# Cache-Out: Leaking Cache Memory Using Hardware Trojan

Mohammad Nasim Imtaiz Khan, Asmit De and Swaroop Ghosh

***Abstract*— Data leakage is an important security concern in current systems. Existing data leakage prevention techniques assume that the underlying hardware platform is secure and free from tampering. In this work, we present Cache-Out, a class of system attacks involving hardware compromised with a Trojan embedded in the CPU. We assume that a memory Trojan trigger is present in L1 d-cache and gets activated if one particular address of L1 d-cache is hammered with a particular data pattern for a certain number of times. Once the Trojan is triggered, accessing another address delivers payloads such as, read disturb, write disturb, retention failure and information leakage. We mainly exploit the advanced circuit features employed in the peripherals of nanometer cache memories such as, Wordline Underdrive (WLUD) (prevents read disturb) and Negative Bitline (NBL) (assists write) for Static RAM (SRAM) to deliver the payloads. Simulation indicates that WLUD and NBL manipulation can inject read and write failures, respectively. We also show that WLUD activation during write operation can inject write failure. Furthermore, NBL along with column multiplexing can also be leveraged to steal data. We validated Cache-Out using GEM5 architectural simulator. We propose L1 address obfuscation, read/write verification, scrambling Error Correcting Code (ECC) bits and trusted ECC as countermeasures. Results indicate that read/write verification incurs 7.56$\mu$m$^2$ of area and 0.1$\mu$W/91.3$\mu$W of static/dynamic power in 22nm technology for 64-bit word size.**

## I. Introduction

Globalization of semiconductor design and fabrication processes have lead to Integrated Circuits (ICs) becoming increasingly vulnerable to malicious modifications in the form of Hardware Trojans [1]. Ideally, these modifications should be detected during the pre-Silicon verification and the post-Silicon testing. However, adversaries can design Trojans that only activate under rare conditions and remain undetected during the test phase. The Trojans, once activated, perform undesirable operations such as, operation failure or even leak sensitive data. This threat is of special concern to government agencies, military [2], technology developers, finance, and energy sectors. Incidents such as, tampering of server motherboards by Chinese manufacturers that affected top US companies like Amazon, Apple etc. [3] and hardware fabricated with hidden-backdoor to disable radars in Syria [4] serve as a strong motivation to investigate the possibility of hidden components in the design and manufacturing process.

Memory Trojan can lead to read/write/retention failures and information leakage. In prior works, authors have proposed memory Trojan trigger circuits and payloads that can evade testing phase and cause different failures. A Trojan for embedded SRAM is proposed in [5] which uses unique data patterns written to pre-selected address to trigger it. These unique patterns are not tested during standard post-manufacturing tests and thereby, remains undetected. That data pattern feeds the input of the Trojan transistors which short the data node of a victim SRAM cell to ground and corrupts the data. The feasibility of such Trojan is limited since the payloads require connecting the bitcells with the Trojan transistors. This is difficult since bitcells are placed side by side with a tight margin.

Designing a small, controllable and undetectable Trojan is the key element to deploy an efficient one. In [6], a capacitor-based analog Trojan trigger, A2, is presented which is controllable, stealthy and small. In [7], another capacitor-based Trojan trigger is proposed which gets activated if a pre-selected address is written with specific data patterns for a specific number of times. A Trojan trigger similar to [7] has been considered in this work (details in Section II.A). Once triggered, the Trojan delivers payloads to the SRAM cache, such as, a particular L2 cache line. We have considered [7] for trigger (over [6]) since it, i) is robust against process and temperature variation; ii) evades post silicon testing and system level detection mechanisms; and, iii) incurs less area overhead. However, any trigger that remains hidden and gets activated under unique condition can serve our purpose.

Once the Trojan trigger is activated, the adversary can selectively launch read/write failure or information leakage attack. The conventional Static RAM (SRAM) employs read and write assist circuits e.g., wordline underdrive (WLUD) [8] and negative bitline (NBL) [9] in the peripherals to enable functional read/write operations. This is especially true for high density, fast and low power cache memories. In this work, we leverage them to achieve the payloads. The uniqueness of targeting the peripherals over the bitcell (similar to the case of [5]) lies in the fact that the component density in the peripheral area is relaxed.

**Attack Model:** We assume that adversary can be either in the design house or fabrication house to tamper with the memory. We also assume that Trojan triggers such as [7] is already implemented in the L1 d-cache (in the filler areas of the non-memory logic, e.g., address pre-decoding and pipelining units, also called midlogic [10]) and generates a Trojan signal namely, $V_{\text{Tr}}$, when a pre-selected address is accessed $N$ times with a pre-selected data pattern. This $V_{\text{Tr}}$ signal can be leveraged to launch the attacks. As mentioned

*This work is supported by SRC 2847.001, NSF CNS- 1722557.

The authors are with the School of Electrical Engineering and Computer Science, Pennsylvania State University, State College, PA 16802 USA (e-mail: muk392@psu.edu; asmit@psu.edu; szg212@psu.edu)

before, we focus on payloads in this work that can be realized by inserting Trojans in the memory peripherals. After the deployment of the chip in a system, the adversary can launch a malicious program to trigger the Trojan to deliver the desired payloads. We assume that the adversary has standard user privileges in the compromised system, where he can interact with the existing applications running on the system and can also compile and run his own malicious program.

In summary, we, (a) investigate SRAM peripherals and their security implications; (b) propose novel Trojans that can tamper with the read/write assist to launch fault injection and DoS attacks; (c) propose novel Trojans that can leak the data by exploiting the column multiplexing and NBL; (d) demonstrate a data leakage exploit using Cache-Out in GEM5 simulator; (e) propose countermeasures that prevents Trojan triggering and detects the payloads.

The paper is organized as follows: Section II describes the background of the Trojan trigger used in this work along with the background on system architecture; Section III describes the background of SRAM. Section IV presents SRAM read/write assist techniques, their vulnerabilities and ways to manipulate them; Section V demonstrates data leakage by Cache-Out attack using GEM5; Section VI presents a discussion on the practicality of memory Trojans and possible defenses; Finally, Section VII draws the conclusion.

## II. Background

In this section, we describe the background on the Trigger circuit used in this work and the system architecture.

### A. Trigger Design [7]

The trigger circuit (Fig. 1) is designed to be activated if a particular memory address ($Add_{SET}$, chosen during Trojan design) is written with a specific data pattern, $P_{\text{SET}}$ for at least $N_{SET}$ times. The trigger has two inputs namely, $V(Add_{\text{SET}})$ and $V(P_{\text{SET}})$. $V(Add_{\text{SET}})$ (= 1V) is the wordline enable signal of $Add_{SET}$. For $V(P_{\text{SET}})$, a simple logic circuit can be implemented which outputs logic 1 (1V) if a specific data pattern is sent to data bus. For example, let's consider that the data bus width is 8-bits. Assume that four specific data bits are taken to design the trigger logic e.g., data[0], data[3], data[4] and data[6]. The logic circuit outputs

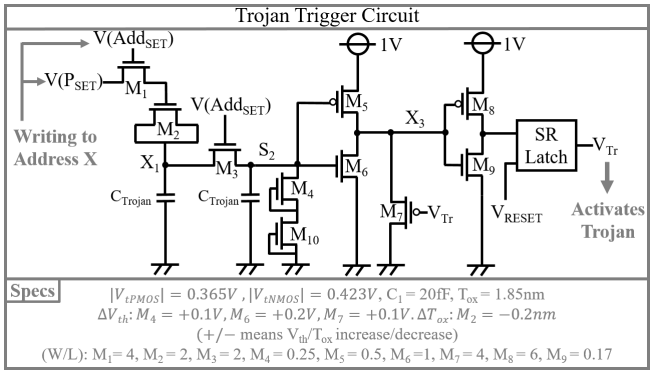



Fig. 1: Capacitor-based Trojan trigger circuit proposed in [7].

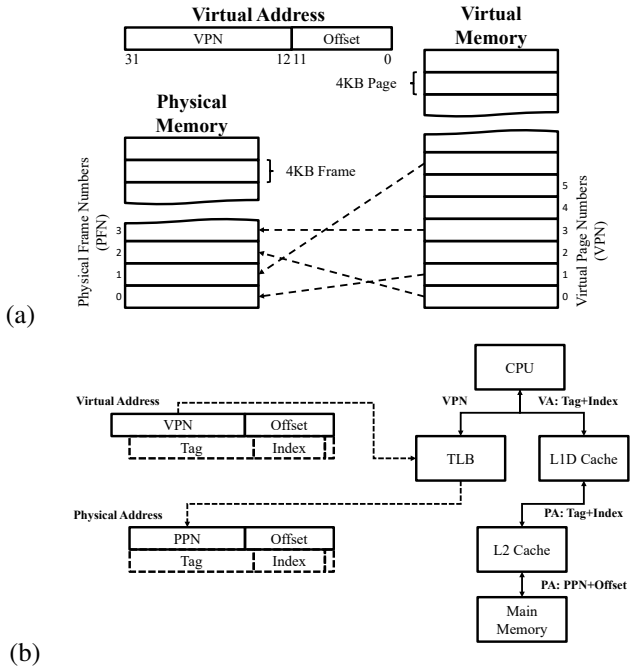



Fig. 2: (a) Paging in a 32-bit address space with 4KB pages; (b) cache hierarchy including TLB showing parallel access.

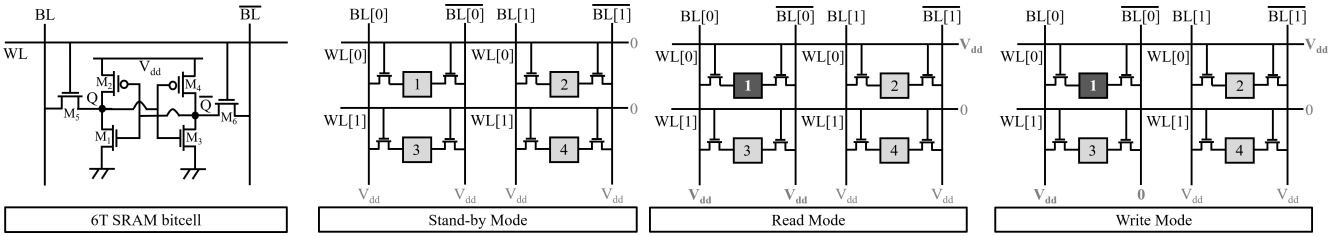
'1' if data[0] = data[3] = data[6] = 1 and data[4] = 0. In practice, data bits with low activation probabilities can be used to lower the overall probability of unintended assertion.

Whenever $Add_{SET}$ is written with $P_{\text{SET}}$ data pattern, MOSFETs $M_1$ and $M_3$ turn ON and $C_{\text{Trojan}}$ charges up from the $V(P_{\text{SET}})$ via Fowler Nordheim (FN) tunneling [11] through $M_2$. Note that $M_2$ has a thinner gate oxide compared to other MOSFETs and its source and drain are shorted (i.e. $M_2$ behaves as a capacitor). $M_4$ is an OFF transistor which offsets the gate leakage of $M_5$ and prevents unwanted charging-up of node $X_2$. $M_7$ keeps node $X_3$ as low as possible until node $X_2$ charges up sufficiently. The node $X_4$, that is charged up during the hammering process, is used as the SET input of an SR latch. SR latch output ($V_{Tr}$) transitions from $0 \rightarrow 1$ when $X_4$ charges up to $\sim$0.5V which requires $N_{SET} = 1837$. The signal $V_{Tr}$ is then used to deploy the RF Trojan payloads.

The work [7] shows that adversary hammers for a duration of $T_{\text{ON}}$ and then stays idle for $T_{\text{OFF}}$ and repeats this cycle. However, the circuit can still be triggered (minimum $T_{\text{ON}}\%$ = 30) but with a higher $N_{SET}$ since $C_{\text{Trojan}}$ does not significantly leak during the OFF cycle. Therefore, it becomes even harder to prevent this Trojan activation using system level techniques such as limiting consecutive accesses to one particular address up to a threshold.

$V_{RESET}$ can be generated to deactivate the trigger by writing to $Add_{RESET}$ for at least $N_{RESET}$ times with a specific data pattern, $P_{\text{RESET}}$ and using a circuit similar to the trigger one. A smaller $C_{Trojan}$ ($\sim$1fF) can be used in the RESET circuit to minimize the area ($N_{\text{RESET}} = 92$). However, the AND'ed output of $V(Add_{RESET})$ and $V(P_{RESET})$ can also serve as $V_{RESET}$.

The trigger is robust and evades testing under worst-case




(a) (b) (c) (d)

Fig. 3: SRAM (a) bitcell, (b) stand-by mode, (c) read mode, (d) write mode.

process and temperature variation [7]. Trigger area/static power are both $< 0.0001\%$ of that of a typical memory [12].

### B. System Architecture Overview

We consider a X86 microprocessor architecture with L1 (separate instruction and data) and L2 caches, where the L1 cache is virtually indexed and physically tagged to improve performance, and the L2 cache is physically indexed. The L1 d-cache is direct-mapped to improve hit times, while the unified L2 cache is set-associative to reduce miss rates. The system runs a standard Linux kernel with paging enabled and a flat linear address space (transparent segmentation).

**Memory Paging:** Applications running on a specific CPU architecture have their complete view of the memory (known as the virtual address space) addressable by the CPU. The physical memory (and the physical address space) is limited, and so the virtual address space is divided into smaller chunks known as virtual pages (Fig. 2a) that are mapped to the physical memory as physical frames. The corresponding mapping information (virtual page $\rightarrow$ physical frame) is stored in a per-process page table in the OS kernel, and also cached onto a hardware TLB in the CPU.

**Cache Memory:** Modern CPUs are equipped with multi-level caches to improve memory access latency. The cache hierarchy is shown in Fig. 2b. Each CPU core has two levels of cache memory, the L1 and L2 cache. In multi-core CPUs, a L3 cache is also present, and is shared between all the cores. The L1 cache is closest to the core and is typically small to enable fast access. The L1 cache is also split into instruction and data caches. Each cache line is mapped to a portion to the memory. The lower order bits of the address is used to index the cache line, while the higher order bits are used as the cache tag. A tag match for a particular index signifies a cache hit. For the L1 cache, this mapping is done with the virtual memory instead of the physical memory. This allows the system to quickly look-up the data without the need for a virtual to physical address translation (which increases the access time). At the same time, the virtual address is also sent to the paging unit for address translation. If the data is found in the L1 d-cache, the lookup is complete. However, if there is a L1 miss, L2 cache is queried. L2 cache is physically indexed, and is larger than L1. L2 is also mapped set-associative with the physical memory, so that data is infrequently replaced in the cache. While the L1 cache is being queried, the address translation is done by the TLB. For a L1 miss, the translated physical address is used to query the L2 cache, and thereafter the lower memory hierarchy. We implement the Trojan trigger in the L1 d-cache. This is to provide the adversary with higher control of the trigger address, since the adversary can compute the virtual addresses that can direct-map to the L1 d-cache line where the trigger is implemented. We target the L2 cache to deploy the Trojan payload because the data is allowed to persist in the L2 longer than L1. The Trojan payload can also be deployed in the L1 cache, however, this can reduce the possibility for a successful data leak exploit, since the data in the L1 is likely to be replaced frequently. In systems with L3 caches, the payload can also be deployed in the L3. This has the benefit of increasing the window of opportunity for a successful exploit. However, in multi-core systems, L3 is shared between the cores. Hence, a payload in L3 can inadvertently corrupt data for processes sharing that data.

## III. SRAM Architecture and Design Trade-Off

In this section, we present the basics of SRAM, SRAM architecture and write-ability/read-ability trade-off of SRAM.

### A. Basics of SRAM

Fig. 3a shows the schematic of 6T-SRAM bitcell. During read operation, Bitline ($BL$) and $\overline{BL}$ are precharged to $V_{\text{dd}}$ and Wordline ($WL$) is asserted. If the cell stores data 1 (0), $BL$ voltage stays precharged (discharge) and $\overline{BL}$ discharges (stay precharged). Sense amp outputs the data based on the voltage difference of $BL$ and $\overline{BL}$. During write operation, $BL$ ($\overline{BL}$) is driven to $V_{\text{dd}}$ (0) for writing data 1 (0). Next, $WL$ is asserted and the node $Q$ charges (discharges) to $V_{\text{dd}}$ (0). The node holds the data since the back to back connected inverters ($M_1$-$M_2$ and $M_3$-$M_4$) ensures data stability.

### B. SRAM Architecture

Conventionally, every global column of SRAM consists of several local columns that share a single sense amp and write driver (Fig. 4). Each local column has several bitcells (one in every row). However, only one bitcell from each global column can be written or read at a time. SRAM has three modes of operation:

1. Stand-by: In this mode, all the bitlines of same global column are precharged to $V_{\text{dd}}$ and and all wordlines are driven to ground (Fig. 3b) to hold their data.

2. Read: In this mode, all bitlines are precharged to $V_{\text{dd}}$. However, the cell to be read is selected through asserting the corresponding wordline and read column select. In Fig. 3c,

TABLE I: SRAM Cell Design Metrics

| Feature | $M_5/M_6$ | $M_1/M_3$ | $M_2/M_4$ |
|---|---|---|---|
| Write-ability | ↑ | ↓ | ↓ |
| Read Stability | ↓ | ↑ | ↑ |
| Access Time | ↑ | ↑ | - |
| Hold-ability | - | Balance pull up/down | |
| Leakage | ↓ | ↓ | ↓ |
| Area | ↓ | ↓ | ↓ |

$WL[0]$ is asserted and $BL[0]$ and $\overline{BL}$ are connected to sense amp (through read column select) in order to read Cell 1.

3. Write: In this mode, all the bitlines of same global column are still precharged to $V_{\text{dd}}$. The $\overline{BL}$ ($BL[0]$) of the Cell 1 is discharged to zero via write column select to write 1 (0) (Fig. 3d). The wordline $WL[0]$ is also asserted during the write operation. Note that in this case Cell 2 is in pseudo-read and can be disturbed.

### C. Write-ability and Read-ability Trade-Off

For better write-ability, the access transistors $M_5/M_6$ should be stronger while the pull up transistors $M_2/M_4$ and pull down transistors $M_1/M_3$ should be weaker. For better read-ability, its the other way around. Balanced pull up/down transistors are needed for hold stability. Table I summarizes the trade-off for different conditions. ↑ and ↓ means stronger and weaker transistor respectively.

SRAM design becomes difficult considering the above-mentioned trade-off. Furthermore, increased process variation at scaled technologies and the need for low voltage operation (to reduce power) impose further challenges. For example, threshold voltage of the SRAM transistor can incur 30mV variation per $\sigma$ [14]. Therefore, different read/write assist techniques such as, WLUD [8], negative bit-line [9] are implemented in the state-of-the-art SRAM. Furthermore, power gating is implemented to reduce the leakage power in the stand-by mode. In the next section, we explain these advanced techniques and describe impact of adversarial manipulation to cause read/write fault and information leakage.

We have used 22nm PTM technology [15] for SRAM circuit simulation with 0.8V of supply voltage.

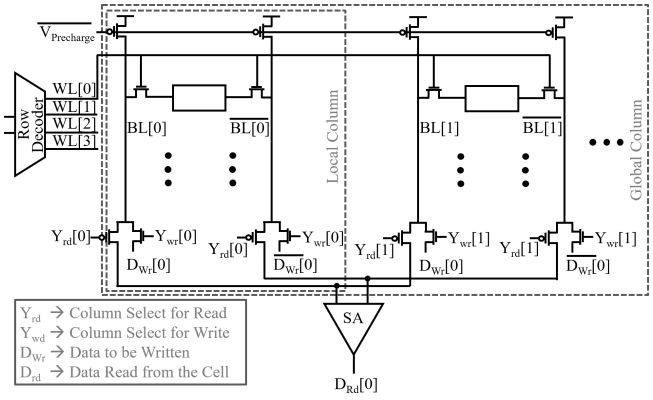



Fig. 4: SRAM architecture for read and write operation showing global and local columns.

## IV. SRAM Assist and Attack Models

In this section, we present SRAM advanced assist techniques and methods to exploit their vulnerabilities

### A. WLUD (Prevents Read Disturb)

Node Q or $\overline{Q}$, whichever is at 0V, incurs a disturb during a read operation. If the disturb voltage crosses the trip point of the other inverter, the data can flip. $WL$ voltage is underdriven to $V_{\text{dd}} - \Delta$ instead of $V_{\text{dd}}$ (Fig. 3c) to reduce the disturb voltage. Fig. 5a shows two techniques proposed in prior work on WLUD [13], [8].

**Inducing Read Disturb:** Adversary can manipulate WLUD in several ways. For example, a PMOS, $M_3$ with higher $(W/L)$ compared to $M_2$ can be inserted in parallel with $M_2$ (Fig. 5b) for circuit $T_1$ [13] of Fig. 5a. If both PMOS ($M_2/M_3$) are asserted at the same time, $WL$ voltage becomes higher than $V_{\text{dd}} - \Delta$. Similarly, a PMOS $M_5$ shown in Fig. 5b can fight with $M_4$ to increase the $WL$ voltage. Note that these malicious transistors are only asserted once the Trojan gets activated and generate $V_{\text{Tr}}$ (Fig. 1).

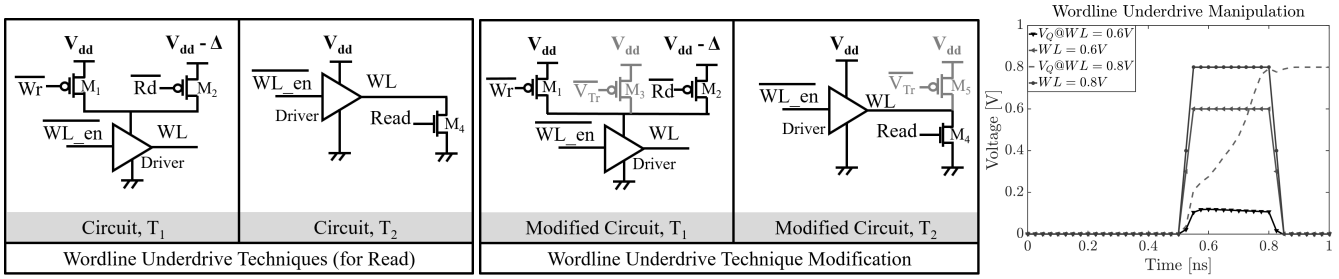
A higher wordline voltage can cause read disturb to the weaker bits. This means that the target cell needs to be weak (due to process variation) for a successful read disturb. Simulation results indicate that if the wordline is driven to 0.8V instead of 0.6V during read operation, the cells with more than $5\sigma$ variation incurs read disturb (corresponding error rate = $5.73 \times 10^{-5}\%$). Adversary can intentionally implement low/high threshold voltage (LVT/HVT) to both access transistor/pull down network of chosen SRAM cells to make them vulnerable. Simulation results indicate that adversary can increase (decrease) threshold voltage of pull down (access) transistors by 150mV and disable the underdrive technique to create a definitive read disturb. Fig. 5c shows the simulation result. Note that these chosen cells do not incur read disturb with WLUD (0.6V) indicating that this kind of fault injection evades testing. We call this as Wordline Overdrive (WLOV) Trojan.

### B. Negative Bitline (Assists Write)

$BL$ ($\overline{BL}$) can be driven to negative voltage instead of 0V for writing data '0' ('1'). This improves access transistor strength (more $V_{GS}$) for reliable write (Fig. 6a).

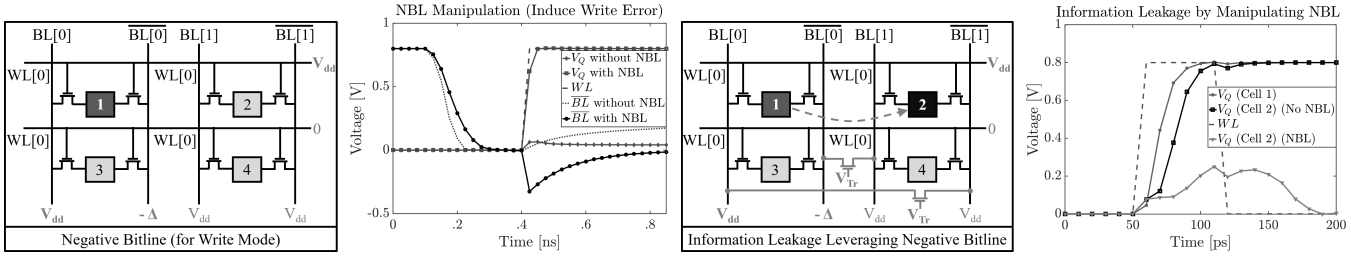
**Inducing Write Error:** Adversary can disable NBL that can cause write errors. The strong bits (intentionally designed or process variation created) that cannot be written without NBL fails. Fig. 6b shows the corresponding result. We call this as NBLW Trojan.

**Information Leakage:** The idea is depicted in Fig. 6c. It is assumed that victim and adversary have control over Cell 1 and Cell 2, respectively. The corresponding $BL$s and $\overline{BL}$s of this two cells are coupled through Trojan transistors (switch). If the switch is activated (by $V_{\text{Tr}}$ from Trojan trigger), the data will be copied to Cell 2 whenever the victim writes to Cell 1. The adversary can read Cell 2 to leak victims write data. Fig. 6d shows that the information is copied from Cell 1 to Cell 2 if NBL is implemented. Information leakage will fail without NBL since the voltage of node Q goes back to its




(a) (b) (c)

Fig. 5: WLUD, (a) implementation circuits $T_1$ [13] and $T_2$ [8]; (b) circuit $T_1$ and $T_2$ modified by Trojan insertion; and, (c) induced read error by manipulating the WLUD circuit.




(a) (b) (c) (d)

Fig. 6: (a) NBL circuit [9]; (b) NBL manipulation to induce write error; (c) & (d) Trojan insertion to leak information from Cell 1 to Cell 2 by exploiting column multiplexing and leveraging NBL mechanism.

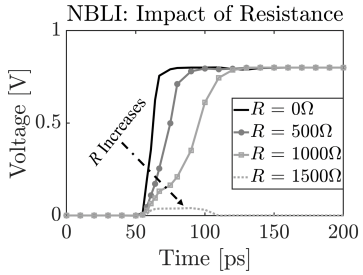
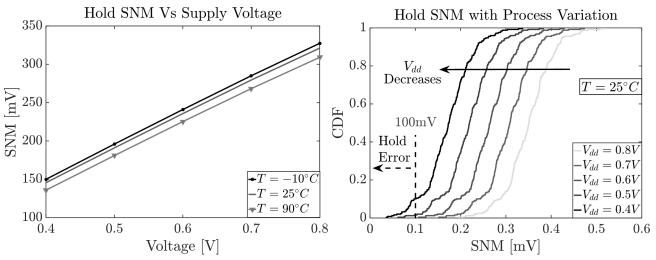



Fig. 7: Impact of interconnect resistance between two addresses that are being coupled for NBLI Trojan deployment.




(a) (b)

Fig. 8: (a) Hold SNM decreases as sleep voltage is lowered (to save power); (b) hold error rate increases considering process variation at lower supply voltage.

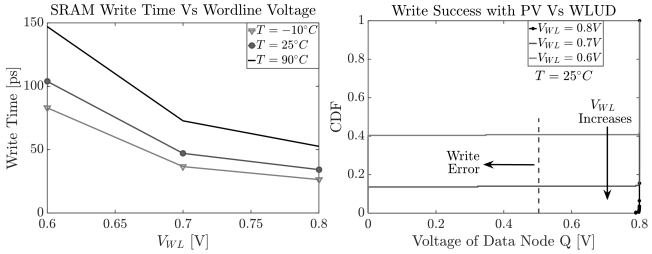
previous state. This is true since NBL gives extra advantage which ends up writing Cell 2. We call this as NBLI Trojan.

Note that $BL$ precharge transistors should be disabled during the data leakage in order to prevent any contention. That can be done by inserting a PMOS transistor between the precharge and the $V_{\text{dd}}$. Another important point is that Cell 1 and Cell 2 in this example does not need to be located side by side. In order to investigate that, we coupled the $BL$s and $\overline{BL}$s with a pass transistor (for full swing) and considered a series resistance to model the metal resistance, $R$ to connect both $BL$s/$\overline{BL}$s. The result is shown in Fig. 7. A high resistance leads to a high voltage drop across it and therefore, Cell 2 does not get enough voltage headroom to get written. We conclude that R can be up to 1000Ω for NBLI to leak the cache data. However, adversary can also leverage upper metal layers to keep R within the limit.

### C. Power Gating (Saves Power)

SRAM is power gated during stand-by mode. Various settings are employed to sleep the cache depending on wake-up requirement. For example, C1 sleep lowers the SRAM voltage mildly in anticipation of quick wake-up whereas C6 lowers the SRAM voltage to maximum possible value during hibernation. Both PMOS (header) and NMOS (footer) based sleep have been proposed to cut down $V_{dd}$ and $GND$, respectively. A binary sized (e.g., W, 2W, 4W etc.) parallel PMOS/NMOS bank is used as header/footer to control the sleep voltage. For maximum sleep all transistors in the bank could be turned OFF. Note that, the sleep setting is optimized after post-Silicon test and therefore, maximum/large sleep settings may not be used to ensure data retention. However, the Trojan can enable maximum sleep to launch DoS attack. We call this as Retention Trojan.

Fig. 8a shows that the hold Static Noise Margin (SNM) reduces at scaled sleep voltages (due to power gating). SNM also decreases with higher temperature. This indicates that the adversary can corrupt the bits by applying excessive power gating (i.e, lower sleep voltage). We have performed a 1000 point Monte-Carlo analysis on SRAM with mean threshold voltage for PMOS = -0.4606V and NMOS = 0.5031V with 120mV ($4\sigma$) absolute variation at $T = 25°C$.




(a) (b)

Fig. 9: (a) SRAM write time increases as $WL$ voltage is reduced; (b) write error rate increases considering process variation at various $WL$ voltages.

The result is shown in Fig. 8b. We note that no error is incurred if the sleep voltage is scaled down to 0.6V from 0.8V. However, if the adversary tweaks the sleep setting to reduce the sleep voltage to 0.4V, about $10\%$ of the cells may loose their data (considering min SNM = 100mV at $T = 25°C$). Adversary can also implement similar technique, $T_2$ shown in Fig. 5a to further reduce sleep voltage.

### D. Enabling WLUD During Write

Adversary can enable WLUD during write operation by implementing a Trojan NMOS controlled by $V_{Tr}$ in parallel with transistor $M_4$. In this case, the access transistor will get lower gate to source voltage even with NBL (same as without NBL) leading to a write failure. Fig. 9a shows that SRAM write time (i.e., time to switch the node voltage after $WL$ assertion) increases as $WL$ voltage is lowered. The figure also shows the temperature dependency. Next, we performed a 1000 point Monte-Carlo analysis on SRAM write operation with the same setup as before for varying $WL$ voltages at $T = 25°C$. The result is shown in Fig. 9b. We conclude that if the $WL$ voltage is reduced to 0.7V and 0.6, the corresponding write error increases to 14% and 40.8%, respectively. We call this as WLUD Trojan.

### E. Area and Power Analysis of the Payloads

Table II summarizes the area/power overheads for 1-bit Cache Trojans along with their use case. However, we didn't consider the parasitics of interconnects since they only affect the delay. Note that all the Trojans are implemented in the peripheral area where there are enough empty space. Furthermore, accurate measurement and estimation of the power profile is the key to detect attacks [16]. Techniques such as [17] can provide accurate characterization of power consumption in digital systems. However, static power consumed by the Trojans (as shown in Table II) is so little that it could be overshadowed by process variation and thermal noise. Therefore, we can conclude that the static power overheads of the Trojans is negligible. The dynamic power of some cases is high. However, they will only occur one time (when Trojan gets activated). The area/static power are both $< 0.0001\%$ of that of a typical memory [12].

TABLE II: Features of the Proposed 1-bit Cache Trojans

| Trojan | Static Power (nW) | Dynamic Power ($\mu$W) | Area ($\mu m^2$) | Use Case |
|---|---|---|---|---|
| **WLOV** | 0.0024 | 1.1 | 0.0027 | DoS |
| **NBLW** | 0.001 | 0.34 | 0.0036 | DoS |
| **NBLI** | 0.004 | 1.3 | 0.0144 | Data Leak |
| **Retention** | 0.002 | 0.28 | 0.048 | DoS |
| **WLUD** | 0.0024 | 1.1 | 0.0027 | DoS |

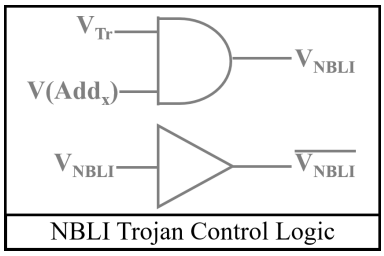



Fig. 10: Control signal for NBLI Trojan.

### F. Delay Analysis of the Trigger and Payloads

The Trojan trigger circuit or the payloads will add delays. For example, the gate of $M_1$ and $M_3$ transistors are connected with the wordline enable signal of $Add_{SET}$. Therefore, that address will see an increased delay. However, note that the wordline of $Add_{SET}$ is driving 512 access transistors (for a 512bit cache line). Driving two more small size transistors increases its load by 0.4%. Due to process variation, this delay addition by the Trojan trigger can not be detected. Note that the adversary can add two minimal size back to back inverters between the wordline enable signal of $Add_{SET}$ and the gate of $M_1$ and $M_3$ transistors. This will lower the delay addition even further.
Similarly, the payload delays are also minimal since they are implemented using 1 or 2 basic gates or transistors. Furthermore, they are only turned ON when the Trojan is activated. Therefore, during the post-Silicon hardware Trojan detection, the Trojan needs to be activated which is not feasible. In summary, the delay addition by the Trojan trigger or the payloads are insignificant and may not be leveraged to detect Trojan during post-Silicon testing.

## V. Information Leakage Attack Demonstration

In this section, we describe and demonstrate an attack on L2 cache to leak information using NBLI Trojan.

### A. Attack Description

The example of information leakage attack in Fig. 6c uses $V_{Tr}$ as control signal which is always ON once the Trojan trigger of Fig. 1 gets activated. However, in practice, the shorting should be enabled only when the victim address is getting written after the trigger is activated. Fig. 10 shows the required control logic. Adversary hammers $Add_{SET}$ of L1 d-cache with $P_{SET}$ data pattern $N_{SET}$ times to activate the Trojan trigger of Fig. 1 (i.e. $V_{Tr}$ = 1). Next, when there is a context switch and a write operation is performed by the victim process to $Add_x$ (i.e. $V(Add_x)$ = 1), $V_{NBLI}$ becomes 1 and $\overline{V_{NBLI}}$ becomes 0. Therefore, the pass transistor which shorts the $BL$s/$\overline{BL}$s of $Add_x$ and $Add_y$ (two cache lines in L2), gets turned ON. This copies the data to $Add_y$. The attack is demonstrated in the next subsection.

The area/static power/dynamic power of the control logic are 0.0099$\mu$m$^2$/0.046nW/0.6$\mu$W. For a 64bit data leakage, the total area/static power/dynamic power are 0.93$\mu$m$^2$/0.302nW/83.8$\mu$W which includes overheads for the control logic along with the data copy. The dynamic power is high since it effectively writes another cache line. However, this will incur only after the trigger is activated.

### B. Attack Demonstration

We describe a simple proof-of-concept attack. The goal is to gain access to data from process's address space which would typically not be accessible to an adversary. Let us consider a single victim process which accesses a certain memory location $T$ that is initialized (written) with a specific bit pattern, and then repeatedly accessed. The number of accesses to the location is controlled by a user input $N$. The memory location $T$ is mapped to the L1 d-cache line $Add_{SET}$. Let us also assume that the process stores some 'secret' data in memory location $S$, which is mapped to the L2 cache line $Add_x$. This 'secret' data is not available for the user to view in its raw form. This can happen for example, in a program which performs encryption/decryption in software by using some cryptographic keys. The keys will not be available to the user, but it may still be loaded into the process's memory. The goal of the adversary posing as the user is to obtain the secret cryptographic key, using some existing input/output operation in the program. The process also reads back and prints data from location $P$ to the user. Address $P$ is mapped to the L2 cache line $Add_y$. The adversary interacts with the victim program by controlling $N$, and forces it to print the secret data using the code which normally prints from location $P$.

The malicious Cache-Out Trojan trigger is placed in the L1 d-cache line $Add_{SET}$, that monitors its access for $N_{SET}$ times. This action activates the trigger which generates a signal for the Trojan payload. The payload deployment circuit is placed in the L2 cache. When the payload is deployed, the cache lines $Add_x$ and $Add_y$ are shorted. Thereafter, any writes to $Add_x$ maliciously copies the data to $Add_y$.

The example program is shown in Listing 1. The adversary initially provides `n` ($N$) as 2000, which is larger than $N_{SET}$ ( = 1837). This makes the process hit the L1 d-cache line $Add_{SET}$ mapped to `addr_t` (`0x602010`) for at least $N_{SET}$ times, which serves as the trigger for the Trojan. The activated Trojan shorts the addresses `addr_s` (`0x602030`, mapped to L2 cache line $Add_x$) and `addr_p` (`0x602050`, mapped to L2 cache line $Add_y$). In our example program, the process writes to `addr_s` ten integers from 1 to 10 in a loop. The process prints data from `addr_p` and gains access to the data from `addr_s`. Fig. 11 shows the program output.

In this example, the victim program coincidentally happened to work in favor of the Trojan, which may not be the case in reality. However, the adversary can use the same attack methodology to write his own malicious program with the knowledge of the address ($T$) which the Trojan uses as a trigger, the number of accesses required to trigger the Trojan ($N$), and the cache line mappings of secret data $S$ and public data $P$. Here, $S$ is in a page frame of the victim process, while $P$ and $T$ are in the adversary's exploit program.

```c
void main() {
    int *addr_t, *addr_s, *addr_p;
    int i, data;

    /* N: Number of accesses to trigger address,
     ↪  minimum 1837 */
    int n = 2000;

    /* T: Address of trigger; value initialized */
    addr_t = (int*)malloc(sizeof(int));
    *addr_t = 10;

    /* S: Address of secret data */
    addr_s = (int*)malloc(sizeof(int));

    /* P: Address of public data */
    addr_p = (int*)malloc(sizeof(int));
    *addr_p = 0;

    /* Before Trojan activation, P points to public
     ↪  data */
    printf("## Before Trojan is activated ##\n");
    printf("addr_t = %p\tvalue = %d\n", addr_t,
     ↪  *addr_t);
    printf("addr_s = %p\tvalue = %d\n", addr_s,
     ↪  *addr_s);
    printf("addr_p = %p\tvalue = %d\n", addr_p,
     ↪  *addr_p);

    /* Trigger the Trojan by N accesses to T */
    for (i = 0; i < n; i++) {
        data = *addr_t;
    }

    /* After Trojan activation, write to S and read
     ↪  from P */
    printf("## Trojan activated ##\n");

    for (i = 1; i <= 10; i++) {
        *addr_s = i;
        printf("addr_p = %p\tvalue = %d\n", addr_p,
         ↪  *addr_p);
    }
}
```

Listing 1: Example attack code using the Cache-Out Trojan

We modeled the effects of Cache-Out Trojan in GEM5 [18]. The system is designed with an AtomicSimpleCPU model running a single core at 1GHz emulating a X86 architecture, with L1 and L2 caches, and connected to a 512MB DDR3 physical memory. L1 I-Cache is 32KB and L1 D-Cache is 64KB, while L2 cache is 2MB. Each cache line is 64bit wide. The MMU is configured with a 64 entry TLB. We modified the GEM5 cache source code to implement the address monitoring (trigger) in L1 d-cache, and address shorting of two cache lines in L2 cache (payload). The attack code in Listing 1 is compiled with `gnu-gcc`. The attack is evaluated by running the compiled binary in the GEM5 system$WL$. As seen in Fig. 11, we were successfully able to read the data written to `addr_s` by accessing `addr_p`.

## VI. Discussion

In this section, we present discussion on the practicality and scope of memory Trojans and their detection.

```
## Before Trojan is activated ##
addr_t = 0x602010       value = 10
addr_s = 0x602030       value = 0
addr_p = 0x602050       value = 0
## Trojan activated ##
addr_p = 0x602050       value = 1
addr_p = 0x602050       value = 2
addr_p = 0x602050       value = 3
addr_p = 0x602050       value = 4
addr_p = 0x602050       value = 5
addr_p = 0x602050       value = 6
addr_p = 0x602050       value = 7
addr_p = 0x602050       value = 8
addr_p = 0x602050       value = 9
addr_p = 0x602050       value = 10
```

Fig. 11: GEM5 console output for the attack code in Listing 1. Note that $P$ reads data of $S$ after Trojan activation.

### A. Scope of Attacks

**Trigger and Payload Locations:** The proposed Cache-Out Trojan is most effective when the trigger is in L1 and payload is in LLC. The trigger in L1 makes it easier to activate it, since if it is repeatedly accessed, it is likely to persist in L1 and cause higher read/write hits. The payload in LLC ensures that the data is likely to get frequently evicted, since LLCs typically have lower miss rates than higher cache levels. This increases the chances of the pages in the adversary's exploit program and the victim program to simultaneously reside in the cache, providing a higher attach success. Cache-Out trigger/payload can also be placed hierarchically closer in the cache (easier implementation) with the trade-off being lower attack success rate.

**Payload Coverage:** In a real multi-process system, the data in the caches are continuously updated and replaced. To circumvent the issue of data replacement, the adversary may have to run his exploit multiple times to repeatedly trigger the Trojan and try to read from his controlled cache line. Furthermore, the adversary cannot guarantee that the data in the victim process will use the Trojan deployed cache line. In this case, the Trojan payload can be sprayed in multiple LLC cache lines, where each cache line shorting can be activated by a different trigger address in the L1 d-cache. Each Trojan payload can cover the entire set of addresses in the main memory that map to the same victim cache line in the LLC. The shorting of the cache lines can be between any addresses in the adversary's page and the victim's page. The implementation is easier if the physical location of the two cache lines are closer, since it reduces the interconnect distance. This is practical, since, in a SRAM layout, two wordlines in neighboring banks may have logically distant addresses irrespective of their physically location.

**Trojan Use Cases:** The Trojan can be deployed to launch generic data-leakage exploits, or it can be tailored to a specific victim program. The choice of the adversary's cache lines can be easily determined, since the exploit program is written by the adversary. The adversary can easily profile his exploit program in a cache simulator to determine the probable cache lines his controlled addresses can map to. For a generic exploit, the adversary has to randomly place the payloads spread throughout the LLC. The adversary can then, depending on the victim program, choose the trigger that activates the payload that is most suitable for the program based on its cache access pattern. For a more tailored exploit, the adversary can profile the victim program and study its cache access patterns. This will provide the adversary a more accurate probable cache location for targeting the payload for that program. The Trojan can then be designed to be placed in those cache lines which are frequently used by the application. An example of such attack may be the SSH daemon which reads the user's private key to perform SSH authentication. The adversary can study the cache access patterns of the daemon process and design the Cache-Out Trojan payload based on the LLC access patterns of the SSH daemon. The payload will effectively short the cache line ($S$) that loads the SSH private key to an adversary chosen cache line ($P$). He can then run his exploit code based on the designed Trojan to read the SSH key through his chosen address. A tailored exploit provides simplicity of implementation; however, the adversary needs to have prior access to the 'golden' chip design/RTL.

### B. Fault Injection and DoS Attack

If read/write/retention failure occurs for one polarity (either for data '0' or '1'), it is considered as fault injection [19]. Such attack can leak system assets [20] such as, keys. One example is when an adversary induces single-bit or multi-bit faults in a cryptographic system and performs differential fault analysis by observing correct and faulty pairs of inputs and outputs and subsequently derives simplified equations to extract the keys. Multiple methods for extracting keys using fault injection have been extensively studied and demonstrated [21]. However, if failure occurs for both polarities, it is considered as DoS attack [19].

### C. Evading Test and Detection

**Trigger Detection:** The concealment of the Trojan is dependent on the stealth of the trigger, i.e., the likelihood of the trigger being activated under test or normal usage. Our trigger is activated with repeated access to a specific address with a unique data pattern. In conventional memory testing, each memory location is tested for successful write and readback operation. During March tests, the test pattern consists of a finite sequence of March elements consisting increasing or decreasing address order of read and/or write operations covering all memory cells. This is done to ensure linear complexity of test time based on the size of memory. However, such standard test procedures do not account for multiple repeated write attempts to a memory location, nor does it account for multiple repeated writes of various data at the same address (significantly increases test time). Hence, the trigger is unlikely to get detected under standard tests.

Memories often employ row-hammer detection techniques to prevent data corruption. However, such techniques detect continuous repetitive access (hammering) to the same address. This, however, does not affect our trigger, since it can still be activated without hammering, as long as the adversary accesses the trigger address the required number of times.

**Payload Detection:** Since the payloads short two memory locations on activation, this may reduce the resilience of the cells in those locations after the Trojan is triggered. During testing, this may get detected if some reads and writes fail in those cells. However, this is only possible in the unlikely event that the trigger is activated during testing. Extensive delay/power profiling on the hardware may be able to detect the payload due to the additional overhead, especially if multiple payloads are deployed.

**Bypassing Error Detection:** In state-of-the-art memory, techniques like Cyclic Redundancy Check (CRC) [22] or Error Correcting Code (ECC) [22] is implemented. The ECC and/or CRC word is computed for the raw data and written along with data during write operation. During read operation, CRC/ECC is again calculated based-on the read data and matched with the stored CRC/ECC. If read or even write operation incurs an error, CRCs/ECCs will not match and the data can be discarded. Furthermore, ECC can correct 1/2 bits of error (based on the ECC type). For example, SECDEC ECC can detect double-bit errors and correct single-bit error. Therefore, fault injection will fail and adversary can only launch DoS. However, if the CRC or ECC bits are also tampered to match the data with injected fault, the manipulated data will be considered as valid data.

### *D. Countermeasures*

The proposed countermeasures are described below. Their coverage and associated overheads are presented in Table III.

**L1 Address Obfuscation:** Typically, L1 cache is virtually indexed and physically tagged. However, this is a vulnerability since adversary can hammer L1 cache using virtual address. Therefore, L1 address obfuscation (using a PUF, for example) to change virtual to physical mapping of L1 cache can add a layer of complexity on the adversary. This is true since the predefined memory address can no longer be hammered. Similar technique has been shown to be effective in preventing side-channels in the cache [23].

**Read/Write Followed by Validating Read:** We propose to read each address after they are written or read. This will capture any write failure or read disturb caused by the Trojans. For example, if a fault is injected during write operation of one address, reading that address in subsequent cycle will give a data that does not match with the original write data. Therefore, write fault injection will be detected even if ECC fails to catch it. Similarly, if read disturb (corrupts the original stored data) occurs during read operation, reading it again will output data that will not match with the first read. Therefore, the read fault injection can be detected. Note that, in case of read failure (e.g. sense failure) both read operation may give the same data (if the fault injection affects both reads) and the fault injection may remain undetected. Furthermore, if the adversary targets both read/write operation and verification, fault injection will remain undetected. However, fault injection during both read/write operation will require a significant overhead.

Read/write verification will incur, (i) area overhead due to additional flops (64 flops occupy $7.56\mu m^2$ in 22nm technology, which is $<0.001\%$ of a L2 cache area [12]) for holding the first read/write data; (ii) power overhead due to additional read operation and extra flops $0.1\mu$W/$91.3\mu$W of static/dynamic power for 64 flops); (iii) performance overhead due to extra read operation after every read/write. To minimize the performance loss, the verification can be done at the memory bank level instead of bringing the data to CPU. However, this will prevent back-to-back accesses of a bank. Note that that stretching the cache write operation by 1 cycle in the L2 cache incurs a minimal performance loss of 0.2% (based on SPEC CPU2000 benchmark simulations) [24]. Extending the technique to both L1/L2 caches result in 3.6% performance overhead. Therefore, the expected performance overhead of the proposed validation technique is very small.

TABLE III: Coverage of the Proposed Countermeasures

| Countermeasures | Coverage | Overhead |
|---|---|---|
| **L1 address Obfuscation** | Prevents Trojan Trigger | Area/Perf. |
| **Read/Write Verification** | Fault Injection/DoS | Area/Power/Perf. |
| **Scrambling ECC Bits** | Fault Injection/DoS Information Leakage | Routing Overhead |
| **Trusted ECC** | Fault Injection/DoS Information Leakage | Routing Overhead |

**Scrambling ECC Bits:** Whenever a data is written to an address, ECC circuit calculates the corresponding ECC and stores it along with the write data. Therefore, if adversary shorts the ECC columns in a similar way as data column then data copy will go undetected. However, the column multiplexing for ECC can be scrambled to switch the ECC bit ordering. For example, ECC logical column $n-1$ can be written to physical column $n+m$ and vice versa. More complex logical to physical mapping can be implemented too. If the adversary shorts/tampers the ECC bits blindly without knowing the mapping, ECC for the data will not match and data copy will be detected. This can also detect fault injection/DoS attacks since adversary will not be able to inject faults to ECC bits correctly without knowing the mapping. If one particular address incurs frequent errors, that can be marked corrupted and system can skip using it.

Note that adversary can inject the fault keeping the ECC of original and fault injected data the same (i.e. ECC collision attack). Such probability is $2^{-5}$ with 5 error correcting bits. In that case, scrambling ECC cannot detect the attacks.

**Trusted ECC:** The current implementation of ECC adds a few global columns in the cache. For example, if the data width is 64 bit and ECC needs 5 bits, a total of 69 global columns are implemented in the memory array. This is a vulnerability since the ECC bits can also be tampered. We propose to separate the ECC bits from the data bits and store them in a trusted memory known to be Trojan free through rigorous validation (possible due to small size). This way the fault injection, DoS and information leakage attacks can be detected since ECC bits can be checked to detect the tampering at run-time (once Trojan is activated). However, ECC collision attacks cannot be detected through trusted ECC. Note that Trojan implementation is fairly complex for ECC collision attacks since it needs to sense the data, find another data with the same ECC to replace the original one.

## VII. Conclusions

In this paper, we investigate the advanced circuit features employed in the cache (SRAM) peripherals and show that adversary can manipulate them to launch fault injection/DoS attacks. The adversary can also leverage bitline shorting in order to leak cache data. We demonstrate a data leakage exploit using the Cache-Out Trojan in the GEM5 simulator. Finally, we propose countermeasures to prevent the Trigger and detect the Trojans payload.

## References


[1] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design Test of Computers*, vol. 27, pp. 10–25, Jan 2010.

[2] "Trusted Integrated Circuits (TRUST). [Online]. Available: https://www.darpa.mil/program/trusted-integrated-circuits. Accessed: Oct 28, 2018," 2018.

[3] "The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies. [Online]. Available: https://bloom.bg/2owldii. Accessed: Oct 28, 2018," 2018.

[4] "The Hunt for the Kill Switch. [Online]. Available: https://spectrum.ieee.org/semiconductors/design/the-hunt-for-the-kill-switch. Accessed: Oct 28, 2018," 2008.

[5] T. Hoque, X. Wang, A. Basak, R. Karam, and S. Bhunia, "Hardware Trojan Attacks in Embedded Memory," in *2018 IEEE 36th VLSI Test Symposium (VTS)*, pp. 1–6, April 2018.

[6] K. Yang, M. Hicks, Q. Dong, T. Austin and D. Sylvester, "A2: Analog Malicious Hardware," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 18–37, May 2016.

[7] M. N. Imtiaz Khan, K. Nagarajan, and S. Ghosh, "Hardware trojans in emerging non-volatile memories," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 396–401, March 2019.

[8] E. Karl, Y. Wang, Y. Ng, Z. Guo, F. Hamzaoglu, U. Bhattacharya, K. Zhang, K. Mistry, and M. Bohr, "A 4.6GHz 162Mb SRAM Design in 22nm Tri-Gate CMOS Technology with Integrated Active VMIN-Enhancing Assist Circuitry," in *2012 IEEE International Solid-State Circuits Conference*, pp. 230–232, Feb 2012.

[9] S. Mukhopadhyay, R. M. Rao, J. Kim, and C. Chuang, "SRAM Write-Ability Improvement with Transient Negative Bit-Line Voltage," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, pp. 24–32, Jan 2011.

[10] U. Bhattacharya, W. Yih, F. Hamzaoglu, N. Yong-Gee, W. Liqiong, C. Zhanping; J. Rohlman; I. Young, K. Zhang, "45nm SRAM Technology Development and Technology Lead Vehicle," in *Intel Technology Journal*, vol. 12, pp. 111–120, June 2008.

[11] N. M. Ravindra and J. Zhao, "Fowler-Nordheim Tunneling in Thin $SiO_2$ Films," *Smart Materials and Structures*, 1992.

[12] M. Chang and P. Rosenfeld and S. Lu and B. Jacob , "Technology comparison for large last-level caches: Low-leakage sram, low write-energy stt-ram, and refresh-optimized edram," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 143–154, Feb 2013.

[13] M. Yabuuchi and M. Morimoto and Y. Tsukamoto and S. Tanaka and K. Tanaka and M. Tanaka and K. Nii , "16nm FinFET High-k/Metal-gate 256-kbit 6T SRAM Macros with Wordline Overdriven Assist," in *2014 IEEE International Electron Devices Meeting*, Dec 2014.

[14] S. Ghosh, S. Mukhopadhyay, K. Kim, and K. Roy, "Self-Calibration Technique for Reduction of Hold Failures in Low-power Nano-Scaled SRAM," in *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, (NY, USA), pp. 971–976, ACM, 2006.

[15] "22nm PTM Technology File [Online]. Available: http://ptm.asu.edu/modelcard/lp/22nm˙lp.pm. Accessed: Feb 10, 2019," 2018.

[16] J. Wu, Y. Shi, and M. Choi, "Measurement and evaluation of power analysis attacks on asynchronous s-box," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, pp. 2765–2775, Oct 2012.

[17] V. Konstantakos, K. Kosmatopoulos, S. Nikolaidis, and T. Laopoulos, "Measurement of power consumption in digital systems," *IEEE Transactions on Instrumentation and Measurement*, Oct 2006.

[18] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, Aug. 2011.

[19] M. N. I. Khan and S. Ghosh, "Fault injection attacks on emerging non-volatile memory and countermeasures," in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '18, pp. 10:1–10:8, ACM, 2018.

[20] Bhattacharya, Sarani and Mukhopadhyay, Debdeep, "Formal Fault Analysis of Branch Predictors: Attacking Countermeasures of Asymmetric Key Ciphers," *Journal of Cryptographic Engineering*, vol. 7, pp. 299–310, Nov 2017.

[21] S. Bhasin and D. Mukhopadhyay, "Fault injection attacks: Attack methodologies, injection techniques and protection mechanisms," in *Security, Privacy, and Applied Cryptography Engineering*, Springer, 2016.

[22] G. -H. Asadi and M.B. Tahoori, "Soft Error Mitigation for SRAM-based FPGAs," in *23rd IEEE VLSI Test Symposium (VTS'05)*, pp. 207–212, May 2005.

[23] F. Liu, "Newcache: Secure cache architecture thwarting cache side-channel attacks," *IEEE Micro*, vol. 36, Sep. 2016.

[24] A. Goel, P. Ndai, J. P. Kulkarni, and K. Roy, "Read/access-preferred (reap) sram - architecture-aware bit cell design for improved yield and lower vmin," in *2009 IEEE Custom Integrated Circuits Conference*, pp. 503–506, Sep. 2009.