Rapid Detection of RowHammer Attacks using Dynamic Skewed Hash Tree

SARU VIG

SIEW-KEI LAM

NANYANG TECHNOLOGICAL UNIVERSITY, SINGAPORE

SARANI BHATTACHARYA

DEBDEEP MUKHOPADHYA

INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

Overview

Introduction

Motivation

Proposed Work

Results

Conclusion

Q&A

Introduction

What is the *RowHammer* Attack ?

- DRAM is hierarchically composed of Channels, Rank and Banks.
- Each bank is a collection of cells having typically 2^{14} to 2^{17} rows
- Attackers can repeatedly open (i.e. activate) & close (i.e. precharge) DRAM rows in the same memory bank to induce bit flips in the adjacent rows.

```
Code-hammer
{
        mov (X), %eax      //read from address X
        mov (Y), %ebx      //read from address Y
        clflush (X)        //flush cache for address X
        clflush (Y)        //flush cache for address Y
        jmp Code-hammer
}
```

Introduction

What is the *RowHammer* Attack ?

- In particular, the *repeated* charging and discharging of row cells in a short span of time causes electronic disturbance which could result in bit-flips in the DRAM cells of the adjoining rows.
- The row which is being repeatedly accessed is denoted as the *aggressor* row.
- The two adjoining vulnerable rows, where the flips occur are called the *vulnerable* rows.

Motivation

Existing Strategies

- Hardware resources to maintain the state of DRAM rows
- Selective refreshing
  - High power and performance overhead
  - Increased Latency of memory operations

Memory Integrity Trees

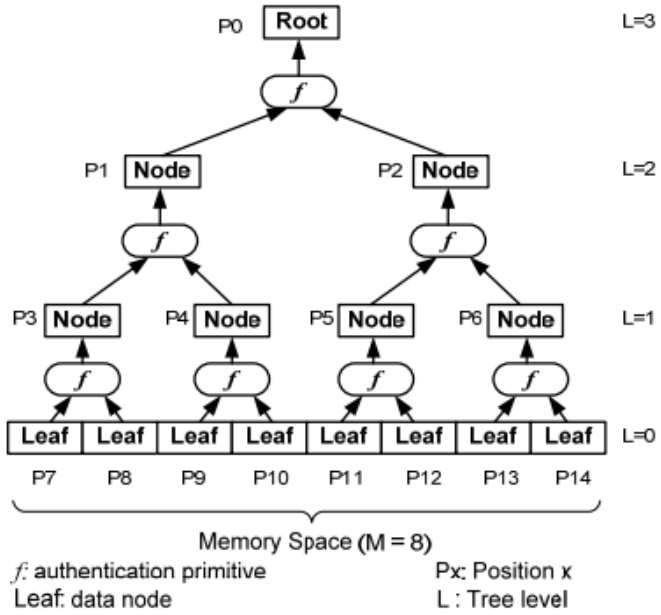
Most memory protection schemes consists of some form of encryption and an Integrity Tree

What are Integrity Trees ?

- Equal sized blocks which form the leaf nodes
- Recursively apply an authentication function to generate tree
- Root hash has to match with the value stored on chip
- Verification is performed on each level while accessing the data stored in leaf nodes

What is a Dynamic Integrity Tree ?

- Tree that can re-structure itself during run-time (add and delete nodes).



General form of a 2-ary binary tree

Proposed Work

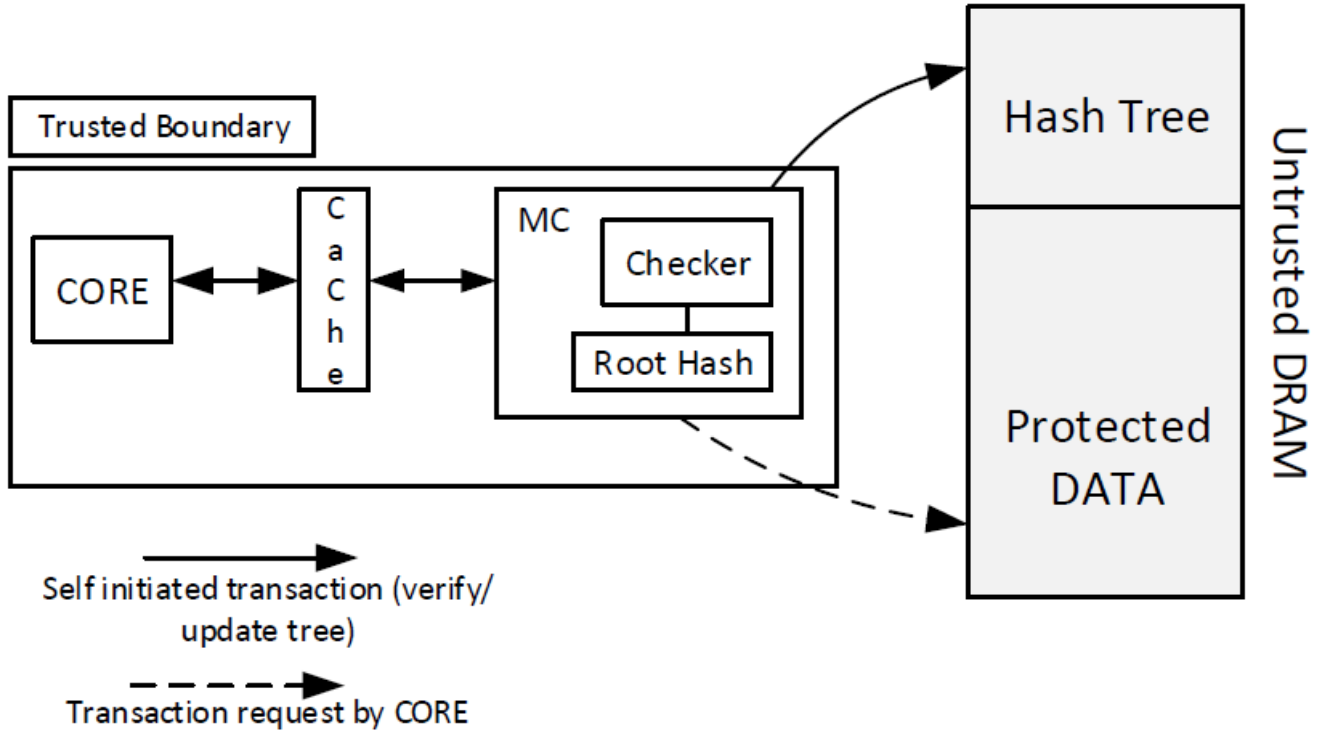
Main Contribution

- A sliding window mechanism is introduced to identify *vulnerable* rows
- Dynamic integrity tree structure is proposed to enable newly detected vulnerable rows to be dynamically inserted into the tree, while rows that are no longer a concern are removed

Mechanism

- RowHammering on processors with DDR3 DRAM was performed. Memory access logs were studied to show that the combination of the sliding window mechanism and dynamic tree structure effectively detects bit flips. Also, it constrains the height of the tree, which enables low-overhead and rapid detection of bit-flips

Framework


Trusted Boundary
CORE
Cache
MC
Checker
Root Hash
Hash Tree
Protected DATA
Untrusted DRAM
Self initiated transaction (verify/ update tree)
Transaction request by CORE

Framework

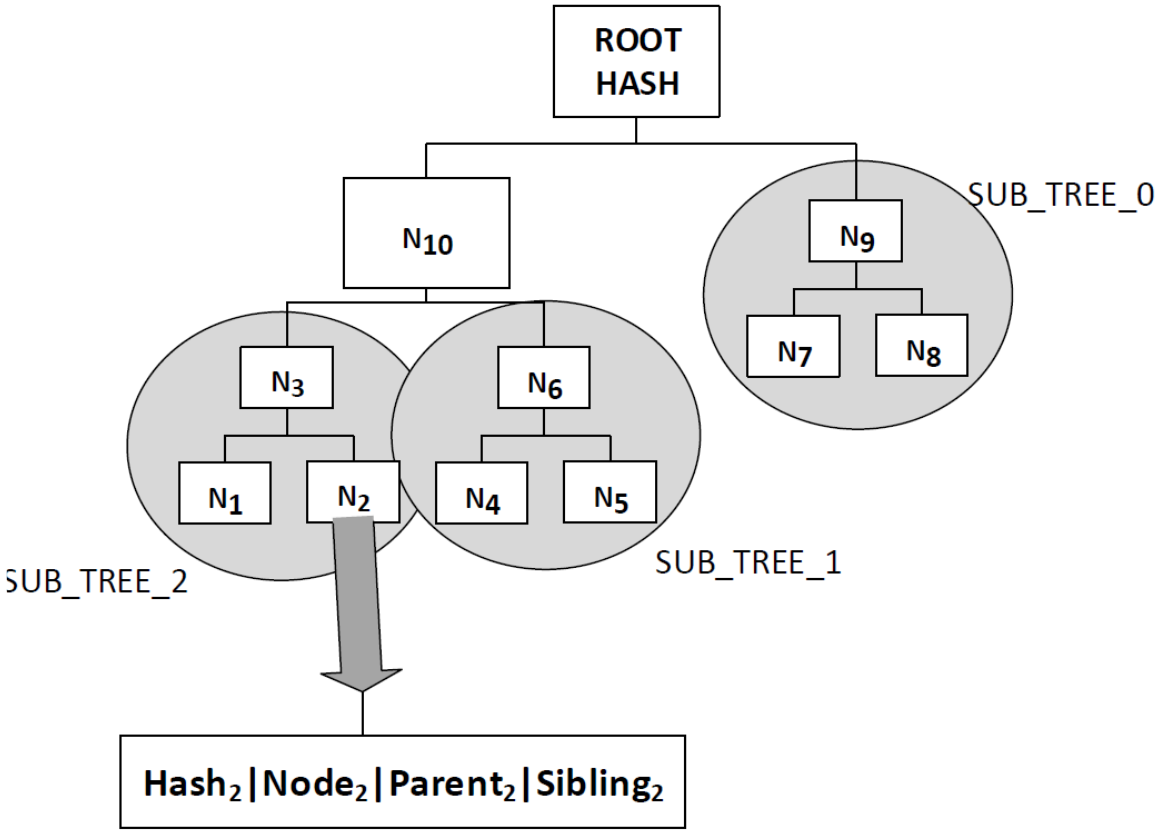
When is a row vulnerable ?

- At least *X* DRAM accesses made to the neighbouring rows from the same bank within window frame of size *p*
- *p* stands for the number of DRAM accesses that the window is going to be monitoring at any given time

Window Frame Size

- Depends on time taken for one DRAM access after performing *clflush* instruction
- For hammering to be successful, a minimum number of DRAM accesses of the same bank must be made within a small activation interval before (~ 500ns) the DRAM refreshes (~64 ms)

Tree Representation



- A SUB_TREE consists of two leaf nodes and their parent. At any one time, we add/remove a single subtree rather than a single node (i.e. two adjoining neighbours of the *aggressor* rows, which form the leaf nodes of the SUB_TREE)
- Tree node structure has additional fields of parent and sibling node number.

Tree Representation

ReadNCheck

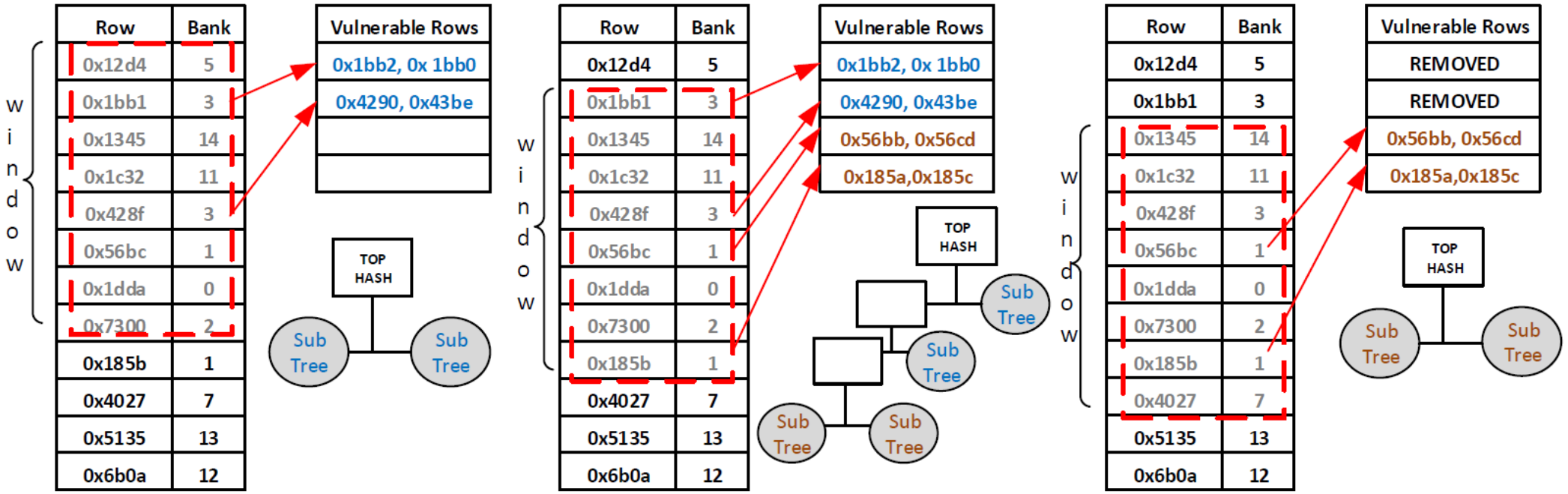
- A recursive procedure to re-calculate the hash at all the levels of the tree and match them with the one already stored in the tree
- The verification will be performed at two instances:
  - If any node of the tree is accessed
  - Whenever a node is removed

Hash Function

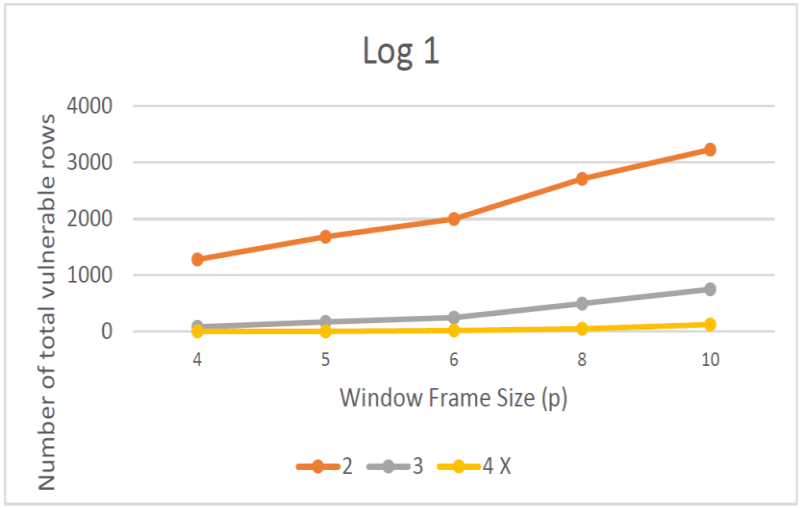
- We use SHA3-256 (Keccak[512] (M || 01, 256)) as the hash algorithm. The output of this function is 256 bits for any given input

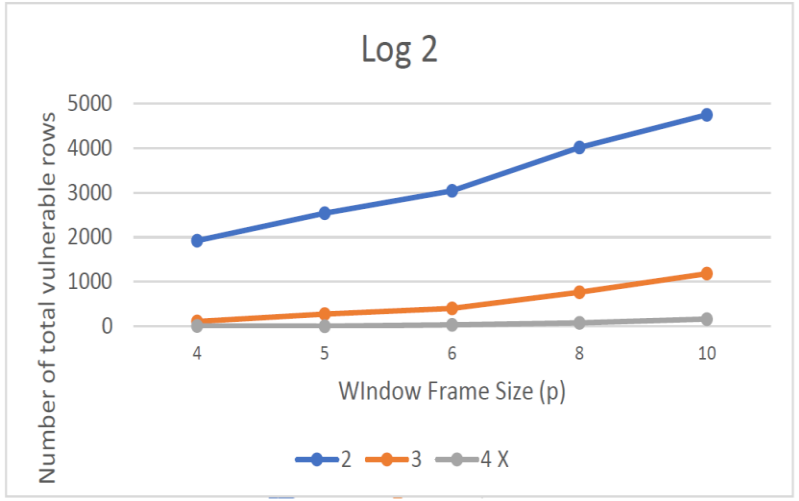
Implementation Example

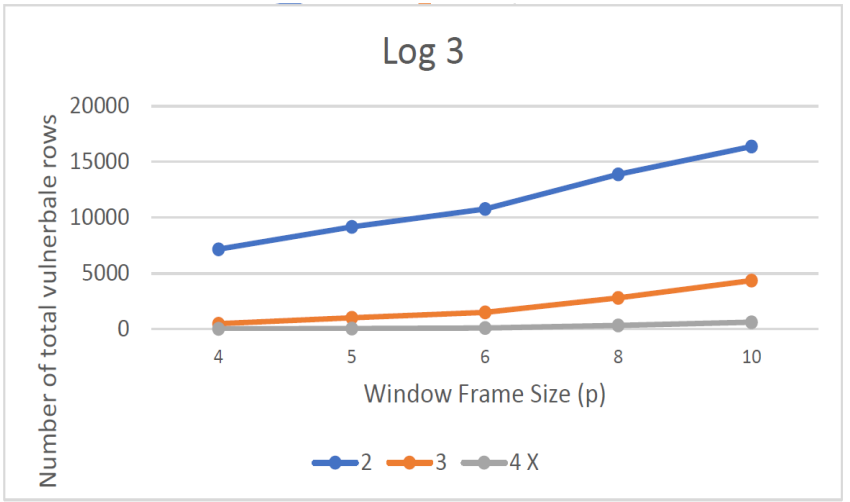
X=2 and p=10



Results







- Examined three different memory logs to identify the patterns exhibiting bank locality
- Varying X and p
- High number of vulnerable and selectively refreshing them without confirming presence of an error will cause a high overhead

Results

Parameters	Processor 1	Processor 2	Processor 3	Processor 4
Clock Frequency	3.5 GHz	2.4 GHz	3.6 GHz	2.7 GHz
Cache Size	12 MB	15 MB	8 MB	4 MB
Memory Type	DDR3	DDR3	DDR3	DDR3
Memory Size	256 GB	64 GB	8 MB	8 MB
Operating System	Windows 7	Windows 7	Windows 7	Windows 7
Repetitions (X)	2	2	2	2
Window Size (p)	10 access	10 access	10 access	10 access
Height of tree	3-4	3-4	3-4	3-4
Avg. no. of leaf nodes	8	8	8	8
Time to make a sub tree	1.91 ms	1.11 ms	1.50 ms	1 ms
Time to add/remove node (depends on tree height)	1.99-5.9 ms	1.28-3.8 ms	1.62-4.97 ms	1.07-4.22 ms

- The experiments revealed that at any given time, the average number of aggressor rows in a single frame is 4 at any given time, with the maximum being 8
- Limits the height of the tree from 3-4 levels

Memory and Time Overhead

For a tree with n leaf nodes the overhead is calculated to be in bits as

- $M_{DT} = (256 + 2 * \log_2 n) * (2 * n - 1)$
- *n* varies from 4 to 8

Total time taken to create a SUB_TREE in all the four cases is ≤ 2ms

Adding/Removing nodes from the tree have a overhead between 2-6 ms depending on which level of tree the update occurs

Additional latency of accessing the memory rows after tree traversal and verification caused by ReadNCheck function pertains only to the victim rows that are accessed while they are a part of the tree. The aggressor rows and other row access are still being read with the same frequency as under normal conditions.

Conclusion

Proposed a framework for rapid detection of multiple bit flips due to RowHammer using dynamic integrity tree, where nodes can be added/removed based on a vulnerability criterion

A sliding window that effectively limits the height of the tree for maintaining vulnerable rows

The criterion and size of the sliding window can be fixed to attain maximum security

Experimental results confirm that the proposed framework will enable rapid detection of bit flips due to RowHammer attack

Future Work:

- More experimental results: Miss Rate, Power, Area etc.
- More quantitative comparison with other existing techniques

Thanks & Questions