U-TRR
Uncovering in-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications

Hasan Hassan

Yahya Can Tugrul    Jeremie S. Kim    Victor van der Veen
Kaveh Razavi        Onur Mutlu

ETH zürich          TOBB ETÜ    Qualcomm

SAFARI
DRAM **RowHammer** vulnerability leads to critical reliability and security issues

**Target Row Refresh (TRR):**
a set of obscure, undocumented, and proprietary RowHammer mitigation techniques

**Is TRR fully secure? How can we validate its security guarantees?**

<table>
<thead>
<tr>
<th>U-TRR</th>
<th>A new methodology that leverages <em>data retention failures</em> to uncover the inner workings of TRR and study its security</th>
</tr>
</thead>
</table>

**High-Level Operation**

1) Profile the retention time of a row $R$
2) Find when TRR refreshes $R$ to understand the underlying TRR mechanism

**15x Vendor A DDR4 modules**

**15x Vendor B DDR4 modules**

**15x Vendor C DDR4 modules**

**All 45 modules we test are **vulnerable**

**99.9% of rows** in a DRAM bank experience **at least one RowHammer bit flip**

**Up to 7 RowHammer bit flips** in an 8-byte dataword, **making ECC ineffective**

**U-TRR can enable **more secure** RowHammer solutions**
Outline

1. DRAM Operation Basics
2. RowHammer & Target Row Refresh
3. The U-TRR Methodology
4. Observations & New RowHammer Access Patterns
5. RowHammer Bit Flip Analysis
6. Takeaways and Conclusion
DRAM Organization

Memory Bus

Memory Controller

CPU
Accessing DRAM

- Precharge
- Activate
- Read/Write

DRAM Bank

- DRAM Cell
- DRAM Row
- Sense Amplifier
DRAM Cell Leakage

Each cell encodes information in **leaky** capacitors

Stored data is **corrupted** if too much charge leaks (i.e., the capacitor voltage degrades too much)

[Patel+, ISCA'17]
Periodic refresh operations preserve stored data

[Patel+, ISCA’17]
Outline

1. DRAM Operation Basics
2. RowHammer & Target Row Refresh
3. The U-TRR Methodology
4. Observations & New RowHammer Access Patterns
5. RowHammer Bit Flip Analysis
6. Takeaways and Conclusion
Repeatedly opening (activating) and closing (precharging) a DRAM row causes RowHammer bit flips in nearby cells.
DRAM vendors equip their DRAM chips with a proprietary mitigation mechanisms known as **Target Row Refresh (TRR)**

**Key Idea:** TRR refreshes nearby rows upon detecting an aggressor row.

![Diagram of TRR-equipped DRAM Chip]

- **Aggressor detected:** Row 2
- **Refresh neighbor rows**
- **TRR-induced refreshes**
The Problem with TRR

TRR is **obscure, undocumented, and proprietary**

We **cannot** easily study the *security properties* of TRR
## Goal

**Study in-DRAM TRR mechanisms to**

1. **understand** how they operate
2. **assess** their security
3. **secure** DRAM completely against RowHammer
Outline

1. DRAM Operation Basics
2. RowHammer & Target Row Refresh
3. The U-TRR Methodology
4. Observations & New RowHammer Access Patterns
5. RowHammer Bit Flip Analysis
6. Takeaways and Conclusion
Overview of U-TRR

**U-TRR**: A new methodology to **uncover** the inner workings of TRR

**Key idea**: Use **data retention failures** as a side channel to **detect when a row is refreshed** by TRR
High-Level U-TRR Operation

U-TRR has two main components:
**Row Scout (RS) and TRR Analyzer (TRR-A)**

**Row Scout:** finds a set of DRAM rows that meet certain requirements as needed by TRR-A and identifies the data retention times of these rows.

**TRR Analyzer:** uses RS-provided rows to distinguish between TRR-induced and regular refreshes, and thus builds an understanding of the underlying TRR mechanism.

---

**Diagram:**
- Profiling Configuration
  - row group layout
  - row group count
  - bank
  - range
  - ...
- Row Scout (RS)
- Retention Profiled Rows (RPR)
- TRR Analyzer (TRR-A)
- Analysis
- RPRs refreshed by TRR-induced refresh
  - aggressor (A) row addr.
  - dummy (D) row addr.
  - hammering mode
  - number of rounds
  - A/D hammer counts
  - REF count
  - ...

---

SAFARI
Row Scout (RS)

**Goal:** Identify a list of *useful* DRAM rows and their *retention times*

Row Scout **must** find:

- **✓** Rows with *consistent* retention times
  - To correctly infer whether a row has been refreshed

- **✓** **Multiple rows** that are located at *certain configurable distances* and have the *same retention time* (i.e., Row Group)
  - To observe whether TRR can refresh multiple rows at the same time

---

* The retention time of a DRAM row may change over time due to Variable Retention Time (VRT) effects

---

SAFARI
Row Scout (RS) Operation

Row Group: \( V \ Y \ V \ Y \ V \)

Profiling the retention time of a DRAM row:
1) write data
2) wait for T
3) check for retention bit flips

Find DRAM rows with retention time T
row addresses

row groups
Combine rows to match the group layout

candidate row groups

Are the candidates enough?

verify retention time consistency

Retain Profiled Rows (RPR)

increase T

NO

Enough row groups pass?

NO

row groups

YES

increase T
**TRR Analyzer (TRR-A)**

**Goal:** Use RS-provided rows to determine when TRR refreshes a victim row

**High-level Operation:**
1) Run a certain **DRAM access pattern** (i.e., RowHammer attack)
2) **Monitor** retention failures in RS-provided rows to determine when TRR refreshes any of these rows
3) Develop an understanding of the **underlying TRR operation**
TRR Analyzer (TRR-A) Operation

Row Group: $V\ A\ V\ A\ V$

V: victim (RS-provided) rows
A: aggressor rows
D: dummy rows

TRR-A helps to understand how TRR operates based on when Retention Profiled Rows are refreshed by TRR
Outline

1. DRAM Operation Basics
2. RowHammer & Target Row Refresh
3. The U-TRR Methodology
4. Observations & New RowHammer Access Patterns
5. RowHammer Bit Flip Analysis
6. Takeaways and Conclusion
We implement U-TRR using FPGA-based SoftMC [Hassan+, HPCA’17] modified to support DDR4 DRAM

<table>
<thead>
<tr>
<th>Module</th>
<th>Date (yy-mm)</th>
<th>Chip Density (Gbit)</th>
<th>Organization</th>
<th>HC\textsubscript{first}†</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>19-50</td>
<td>8</td>
<td>1 16 8</td>
<td>16K</td>
</tr>
<tr>
<td>A1-5</td>
<td>19-36</td>
<td>8</td>
<td>1 8 16</td>
<td>13K-15K</td>
</tr>
<tr>
<td>A6-7</td>
<td>19-45</td>
<td>8</td>
<td>1 8 16</td>
<td>13K-15K</td>
</tr>
<tr>
<td>A8-9</td>
<td>20-07</td>
<td>8</td>
<td>1 16 8</td>
<td>12K-14K</td>
</tr>
<tr>
<td>A10-12</td>
<td>19-51</td>
<td>8</td>
<td>1 8 16</td>
<td>12K-13K</td>
</tr>
<tr>
<td>A13-14</td>
<td>20-31</td>
<td>8</td>
<td>1 8 16</td>
<td>11K-14K</td>
</tr>
<tr>
<td>B0</td>
<td>18-22</td>
<td>4</td>
<td>1 16 8</td>
<td>44K</td>
</tr>
<tr>
<td>B1-4</td>
<td>20-17</td>
<td>4</td>
<td>1 16 8</td>
<td>159K-192K</td>
</tr>
<tr>
<td>B5-6</td>
<td>16-48</td>
<td>4</td>
<td>1 16 8</td>
<td>44K-50K</td>
</tr>
<tr>
<td>B7</td>
<td>19-06</td>
<td>8</td>
<td>2 16 8</td>
<td>20K</td>
</tr>
<tr>
<td>B8</td>
<td>18-03</td>
<td>4</td>
<td>1 16 8</td>
<td>43K</td>
</tr>
<tr>
<td>B9-12</td>
<td>19-48</td>
<td>8</td>
<td>1 16 8</td>
<td>42K-65K</td>
</tr>
<tr>
<td>B13-14</td>
<td>20-08</td>
<td>4</td>
<td>1 16 8</td>
<td>44K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Date (yy-mm)</th>
<th>Chip Density (Gbit)</th>
<th>Organization</th>
<th>HC\textsubscript{first}†</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0-3</td>
<td>16-48</td>
<td>4</td>
<td>1 16 x8</td>
<td>137K-194K</td>
</tr>
<tr>
<td>C4-6</td>
<td>17-12</td>
<td>8</td>
<td>1 16 x8</td>
<td>130K-150K</td>
</tr>
<tr>
<td>C7-8</td>
<td>20-31</td>
<td>8</td>
<td>1 8 x16</td>
<td>40K-44K</td>
</tr>
<tr>
<td>C9-11</td>
<td>20-31</td>
<td>8</td>
<td>1 8 x16</td>
<td>42K-53K</td>
</tr>
<tr>
<td>C12-14</td>
<td>20-46</td>
<td>16</td>
<td>1 8 x16</td>
<td>6K-7K</td>
</tr>
</tbody>
</table>

Table 1 in our paper provides more information about the analyzed modules
Key Observations: Vendor A

Refresh Types:
- Regular Refresh (RR)
- TRR-capable Refresh (TREF₁ and TREF₂)

Observation: TRR tracks potentially aggressor rows using a Counter Table

TREF₁: Refreshes the victims of row ID with the largest counter value

TREF₂: Refreshes the victims of row ID that TREF₂ pointer refers to
Circumventing Vendor A’s TRR

**Approach:** Ensure an aggressor row is discarded from the Counter Table prior to a REF command.

**Counter Table**

<table>
<thead>
<tr>
<th>row ID</th>
<th>counter value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16 entries

**RR** $\text{TREF}_1 \text{TREF}_2$

REF $\rightarrow$ ACT $([A_1, A_2])$ $\rightarrow$ ACT($D_1$) $\rightarrow$ ACT($D_2$) $\rightarrow$ \ldots $\rightarrow$ ACT($D_{16}$) $\rightarrow$ REF

$N$ times $N+1$ times $N+1$ times $\ldots$ $N+1$ times

$A_i$: aggressor row

$D_i$: dummy row

This RowHammer access pattern requires synchronizing accesses with REF commands

[Circumventing Vendor A’s TRR by **discarding** the actual aggressor rows from the Counter Table]
Key Observations: Vendor B

Refresh Types:
- Regular Refresh (RR)
- TRR-capable Refresh (TREF)

**Observation 1:** TRR *probabilistically* samples the address of an activated row

**Observation 2:** A newly-sampled row overwrites the previously-sampled one

**TREF:** Refreshes the victims of the **last sampled row**
Approach: Maximize the dummy row hammers after hammering the aggressor rows and before the next TREF.

Circumventing Vendor B’s TRR by making it replace a sampled aggressor row by sampling a dummy row.
Key Observations: Vendor C

Refresh Types:
• Regular Refresh (RR)
• TRR-capable Refresh (TREF)

Observation 1: TRR detects an aggressor row only among the first 2K ACT commands issued after a TREF

Observation 2: Rows activated earlier within the 2K ACT commands are more likely to be detected by TRR

TREF: Detects an aggressor row only among the first 2K ACT commands while favoring the earlier activations more
Circumventing Vendor C’s TRR

**Approach:** Hammer dummy rows before aggressor rows to maximize the probability of TRR detecting a dummy row

\[
\begin{align*}
&\text{TREF} \rightarrow \text{ACT}(D_1) \quad N \text{ times} \\
&\quad \rightarrow \text{ACT}([A_1, A_2]) \quad M \text{ times} \\
&\quad \rightarrow \text{TREF} \\
&\quad \quad \quad \quad \quad \quad \text{[A1, A2] not refreshed by TRR}
\end{align*}
\]

Circumventing Vendor C’s TRR by first hammering dummy rows to make aggressor rows less likely to be detected
We craft new RowHammer access patterns that circumvent TRR of three major DRAM vendors. On the 45 DDR4 modules we test, the new access patterns cause a large number of RowHammer bit flips.
Effect on Individual Rows

All 45 modules we tested are vulnerable to our new RowHammer access patterns.

Our RowHammer access patterns cause bit flips in more than 99.9% of the rows.

Why are some modules less vulnerable?
1) Fundamentally less vulnerable to RowHammer
2) Different TRR mechanisms
3) Unique row organization
Effect on Individual Rows

All 45 modules we tested are vulnerable to our new RowHammer access patterns.

Our RowHammer access patterns cause bit flips in more than 99.9% of the rows.

Our access patterns successfully circumvent the TRR implementations of all three major DRAM vendors.

3) Unique row organization
Can ECC Protect Against Our Access Patterns?

ECC DRAM Module

DATA

ECC METADATA

10-byte codeword

corrects 1 bit/symbol

detects 2 bits/symbols

2-byte ECC symbol

8-byte dataword

Memory Controller

ECC Engine

SAFARI
Bypassing ECC with New RowHammer Patterns

Modules from all three vendors have many 8-byte data chunks with 3 and more (up to 7) RowHammer bit flips.

Conventional DRAM ECC cannot protect against our new RowHammer access patterns.
Other Observations and Results in the Paper

- More observations on the TRRs of the three vendors
- Detailed description of the crafted access patterns
- Hammers per aggressor row sensitivity analysis
- Observations and results for individual modules
- …

<table>
<thead>
<tr>
<th>Module</th>
<th>Date (yy-ww)</th>
<th>Chip Density (Gb/s)</th>
<th>Organization</th>
<th>HC\textsubscript{first}[†]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>19-50</td>
<td>8</td>
<td>1 16 8</td>
<td>16K</td>
</tr>
<tr>
<td>A1-5</td>
<td>19-36</td>
<td>8</td>
<td>1 8 16</td>
<td>13K-15K</td>
</tr>
<tr>
<td>A6-7</td>
<td>19-45</td>
<td>8</td>
<td>1 16 8</td>
<td>13K-15K</td>
</tr>
<tr>
<td>A8-9</td>
<td>20-07</td>
<td>8</td>
<td>1 16 8</td>
<td>12K-14K</td>
</tr>
<tr>
<td>A10-12</td>
<td>19-51</td>
<td>8</td>
<td>1 16 8</td>
<td>12K-13K</td>
</tr>
<tr>
<td>A13-14</td>
<td>20-31</td>
<td>8</td>
<td>1 8 16</td>
<td>11K-14K</td>
</tr>
<tr>
<td>B0</td>
<td>18-22</td>
<td>4</td>
<td>1 16 8</td>
<td>44K</td>
</tr>
<tr>
<td>B1-4</td>
<td>20-17</td>
<td>4</td>
<td>1 16 8</td>
<td>159K-192K</td>
</tr>
<tr>
<td>B5-6</td>
<td>16-48</td>
<td>4</td>
<td>1 16 8</td>
<td>44K-50K</td>
</tr>
<tr>
<td>B7</td>
<td>19-06</td>
<td>8</td>
<td>2 16 8</td>
<td>20K</td>
</tr>
<tr>
<td>B8</td>
<td>18-03</td>
<td>4</td>
<td>1 16 8</td>
<td>43K</td>
</tr>
<tr>
<td>B9-12</td>
<td>19-48</td>
<td>8</td>
<td>1 16 8</td>
<td>42K-65K</td>
</tr>
<tr>
<td>B13-14</td>
<td>20-08</td>
<td>4</td>
<td>1 16 8</td>
<td>11K-14K</td>
</tr>
<tr>
<td>C0-3</td>
<td>16-48</td>
<td>4</td>
<td>1 16 x8</td>
<td>137K-194K</td>
</tr>
<tr>
<td>C4-6</td>
<td>17-12</td>
<td>8</td>
<td>1 16 x8</td>
<td>130K-150K</td>
</tr>
<tr>
<td>C7-8</td>
<td>20-31</td>
<td>8</td>
<td>1 8 x16</td>
<td>40K-44K</td>
</tr>
<tr>
<td>C9-11</td>
<td>20-31</td>
<td>8</td>
<td>1 8 x16</td>
<td>42K-53K</td>
</tr>
<tr>
<td>C12-14</td>
<td>20-46</td>
<td>16</td>
<td>1 8 x16</td>
<td>6K-7K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Aggressor Detection</th>
<th>Aggressor Capacity</th>
<th>Per-Bank TRR</th>
<th>TRR-to-REF Ratio</th>
<th>Neighbors Refreshed</th>
<th>% Vulnerable DRAM Rows[†]</th>
<th>Max. Bit Flips per Row per Hammer[†]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATRR1</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>73.3%</td>
<td>1.16</td>
</tr>
<tr>
<td>ATRR2</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>99.2% - 99.4%</td>
<td>2.32 - 4.73</td>
</tr>
<tr>
<td>ATRR3</td>
<td>Counter-based</td>
<td>16</td>
<td>✓</td>
<td>1/9</td>
<td>4</td>
<td>99.3% - 99.4%</td>
<td>2.12 - 3.86</td>
</tr>
<tr>
<td>BTRR1</td>
<td>Sampling-based</td>
<td>1</td>
<td>×</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>2.13</td>
</tr>
<tr>
<td>BTRR2</td>
<td>Sampling-based</td>
<td>1</td>
<td>×</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>0.06 - 0.11</td>
</tr>
<tr>
<td>BTRR3</td>
<td>Sampling-based</td>
<td>1</td>
<td>×</td>
<td>1/4</td>
<td>2</td>
<td>99.9%</td>
<td>1.85 - 2.03</td>
</tr>
<tr>
<td>CTRR1</td>
<td>Mix Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>1.0% - 23.2%</td>
<td>0.05 - 0.15</td>
<td></td>
</tr>
<tr>
<td>CTRR2</td>
<td>Mix Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>7.8% - 12.0%</td>
<td>0.06 - 0.08</td>
<td></td>
</tr>
<tr>
<td>CTRR3</td>
<td>Mix Unknown</td>
<td>✓</td>
<td>1/17</td>
<td>2</td>
<td>39.8% - 41.8%</td>
<td>9.66 - 14.56</td>
<td></td>
</tr>
</tbody>
</table>

SAFARI
# Conclusion

**Target Row Refresh (TRR):**
a set of *obscure, undocumented, and proprietary* RowHammer mitigation techniques

**We cannot** easily study the *security properties* of TRR

**Is TRR fully secure? How can we validate its security guarantees?**

<table>
<thead>
<tr>
<th>U-TRR</th>
<th>A new methodology that leverages <em>data retention failures</em> to uncover the inner workings of TRR and study its security</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15x Vendor A DDR4 modules</td>
</tr>
<tr>
<td></td>
<td>15x Vendor B DDR4 modules</td>
</tr>
<tr>
<td></td>
<td>15x Vendor C DDR4 modules</td>
</tr>
<tr>
<td></td>
<td>U-TRR</td>
</tr>
<tr>
<td></td>
<td>New RowHammer access patterns</td>
</tr>
</tbody>
</table>

**All 45 modules we test are vulnerable**

- **99.9% of rows** in a DRAM bank experience **at least one RowHammer bit flip**
- **Up to 7 RowHammer bit flips** in an 8-byte dataword, **making ECC ineffective**

**TRR does not provide security** against RowHammer

**U-TRR can facilitate** the development of **new RowHammer attacks** and **more secure RowHammer protection** mechanisms
U-TRR

Uncovering in-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications

Hasan Hassan

Yahya Can Tugrul  Jeremie S. Kim  Victor van der Veen
Kaveh Razavi  Onur Mutlu

ETH Zürich  TOBB ETÜ  Qualcomm

SAFARI