацію детектора атак на основі нейромережі MLP, яка показала середню ефективність в 99.5% при навчанні на локально згенерованих даних і відповідно 92.6% при використанні загального датасету зі зниженням до 88.9% для DDR5. Було показано, що адаптація моделі до конкретного обладнання суттєво підвищує точність виявлення атак. Однак цей метод не забезпечує абсолютного захисту від усіх можливих варіантів атак. Також було реалізовано

модель детектування атак на основі частотного масиву, яка продемонструвала більш жорстку політику запобігання атакам, оскільки покладається на заздалегідь визначені порогові значення. Попри високу ефективність, така модель захисту має обмеження у вигляді необхідності попереднього калібрування під конкретні параметри пам'яті, що може знизити його універсальність.

Досліджено перспективи імплементації розроблених моделей захисту від RowHammer у широкій перспективі на існуючі обчислювальні системи а саме те, що модель машинного навчання здатна працювати на нових платах пам'яті за допомогою загального набору даних і не потребує значних ресурсів системи. Модель захисту на основі частотного масиву ж демонструє вищий рівень виявлення атак, при цьому він є залежним від конкретних записаних параметрів системи. Також сформовано можливі пропозиції до покращення, а саме імплементація онлайн навчання, та можливе комбінування частотних масивів і моделі машинного навчання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mazurok, V., & Lutsenko, V. «An analytical overview and trend analysis of RowHammer vulnerabilities for various DRAM vendors.» Social Development and Security, 14(3), 238-244. <https://doi.org/10.33445/sds.2024.14.3.16> (date of access: 26.12.2024)
2. Mazurok, V., & Lutsenko, V. «Improving the effectiveness of Row-Sampling methods to protect against Row-Hammer attacks». Social Development and Security, 14(6), P 61-67. <https://doi.org/10.33445/sds.2024.14.6.7> (date of access: 26.12.2024)
3. Mazurok, V., & Lutsenko, V. «Enhancing Row-Sampling-Based RowHammer defense methods with Machine Learning approach» Theoretical and Applied Cyber Security Vol. 6 No. 2 P 31-35 <https://doi.org/10.20535/tacs.2664-29132024.2.319008> (date of access: 26.12.2024)
4. Yoongu Kim et al. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In ISCA, 2014.
5. Andrew Kwong et al. Rambleed: Reading bits in memory without accessing them. In SP, 2020.
6. Moritz Lipp et al. Nethammer: Inducing RowHammer faults through network requests. In EuroS&PW, 2020.
7. Daniel Gruss et al. RowHammer.js: A remote software-induced fault attack in javascript. In DIMVA, 2016.
8. J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. Usenix Security Symposium, 2008. 12
9. JEDEC. JESD79-3 DDR3 SDRAM, 2013. DRL: https://www.splunk.com/en_us/pdfs/gated/ebooks/state-of-security-2022.pdf (час доступу 02.01.2025)

10. Pietro Frigo, Emanuele Vannacc, Hasan Hassan, Victor Van Der Veen, OnurMutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the many sides of target row refresh. In 2020 IEEE Symposium on Security and Privacy (SP), p 747–762. IEEE, 2020. 18, 19, 23, 54
11. Micron. DDR5 SDRAM product core datasheet, 2021. 12
12. Michael Redeker, Bruce F Cockburn, and Duncan G Elliott. An investigation into crosstalk noise in DRAM structures. In Proceedings of the 2002 IEEE International Workshop on Memory Technology, Design and Testing (MTDT2002), p 123–129. IEEE, 2002. 14
13. Zaid Al-Ars, Said Hamdioui, Ad Van De Goor, Georgi Gaydadjiev, and Joerg Vollrath. DRAM-specific space of memory tests. In 2006 IEEE International Test Conference, p 1–10. IEEE, 2006.
14. Al-Ars, Z.; Hamdioui, S.; Van De Goor, A.; Gaydadjiev, G.; Vollrath, J. DRAM-specific space of memory tests. In Proceedings of the 2006 IEEE International Test Conference, Santa Clara, CA, USA, 22–27 October 2006; pp. 1–10.
15. Chao, M.C.T.; Yang, H.Y.; Huang, R.F.; Lin, S.C.; Chin, C.Y. Fault models for embedded-DRAM macros. In Proceedings of the 46th Annual Design Automation Conference, San Francisco, CA, USA, 26–31 July 2009.
16. Chia, P.C.F.; Wen, S.J.; Baeg, S.H. New DRAM HCI qualification method emphasizing on repeated memory access. In Proceedings of the 2010 IEEE International Integrated Reliability Workshop Final Report, South Lake Tahoe, CA, USA, 17–21 October 2010.
17. SemiEngineering. Side Channel Attacks A class of attacks on a device and its contents by analyzing information using different access methods. https://semiengineering.com/knowledge_centers/semiconductor-security/side-channel-attacks/ 2020. (date of access: 26.12.2024)

18. Pierre Chor-Fung Chia, Shi-Jie Wen, and Sang H Baeg. New DRAM HCI qualification method emphasizing on repeated memory access. In 2010 IEEE International Integrated Reliability Workshop Final Report, p 142–144. IEEE, 2010.
19. Thomas Yang and Xi-Wei Lin. Trap-assisted dram row hammer effect. *IEEE Electron Device Letters*, 40(3):391–394, 2019.
20. Onur Mutlu and Jeremie S Kim. RowHammer: A retrospective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(8):1555–1571, 2019.
21. Jeremie S Kim, Minesh Patel, A Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. Revisiting RowHammer: An experimental analysis of modern dram devices and mitigation techniques. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), p 638–651. IEEE, 2020.
22. Seaborn, M.; Dullien, T. Exploiting the DRAM RowHammer bug to gain kernel privileges. *Black Hat 2015*, 15, 71; Yuan, X.; Zhang, X.; Zhang, Y.; Teodorescu, R. One bit flips, one cloud flops: {Cross-VM} row hammer attacks and privilege escalation. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, USA, 10–12 August 2016; pp. 19–35
23. Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic RowHammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, p 1675–1689, 2016.
24. Rui, Q.; Seaborn, M. A new approach for RowHammer attacks. In *Proceedings of the 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, USA, 3–5 May 2016; pp. 161–166.
25. Daniel, G.; Bidner, D.; Mangard, S. Practical memory deduplication attacks in sandboxed javascript. In *Computer Security— ESORICS 2015: 20th European Symposium on Research in Computer Security*, Vienna, Austria, 21–25

September 2015; Proceedings, Part I 20; Springer International Publishing: Berlin/Heidelberg, Germany, 2015; pp. 108–122.

26. Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. Triggering RowHammer hardware faults on ARM: A revisit. In Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security, p 24–33, 2018.

27. Zelalem Birhanu Aweke et al. ANVIL: Software-based protection against nextgeneration RowHammer attacks. SIGPLAN Notices, 2016

28. Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Grand pwning unit: Accelerating microarchitectural attacks with the GPU. In 2018 ieee symposium on security and privacy (sp), p 195–210. IEEE, 2018

29. Moritz Lipp. Cache attacks and RowHammer on arm. PhD thesis, Graz University of Technology, 2016.

30. Andreas, K.; Juffinger, J.; Qazi, S.; Kim, Y.; Lipp, M.; Boichat, N.; Shiu, E.; Nissler, M.; Gruss, D. {Half-Double}: Hammering From the Next Row Over. In Proceedings of the 31st USENIX Security Symposium (USENIX Security 22), Boston, MA, USA, 10–12 August 2022; pp. 3807–3824.

31. Finn de Ridder et al. SMASH: Synchronized many-sided RowHammer attacks from javascript. In USENIX Security, 2021.

32. Salman Qazi et al. “Half-Double”: Next-row-over assisted RowHammer. https://github.com/google/hammer-kit/blob/main/20210525_half_double.pdf, 2021.

33. Mark Seaborn and Thomas Dullien. Exploiting the DRAM RowHammer bug to gain kernel privileges. Black Hat, 2015.

34. Sarani Bhattacharya and Debdeep Mukhopadhyay. Curious case of RowHammer: flipping secret exponent bits using timing analysis. In International Conference on Cryptographic Hardware and Embedded Systems, p 602–624. Springer, 2016.

35. Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In 25th USENIX Security Symposium (USENIX Security 16), p 1–18, 2016.
36. "BIOS and kernel developer's guide (BKDG) for AMD family 10h processors, 2013.
37. Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM addressing for Cross-CPU attacks. In 25th USENIX security symposium (USENIX security 16), p 565–581, 2016.
38. Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: {Cross-VM} row hammer attacks and privilege escalation In 25th USENIX security symposium (USENIX Security 16), p 19–35, 2016.
39. lessandro Barengi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. Software-only reverse engineering of physical dram mappings for RowHammerattacks. In 2018 IEEE 3rd International Verification and Security Workshop (IVSW), p 19–24. IEEE, 2018. 18
40. Mohamed Hassan, Anirudh M Kaushik, and Hiren Patel. Reverse-engineering embedded memory controllers through latency-based analysis. In 21st IEEE Real-Time and Embedded Technology and Applications Symposium, p 297–306. IEEE, 2015. 18
41. Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A case for exploiting subarray-level parallelism (salp) in dram. In 2012 39th Annual International Symposium on Computer Architecture (ISCA), p 368–379. IEEE, 2012. 18
42. Tao Zhang, Boris Pismenny, Donald E Porter, Dan Tsafir, and Aviad Zuck. RowHammering storage devices. In Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems, p 77–85, 2021. 18
43. Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating software mitigations against RowHammer: a surgical precision hammer. In

International Symposium on Research in Attacks, Intrusions, and Defenses, p 47–66. Springer, 2018. 18

44. Manaar Alam et al. Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks. IACR Cryptology ePrint Archive, 2017. 20 107

45. Anirban Chakraborty et al. Deep learning based diagnostics for RowHammer protection of DRAM chips. In ATS, 2019. 20, 29, 54

46. Anirban Chakraborty, Manaar Alam, and Debdeep Mukhopadhyay. A good anvil fears no hammer: Automated RowHammer detection using unsupervised deep learning. In International Conference on Applied Cryptography and Network Security, p 59–77. Springer, 2021. 20

47. Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. ZebRAM: Comprehensive and compatible software protection against RowHammer attacks. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), p 697–710, 2018. 20

48. Yicheng Wang, Yang Liu, Peiyun Wu, and Zhao Zhang. Discreet-PARA: RowHammer defense with low cost and high efficiency. In 2021 IEEE 39th International Conference on Computer Design (ICCD), p 433–441. IEEE, 2021. 20

49. Mungyu Son, Hyunsun Park, Junwhan Ahn, and Sungjoo Yoo. Making DRAM stronger against row hammering. In DAC, 2017. 20, 27

50. Jung Min You and Joon-Sung Yang. MRLoc: Mitigating row-hammering based on memory locality. In DAC, 2019. 21, 28

51. Li Fan, Pei Cao, Jussara Almeida, and Andrei Z Broder. Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM transactions on networking, 8(3):281–293, 2000. 21

52. Kwangrae Kim, Jeonghyun Woo, Junsu Kim, and Ki-Seok Chung. Hammerfilter: Robust protection and low hardware overhead method for

RowHammer. In 2021 IEEE 39th International Conference on Computer Design (ICCD), p 212–219. IEEE, 2021. 21

53. Biresk Kumar Joardar, Tyler K Bletsch, and Krishnendu Chakrabarty. Learning to mitigate RowHammer attacks. In 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), p 564–567. IEEE, 2022. 21, 29, 100108

54. Seyed Mohammad Seyedzadeh, Alex K Jones, and Rami Melhem. Counter-based tree structure for row hammering mitigation in dram. *IEEE Computer Architecture Letters*, 16(1):18–21, 2016. 22

55. Seyed Mohammad Seyedzadeh, Alex K Jones, and Rami Melhem. Mitigating wordline crosstalk using adaptive trees of counters. In *ISCA*, 2018. 22

56. Ingab Kang, Eojin Lee, and Jung Ho Ahn. Cat-two: Counter-based adaptive tree, time window optimized for dram row-hammer prevention. *IEEE Access*, 8:17366–17377, 2020. 22, 50

57. A Giray Yağlıkçı et al. Blockhammer: Preventing RowHammer at low cost by blacklisting rapidly-accessed DRAM rows. In *HPCA*, 2021. 22, 27, 46, 62

58. Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982. 22, 46

59. Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International conference on database theory*, p 398–412. Springer, 2005. 22

60. Yeonhong Park et al. Graphene: Strong yet lightweight row hammer protection. In *MICRO*, 2020. 22, 27, 37, 46, 62

61. Gururaj Saileshwar, Bolin Wang, Moinuddin Qureshi, and Prashant J Nair. Randomized row-swap: mitigating row hammer by breaking spatial correlation between aggressor and victim rows. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, p 1056–1069, 2022. 22

62. Michael Jaemin Kim, Jaehyun Park, Yeonhong Park, Wanju Doh, Namhoon Kim, Tae Jun Ham, Jae W Lee, and Jung Ho Ahn. Mithril: Cooperative row

hammerprotection on commodity dram leveraging managed refresh. In 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), p1156–1169. IEEE, 2022. 22 109

63. Tanj Bennett, Stefan Saroiu, Alec Wolman, and Lucian Cojocar. Panopticon:A complete in-dram RowHammer mitigation. In Workshop on DRAM Security(DRAMSec), 2021. 22

64. Eojin Lee, Ingab Kang, Sukhan Lee, G Edward Suh, and Jung Ho Ahn. TWiCe:Preventing row-hammering by exploiting time window counters. In Proceedingsof the 46th International Symposium on Computer Architecture, p 385–396, 2019.22, 27

65. Kuljit Bains, John Halbert, Christopher Mozak, Theodore Schoenborn, andZvika Greenfield. Row hammer refresh command, August 25 2015. US Patent9,117,544. 23

66. Kevin Loughlin, Stefan Saroiu, Alec Wolman, and Baris Kasikci. Stop! hammertime: rethinking our approach to RowHammer mitigations. In Proceedings of theWorkshop on Hot Topics in Operating Systems, p 88–95, 2021. 23

67. Kuljit S Bains and John B Halbert. Row hammer monitoring based on stored RowHammer threshold value, August 1 2017. US Patent 9,721,643. 23

68. Hasan Hassan, Yahya Can Tugrul, Jeremie S Kim, Victor Van der Veen, KavehRazavi, and Onur Mutlu. Uncovering in-dram RowHammer protection mechanisms: A new methodology, custom RowHammer patterns, and implications. InMICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture,p 1198–1213, 2021. 23

69. Chia-Ming Yang, Chen-Kang Wei, Yu Jing Chang, Tieh-Chiang Wu, Hsiu-PinChen, and Chao-Sung Lai. Suppression of row hammer effect by doping profilemodification in saddle-fin array devices for sub-30-nm dram technology. IEEETransactions on Device and Materials Reliability, 16(4):685–687, 2016. 23

70. Seong-Wan Ryu, Kyungkyu Min, Jungho Shin, Heimi Kwon, Donghoon Nam,Taekyung Oh, Tae-Su Jang, Minsoo Yoo, Yongtaik Kim, and Sungjoo Hong.

Overcoming the reliability limitation in the ultimately scaled dram using silicon migration technique by hydrogen annealing. In 2017 IEEE International ElectronDevices Meeting (IEDM), p 21–6. IEEE, 2017. 23110

71. S Agarwal, H Dixit, D Datta, M Tran, D Houssameddine, D Shum, and F Benistant. RowHammer for spin torque based memory: Problem or not? In 2018 IEEE International Magnetics Conference (INTERMAG), p 1–1. IEEE, 2018.

72. Mohammad Nasim Imtiaz Khan and Swaroop Ghosh. Analysis of row hammerattack on STTRAM. In 2018 IEEE 36th International Conference on Computer Design(ICCD), p 75–82. IEEE, 2018.

73. Jung Ho Ahn, Sheng Li, O Seongil, and Norman P Jouppi. Mcsima+: A manycore simulator with application-level+ simulation and detailed microarchitecturemodeling. In 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), p 74–85. IEEE, 2013.

74. Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensibledram simulator. IEEE Computer architecture letters, 15(1):45–49, 2015.

75. Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, AliSaidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, SomayehSardashti, et al. The gem5 simulator. ACM SIGARCH computer architecture news,2011.

76. Loïc France et al. Vulnerability assessment of the RowHammer attack using machine learning and the gem5 simulator - work in progress. In SaT-CPS, 2021.

77. Ayaz Akram and Lina Sawalha. A survey of computer architecture simulationtechniques and tools. Ieee Access, 7:78120–78145, 2019.

78. Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. Mcpat: An integrated power, area, and timing modelingframework for multicore and manycore architectures. In Proceedings of the 42ndannual ieee/acm international symposium on microarchitecture, p 469–480, 2009.

79. Rafael Ubal, Julio Sahuquillo, Salvador Petit, and Pedro Lopez. Multi2Sim: A simulation framework to evaluate multicore-multithreaded processors. In 19th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07), p 62–68. IEEE, 2007.
80. Daniel Aarno and Jakob Engblom. Software and system development using virtual platforms: full-system simulation with Wind River Simics. Morgan Kaufmann, 2014.
81. Todd Austin, Eric Larson, and Dan Ernst. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, 2002.
82. Ashutosh Dhodapkar, Chee How Lim, George Cai, and W Robert Daasch. Temepest: A thermal enabled multi-model power/performance estimator. In International Workshop on Power-Aware Computer Systems, p 112–125. Springer, 2000.
83. Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. DRAMSim2: A cycle accurate memory system simulator. *IEEE computer architecture letters*, 10(1):16–19, 2011.
84. Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. DRAMsim3: a cycle-accurate, thermal-capable DRAM simulator. *IEEE Computer Architecture Letters*, 19(2):106–109, 2020.
85. Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. Are we susceptible to RowHammer? an end-to-end methodology for cloud providers. In 2020 IEEE Symposium on Security and Privacy (SP), p 712–728. IEEE, 2020.
86. Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. *ACM SIGPLAN notices*, 42(6):89–100, 2007.
87. Valgrind. <https://valgrind.org/>.

88. Loïc France, Florent Bruguier, Maria Mushtaq, David Novo, and Pascal Benoit. Implementing RowHammer memory corruption in the gem5 simulator. In 32nd International Workshop on Rapid System Prototyping (RSP). IEEE, 2021.
89. Quentin Forcioli, Jean-Luc Danger, Clémentine Maurice, Lilian Bossuet, Florent Bruguier, Maria Mushtaq, David Novo, Loïc France, Pascal Benoit, Sylvain Guilley, et al. Virtual platform to analyze the security of a system on chip at microarchitectural level. In 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), p 96–102. IEEE, 2021.
90. Loïc France, Florent Bruguier, Maria Mushtaq, David Novo, and Pascal Benoit. Implementation of RowHammer effect in gem5. In 15ème Colloque National du GDR SoC2, 2021.
91. Loïc France, Florent Bruguier, Maria Mushtaq, David Novo, and Pascal Benoit. Modeling RowHammer in the gem5 simulator. CHES 2022 - Conference on Cryptographic Hardware and Embedded Systems, September 2022. Poster. 42
92. Micron. Micron DDR5 SDRAM: New features, 2021.
93. Ataberk Olgun et al. DRAM Bender: An Extensible and Versatile FPGA-based Infrastructure to Easily Test State-of-the-art DRAM Chips, 22(5):651–664, 2022.
94. Supreet Jeloka et al. A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6t bit cell enabling logic-in-memory. JSSC, 2016.
95. Kostas Pagiamtzis and Ali Sheikholeslami. A low-power content-addressable memory (CAM) using pipelined hierarchical search scheme. IEEE Journal of Solid State Circuits, 39(9):1512–1519, 2004.
96. Swapan Kumar Ray. Large-capacity high-throughput low-cost pipelined cam using pipelined ctam. IEEE Transactions on Computers, 55(5):575–587, 2006.
97. Loïc France, Florent Bruguier, David Novo, Maria Mushtaq, and Pascal Benoit. Reducing the silicon area overhead of counter-based RowHammer mitigations. In 18th CryptArchi Workshop, 2022.

98. John D. McCalpin. Memory bandwidth and machine balance in current highperformance computers. IEEE Computer Society Technical Committee on ComputerArchitecture (TCCA) Newsletter, 1995.
99. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neuralcomputation*, 9(8):1735–1780, 1997.
100. Paul Werbos. Beyond regression: new tools for prediction and analysis in thebehavioral sciences. Ph. D. dissertation, Harvard University, 1974.
101. Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard EHoward, Wayne Hubbard, and Lawrence D Jackel. Backpropagation appliedto handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
102. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-basedlearning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
103. François Chollet. Keras. <https://keras.io/>, 2015.
104. Machine learning mastery, 2013.
105. Jason Brownlee. LSTMs for human activity recognition time series classification,2018.
106. Jason Brownlee. 1D convolutional neural network models for human activityrecognition, 2018.
107. Jason Brownlee. Deep learning for time series, 2018.