2022 IEEE 40th International Conference on Computer Design (ICCD) | 978-1-6654-6186-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCD56317.2022.00024

MultiCon: An Efficient Timing-based Side Channel Attack on Shared Memory Multicores

Usman Ali and Omer Khan
{usman.ali, khan}@uconn.edu
University of Connecticut, Storrs, CT, USA

***Abstract*—Hardware sharing in multicore processors enables massive performance improvements but brings security challenges. Adversaries exploit shared hardware for timing-based side-channel attacks and leak secret information. This paper proposes a novel multi-channel timing-based side-channel attack (called MultiCon) that exploits aggregated timing variations on shared cache, network-on-chip (NoC), and memory controller hardware in a multicore processor. Baseline cache, memory controller and NoC hardware contention attacks show sufficient timing variations, but their overall efficacy in the presence of external noise reduces significantly. De-noising schemes, such as repetition codes increase the efficacy but at the cost of slowing down the attack. MultiCon uses aggregate timing variations of multiple shared hardware resources and is less sensitive to external noise. Our evaluation on a Tile-Gx72 processor shows that MultiCon is able to achieve higher efficacy in the presence of external noise, and is >3× faster than baseline single-channel attacks.**

I. Introduction

Modern multicores exploit hardware sharing for co-located applications. The sharing of underlying hardware enables massive performance improvements in virtualized systems but brings security challenges. For example, adversary applications leak security-critical information using timing-based side-channel attacks (SCAs) that exploit the underlying shared hardware. Timing-based SCAs leak secret information based on timing variations that are caused due to the presence or absence of contention on shared hardware. Google [1] recently published a proof-of-concept for timing-based SCAs. These attacks are broadly categorized into spatial and temporal contention attacks, where the adversarial code measures the latency difference between contended and non-contended accesses to the shared hardware resource. The spatial contention attacks exploit timing variations due to spatial contention on a hardware resource, such as a cache (or TLB or memory), where shared data is pinned until explicitly removed. Temporal contention attacks exploit timing variation on shared hardware structures that hold transient shared data, such as network-on-chip and memory controller queues and buffers. Both types of attacks have motivated mitigation schemes, and adversaries are inclined towards new attacks that can bypass existing mitigations.

Prior work shows that the spatial contention attacks [2] [3] [4] have a high timing-variation. For example, a state-of-the-art cache contention attack [3] shows ∼100 cycle timing-variation. This timing variation is sufficient to reliably leak secret information. Contrary, the temporal contention attacks [5] [6] combine low timing-variations of a few cycles [5] with de-noising capabilities (such as repetition codes [7]) to successfully leak secret information. Repetition codes aggregate the timing variations and increase the efficacy of the attack but at the cost of the speed of attack. Both attacks are sensitive to external noise, and their success depends on an adversary's capabilities to de-noise the timing side-channel. Prior work uses repetition codes [5] to de-noise the SCAs from external noise. Repetition codes re-transmit information multiple times and decrease the probability of noise affecting the efficacy of attack. However, repetition codes reduce the speed of attack. Existing attacks [3] [2] [5] [6] target single hardware channel to create timing variations that makes them sensitive to external noise. A unified attack that aggregates timing variations across multiple shared hardware channels to improve the attack efficacy is missing in the literature.

This paper aggregates timing variations on spatial and temporal contention channels and proposes *MultiCon*, a multi-channel timing-based SCA that exploits contention on underlying shared cache, network-on-chip, and memory-controller hardware in a multicore processor. It also uses repetition codes [7] to de-noise the channel. Although timing variations on the cache hardware are sufficient for a reliable timing-based attack, an external noise (e.g., a concurrent user application) targeting the shared cache obfuscates cache timing variations. However, an attack with timing variations aggregating from multiple side-channels potentially enables higher efficacy at higher speeds. For two applications based attacks where one application role plays as a *transmitter* and the other as a *receiver*, MultiCon proposes placement of code and data such that they share multiple underlying hardware channels that aggregate the timing variations. The *transmitter* application occupies the shared hardware and creates contention, which consequently causes latency delays for the *receiver* application's accesses. The *transmitter* application creates contention for a short period to leak secret bit 1, and stays idle to transmit secret bit 0.

The proposed MultiCon attack setup is evaluated on the *TileGx-72* multicore processor with and without the presence of external noise originating from concurrently

2576-6996/22/$31.00 ©2022 IEEE
DOI 10.1109/ICCD56317.2022.00024

executing applications. The single-channel baseline cache, network-on-chip, and memory controller attacks are implemented and evaluated in terms of efficacy and speed. The efficacy metric quantifies the true positive (TP) rate of information leakage, which relies on the timing-variations measured by the *receiver* application. On the other hand, the speed of the attack is measured using bits leaked per million cycles (*bpmc*) metric. The *bpmc* relies on the number of repetitions needed to de-noise the timing-channel to achieve a target efficacy, as well as the average time taken by the *receiver* application to conduct its accesses for bit leakage. It is shown that ∼110 cycles timing-variations are sufficient for reliable attack using the cache channel. However, ∼5 cycle timing-variations for network-on-chip channel, and ∼10 cycles for memory controller channel require repetition codes for reliable attacks. In the presence of external noise, all single-channel attacks are impacted with reduced efficacy and require increased repetitions to achieve the target efficiency. Therefore, all these attacks suffer from significant slowdowns. On the contrary, the proposed MultiCon attack aggregates timing-variations (i.e., ∼125 cycles) and shows less sensitivity to external noise since these variations originate from multiple sources of contention points. Consequently, it requires less number of repetitions to achieve the desired efficacy, and results in a speed-efficient attack. The results show that MultiCon achieves a 100% TP rate with the highest speed among all evaluated attacks without external noise. In the presence of external noise, MultiCon achieves a 90% TP rate target that is >3× faster than the baseline single-channel cache, network-on-chip, and memory controller attacks.

II. Single-Channel SCA Attacks

This section discusses the working of existing timing-based SCAs that target individual hardware channels, including shared cache, network-on-chip, and memory controller.

A. Cache attack

Cache contention attacks exploit timing variations for data access operations under different cache states (i.e., cache-hit or cache-evict). This work uses cache contention using the Flush+Reload [3] attack. The efficacy of a timing attack depends on the timing variations (*delta*), which is a difference between two memory access states. Suppose an application makes access to a data (*time*), and data is not available in the cache hardware. This triggers a cache miss, and data is fetched from main memory using a memory controller. This operation typically takes ∼140–160 cycles. On the contrary, if the data is available in the shared cache, the same operation takes ∼30–40 cycles to move requested data to its destination core. This results in a timing-variation (*delta*) of ∼110 cycles. The timing-variations between cache-hit and cache-miss are sufficient to create a practical attack without the need of repetition codes (*codes*).

B. Memory Controller attack

Memory controller SCA attacks temporally exploit timing variations on memory-controller hardware queues and buffers [6]. For a memory controller contention attack, the receiver application makes random accesses to its data, and measures the data arrival *time*, which is in ∼140-160 cycles range. The data comes from the DRAM memory via the memory controller for all data accesses. Concurrently, the transmitter application randomly accesses its data such that all requests involve the same memory controller hardware as the receiver application. This causes contention on the memory controller hardware queues and buffers. This contention results in timing-variations (*delta*) of ∼10 cycles for the applications sharing the memory controller hardware. This timing variation enables the timing attack using the memory controller hardware.

C. Network-on-Chip attack

NoC attack targets shared links, buffers and crossbars of the per-core router hardware for timing variations. Prior work, ConNoC [5] proposes an attack setup for the NoC contention attack. Suppose an application makes data access that is served by another core (remote cache) in the multicore. This takes ∼30–40 cycles on average for the remote cache access *time*. Now, another application randomly accesses its data such that it shares the underlaying NoC hardware crossbars, buffers, and links. This results in contention situation at the NoC hardware, and it increases the *delta* by ∼2–5 cycle for the applications sharing the NoC hardware resources. This timing variation is insufficient for practical attacks, and requires repetition *codes* to improve the efficacy of the attack.

III. MultiCon: Multiple-Channel SCA Attack

This section discusses our motivation for MultiCon attack, threat model, attack setup and practical attacks using MultiCon.

A. Motivation for MultiCon Attack

A cache contention attack efficacy is drastically reduced in the presence of external noise due to decrease in timing variations (*delta*). The external noise can be added by executing a concurrent application on the system. The random accesses by the noise generating application may evict data items from the cache sets participating in the underlying cache SCA attack. In such scenarios, the receiver application may observe high latency due to main memory access, while in the absence of noise the access would have been a fast cache hit. This operation increases the average memory access *time* since it now requires a ∼140–160 cycles compared to a ∼30–40 cycles (when data is present in the cache). This makes the attack slower due to an increase in average *time*. To overcome the noise challenge, the attack requires repetition *codes*. It re-transmits the secret bit value multiple times and reduces the probability of noise affecting multiple instances of secret bit value. The

speed of attack is multiple of *time* and *codes*, and both contribute to slowing down the attack.

The external noise also significantly reduces the timing-variations between contention and no-contention on memory controller hardware. It reduces the overall *delta*, and the adversary requires a high number of repetition codes (*codes*) to overcome this challenge. Further, external noise causes additional unwanted contention on memory controller that results in an increase in the average access time (*time*). A high number of *codes* and expensive *time* reduces the overall speed of the memory controller attack as compared to the cache attack.

Similar to memory controller attack, external noise significantly reduces the timing variation between contention and no-contention on NoC hardware and reduces the efficacy of the attack. Therefore, a higher number of repetition codes are needed to overcome this challenge. Contrary to memory controller attack, where external noise increases the *time* due to occupation of limited buffers, the noise affects distributed buffers of NoC hardware, and *time* shows a negligible increase. Although NoC attack has a lower average memory access time (*time*) due to all accesses involving shared cache access, it needs a large number of repetitions (*codes*) to de-noise the channel. The reason for higher *codes* is due to the already tight timing-variations (i.e., a *delta* of ∼2–5 cycles). However, a high number of *codes* reduces the overall speed of the attack.

B. Threat Model

The threat model is adopted from timing-based SCAs on shared hardware resources [3] [5] [6]. It is assumed that the adversary is capable of executing co-located malicious application(s) on the system. Since it has system-level access, the adversary also controls the co-location of data on shared hardware (i.e., caches, NoC, and memory controller). Additionally, the adversary can measure the timing-variations across multiple data accesses using cycle-accurate hardware counters. The threat model focuses on software-based timing SCAs, and considers physical attacks such as power analysis, thermal monitoring, and electromagnetic attacks as orthogonal attack vectors.

C. MultiCon Attack Setup

The existing baseline attacks target a single shared hardware in a multicore processor that limits their efficacy and speed. We propose MultiCon, a multiple channel timing-based SCA that aggregates timing-variations of cache, NoC, and memory controller hardware. Figure 1 shows the placement of two applications using the proposed MultiCon attack setup. R-App (i.e., receiver) application is pinned on tile 8, where its code is pinned on tile 5. T-App (i.e., transmitter) application is pinned on tile 7, and its data is pinned on tile 5. Both application's code and data are virtually isolated but share the underlying cache, NoC, and memory-controller hardware. Suppose R-App makes memory access, and its data is available in cache memory while NoC and memory controller are

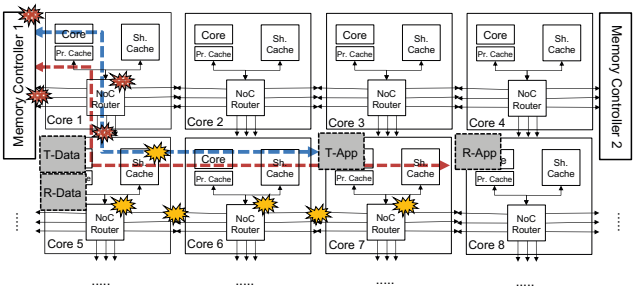

Fig. 1. A block view of a multi-core processor consisting of two applications that share hardware resources including 1) caches, 2) network-on-chip, and 3) memory-controllers, and contention points.

not contented. The data arrives in ∼30–40 cycles. On the other hand, if data comes from the DRAM while NoC and memory controller are contented, it takes ∼160–180 cycles. This additional contention on NoC and memory controller increases the timing-variations (*delta*) to ∼125 cycles. High timing-variations are less sensitive to external noise and require fewer repetition codes (*codes*) to overcome external noise. Although this additional contention on NoC and memory controller increases the average memory access *time*, the increased *delta* decreases the number of repetitions (*codes*) needed to de-noise the channel. MultiCon has an average memory access *time* similar to cache attack, but greater than the NoC attack. However, a high *delta* of ∼125 cycles reduces the required repetitions (*codes*), resulting in a faster attack than cache and NoC. Memory controller attack has the highest *time* and requires high *codes* due to low *delta*. Thus, it is the slowest attack as compared to others, including MultiCon. The higher timing variations of MultiCon enable a fast attack in the presence of realistic conditions (i.e., external noise) for practical SCA attacks, such as covert-communication between co-located applications on the processor.

D. Practical Attacks using MultiCon

This section use MultiCon setup to demonstrate two practical attacks 1) covert-communication attack, and 2) function monitoring attack. In covert-communication attack, two unauthorized applications communicate using timing variations. Whereas in function monitoring attack, a malicious application infer secrets from a secure application that execute secrets-based functions (i.e., RSA crypto-system).

1) Covert-Communication Attack: In a covert-communication attack, the transmitter application (T-App) shares a cache line with the receiver application (R-App) using a shared library. The transmitter is able to modify the state of the shared cache line and contented on NoC and memory controller hardware. Figure 2 shows the pseudo code for the covert-communication attack using MultiCon attack setup. For initial setup, a global cycle counter is used for synchronization purposes between T-App and R-App. For transmission of bit 0, the T-App

```
// transmitter pseudo code

wait_sync_flag();

for every bit:
if (bit == 1):
   for (num of replays):
      contention_generator();
else if (bit == 0):
   for (num of replays):
      stay_idle();
```

```
// receiver pseudo code

wait_sync_flag();

for every bit:
timer_start();
   for (num of replays):
      contention_monitor();
stop_timer();

if (time > threshold):
   bit = 1
else
   bit = 0
```

Fig. 2. Pseudo-code for covert-communication attack using MultiCon attack setup.

executes *contention generation* function that places data in the cache line. Whereas for bit 1, the T-App flushes the data from the cache line and contends on the NoC and memory controller hardware. On the receiver side, the *contention monitor* function measures the contention on cache, NoC, and memory controller with help of a cycle accurate timer. A contention situation is classified as bit 1, whereas non-contention situation is inferred as bit 0. This process is repeated to leak a stream of secret information over the convert communication channel.

```
// secure sample code

wait_sync_flag();

if (secret_bit == 1):
   for (num of replays):
      high_function(secret);
      low_function(secret);
else if (secret_bit == 0):
   for (num of replays):
      low_function(secret);
```

```
// adversary pseudo code

wait_sync_flag();

for secret every bit:
timer_start();
   for (num of replays):
      contention_monitor();
stop_timer();

if (time > threshold):
   current_bit = 1
else
   current_bit = 0

if (last_bit == 1 && current_bit = 0):
   secret_bit = 1
else if (last_bit == 1 && current_bit = 1):
   secret_bit = 0
```

Fig. 3. Pseudo-code for function monitoring attack using MultiCon attack setup.

2) Function Monitoring Attack: The function monitoring attack models a function based on square and multiple algorithm of the RSA crypto-system [3]. Figure 3 shows the pseudo code of the function monitoring attack using MultiCon attack setup. The threat model allows adversarial application to interrupt and control execution of the secure function. This capability allows the adversarial application to implement repetition codes on the secure application. Similar to covert communication attack, a global cycle counter is used for synchronization purposes. For secret bit 1, the RSA type algorithm executes two functions that generate contention and no-contention situations. Whereas for bit 0, the secure function executes the contention generation function only. An adversary uses the contention levels to infer secret bits. For example, a contention situation followed by no contention situation is categorized as secret 1, and a contention situation followed by another contention situation is inferred as secret bit 0.

IV. Methodology

The baseline attack setups and the proposed Multi-Con attack are implemented on the Tilera® *Tile-Gx72™* multicore processor. Tile-Gx72 is a tiled shared-memory architecture with 72 independent tiles and four memory controllers connected to 4 different tiles. Each tile has a 64-bit multi-issue in-order core, $32KB$ private level-1 (L1) data and instruction caches, and a slice of shared level-2 (L2) cache of $256KB$ aggregate (shared cache capacity of $18MB$ distributed in 72 tiles). The cache line size for private and shared caches is 64 bytes. It consists of Network-on-Chip hardware, which comprises five independent 2-D mesh networks with X-Y routing (iMesh™ [8] Technology). Each NoC flit is 8 bytes (64 bits) in size. Four 72-bit ECC-protected DDR memory controllers are connected to corner tiles to access off-chip memory. The system uses a GNU/Linux operating system with kernel version *3.10.55-MDE-4.3.2.182362*. Tilera Multicore Components (TMC) API library is used to manage code and data placement for receiver and transmitter applications and tile resources, including memory-controller and NoC traffic.

A. External Noise Application

Stress application [9] version 1.0.4 is used to generate low, medium, and high noise levels. The Stress application threads are randomly pinned on different tiles such that it shares cache, NoC, and memory controller hardware channels with the co-located transmitter and receiver applications for the SCA attack. It is capable of executing multiple threads that generate noise by injecting random data accesses. A single thread spins on malloc()/free() instructions that access data from the targeted memory controller and contend on the caches and network routers on its path. The evaluation uses a single thread for low noise, four threads for medium noise, and eight threads for high noise.

B. Evaluation Metrics

The baseline and MultiCon attack's accuracy and reliability is measured using Timing-variations (*delta*), True Positive (TP) rate [5], and Discrimination Index (DI) [10] metrics. The *delta* is the average timing difference between two memory accesses under contention and no-contention situations. The TP rate measures the correct contention or no-contention situation detection on shared hardware (i.e., cache, NoC, memory-controller). For example, if a transmitter application creates a contention situation on shared hardware, the receiver application successfully detects a contention situation. We calculate the TP rate using $TP = \frac{d_c+d_n}{t_c+t_n}$, where d_c is total correctly inferred contention cases, and d_n is the correctly inferred no-contention case. The t_c is the total number of contention situations, and t_n is the total number of no-contention

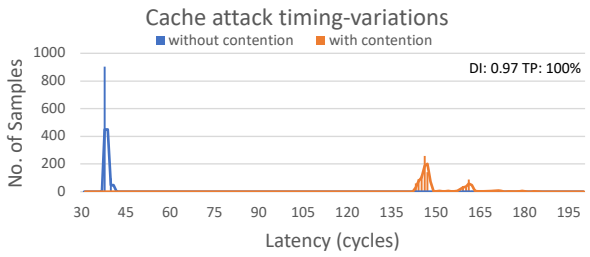

Fig. 4. Timing-variations histogram of cache attack setup.

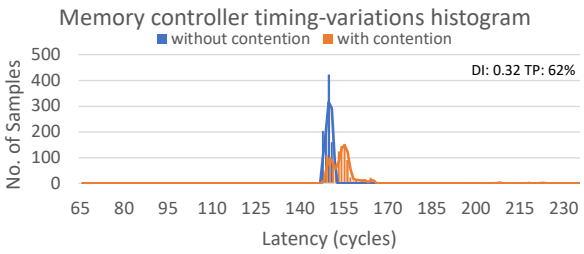

Fig. 5. Timing-variations histogram of memory controller attack setup.

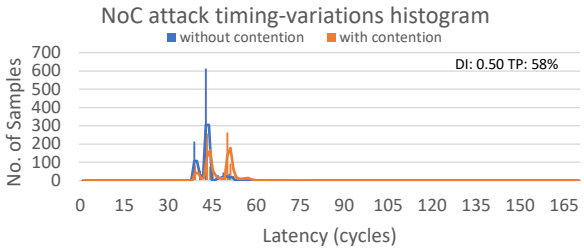

Fig. 6. Timing-variations histogram of NoC attack setup.

situations. The timing variations are plots for contention and no-contention situations on shared hardware, and the TP rate is inadequate to measure the reliability of an attack. The DI metric quantifies the two distributions of timing variations under contention and no-contention. DI uses the values of statistical mean and the variance of contention and no-contention distributions. We calculate the value of DI for a specific attack using $DI = \frac{\mu_c + \mu_n}{\sqrt{\sigma_c^2 + \sigma_n^2}}$, where μ_c is the statistical mean of distribution in contention plot, and μ_n is the statistical mean of distributed in no-contention plot. The σ_c^2 and σ_n^2 are the variance of distributions under contention and no-contention, respectively.

The speed of the attacks is quantified using the average time of memory access for the receiver application (*time*), and the number of repetition codes (*codes*) needed to de-noise the channel. It uses bits per million cycle (*bpmc*) metric for overall speed. *Bpmc* measures the number of bits that a receiver application receives in one million cycles. The attack speed is a function of the average time to make memory access by the receiver (*time*) and the number of times a single bit is received (*codes*). So, for a given attack setup, the speed of attack is calculated as $Speed = \frac{1 \times 10^6}{times \times codes}$ *bpmc*.

V. Evaluation

This section evaluates cache, memory controller, NoC, and MultiCon attack setups on the Tile-Gx72 multicore processor. It uses TP and DI metrics to evaluate the efficacy of attacks. On the other hand, it evaluates the speed of attack using the bits-per million cycle metric. The attacks are also evaluated using *Stress* [9] application under low, medium, and high noise configurations. An information theory technique, repetition codes [7] is used to de-noise the attack setups.

A. Evaluation without repetition codes

This section evaluates baseline and MultiCon attack setups with and without noise. The receiver application makes memory access to a shared cache line for a cache contention attack. In the case of cache-hit, the data arrives in ∼40 cycles (*time*). On the contrary, in the case of cache-miss, the data that arrives from the main memory involves the memory-controller hardware and takes ∼155 cycles (*time*). The transmitter application controls cache data placement, thus controlling cache-hit/miss. This high *delta* of ∼115 cycles is sufficient for a practical SCA attack. Figure 4 shows the timing-variations histogram for cache contention attack. In memory controller contention attacks, the receiver application makes access to random data. Data is fetched from the main memory involving the memory controller. This operation takes ∼155 cycles (*time*). In the other case, the transmitter application creates contention on the memory-controller by continuously making memory accesses and occupying memory-controller shared queues. In this case, the receiver application takes ∼165 cycles (*time*), and overall timing-variations of ∼10 cycles (*delta*) are observed. Figure 5 shows timing-variations for the memory controller attack. Similarly, Figure 6 shows timing variation for the NoC channel, where contention case takes ∼45 cycles for memory access (*time*), and no-contention case takes ∼40 cycles (*time*). This confirms the low *delta* of ∼5 cycles for the NoC attack.

In MultiCon attack, the receiver application makes a data access. In the case of cache-hit, the data arrives in ∼40 cycles (*time*) as the transmitter is performing no operations of its own. Whereas, in the case of cache-miss, the transmitter is simultaneously making random accesses to the DRAM that involves memory controller and NoC hardware as well. This controlled contention delays the latency for receiver application accesses to ∼170 cycles (*time*), resulting in a timing-variation of ∼125 cycles (∼15 additional cycles compared to cache contention attack). Figure 7 confirms the timing variations for MultiCon attack setup without any noise and repetition codes.

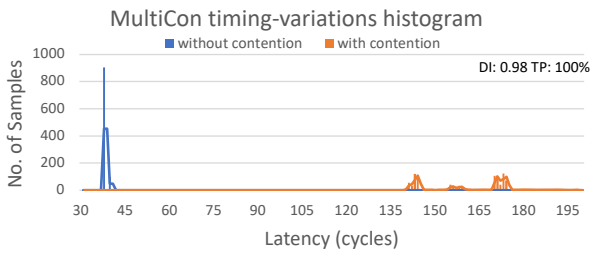

Fig. 7. Timing-variations histogram of MultiCon attack setup.

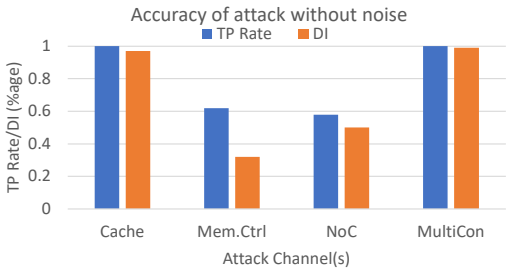

Fig. 8. Efficacy of cache, memory controller, NoC and MultiCon attack without external noise

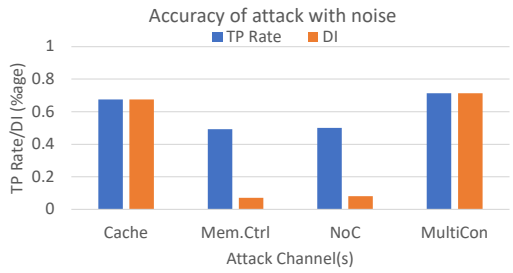

Fig. 9. Efficacy of cache, memory controller, NoC and MultiCon attack with medium level of external noise

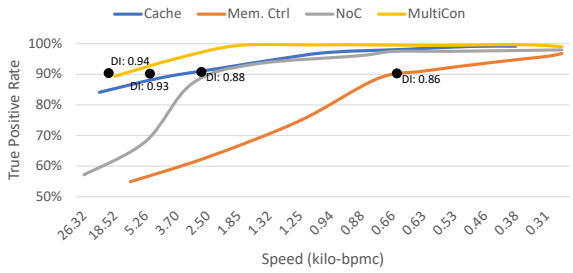

Fig. 10. Speed vs TP rate plots for cache, NoC, memory controller and MultiCon attack with Low noise

The efficacy of attack setups is quantified using TP and DI metrics. Figure 8 summarizes the efficacy for baseline and MultiCon attack setups. Cache and MultiCon show a TP rate of 100% and DI of ∼1 due to high timing variations. The memory controller and NoC attacks show ∼10 cycle and ∼5 cycle timing variations respectively, and low *delta* reduces the efficacy of the attack setup. A low DI of 0.50, 0.32, and a low TP rate of 62% and 58% confirm the low efficacy of memory controller and NoC attacks without repetition codes.

Low, medium and high noise is generated using the Stress application that executes in parallel with the transmitter and receiver applications. Figure 9 shows the efficacy of baseline and MultiCon attack setups in the presence of medium noise levels without repetition codes. The TP rate drops to 68% for cache contention attack, compared to 100% without noise. Similarly, the TP rate drops to ∼50% for memory controller and NoC attacks compared to ∼60% without noise. The MultiCon attack shows a 72% TP rate and DI of 0.68 due to its high efficacy, i.e., timing variations (*delta*=125 cycles). External noise is sufficient to make baseline and MultiCon attacks less effective, and the adversary requires de-noising capabilities (i.e., repetition codes) to make the attack successful.

B. Evaluation with repetition codes

The noise reduces the efficacy and speed of the attack due to an increase in contention on shared buffers/queues of shared hardware and extra memory accesses involving memory-controller hardware. Therefore, repetition codes are needed to increase the efficacy of the attack. Repetition codes re-transmit every bit of secret information multiple times, and multiple transmissions reduce the speed of attack.

Figure 10 confirms the increase in TP rate with a decrease in speed of the attack. Speed of attack depends on average time for a receiver to make a memory access (*time*) and number of repetition codes (*codes*). The external noise impacts both *time* and *codes*. Figure 10 plots attack speed and efficacy (TP rate) in presence of low external noise. A low-noise situation causes cache misses and results in additional DRAM memory access in cache contention attacks, reducing the timing variations (*delta*) and consequently the efficacy (TP Rate). An increase in repetition codes increase the TP rate but reduce the speed of attack. For low noise, the cache contention attack achieves the 90% TP rate with a DI of 0.94 at a speed of 4 *kilo-bpmc* with repetition codes of 5. Similarly, low noise affects the timing variations of MultiCon attack due to extra cache misses, but high timing variations in MultiCon require one repetition code to achieve the target 90% efficacy. This results in a faster attack at a speed of ∼18 *kilo-bpmc*, which is 4.5× faster compared to cache attack. Similarly, NoC attack shows a speed of 2.40 *kilo-bpmc* and memory controller contention attack reaches 90% TP rate at 0.65 *kilo-bpmc*. Both these attacks are much slower than the cache and MultiCon attacks.

Figure 11 plots the attack speed and efficacy (TP rate) in presence of medium external noise. A medium noise

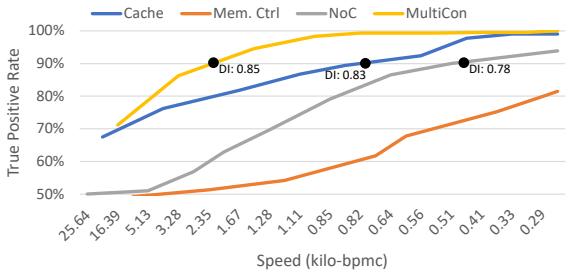

Fig. 11. Speed vs TP rate plots for cache, NoC, memory controller and MultiCon attack with Medium noise

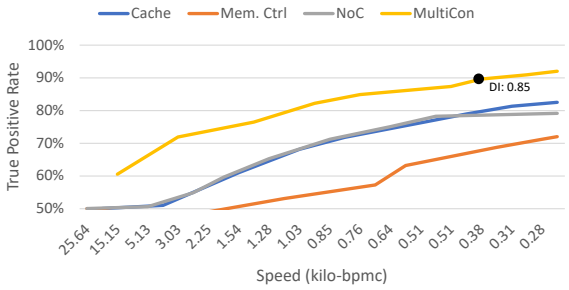

Fig. 12. Speed vs TP rate plots for cache, NoC, memory controller and MultiCon attack with High noise

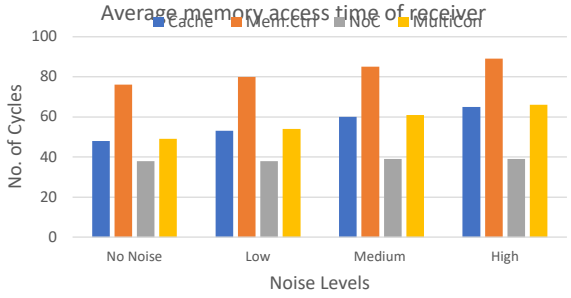

Fig. 13. Average memory access time for receiver applications under no-noise, low, medium, and high noise situations.

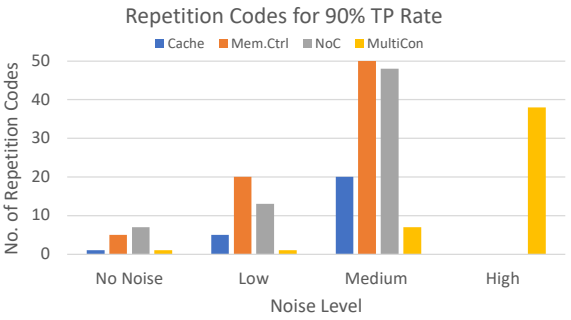

Fig. 14. No. of repetition codes needed to achieve 90% TP rate.

increases cache-misses and contention on memory controller queues for cache contention attacks. It reduces the timing-variations between contention and no contention cases, and causes an increase in repetition codes for the receiver application. The cache contention attack shows 90% TP rate with DI of 0.83 at a speed of 0.83 *kilo-bpmc*. NoC and memory controller have low timing-variations, and medium noise is sufficient to minimize the timing-variations. A low timing-variations requires a high repetition codes to overcome, and results in low attack speeds. The NoC attack shows a speed of 0.48 *kilo-bpmc* for 90% TP rate with DI of 0.78. The memory controller has high average memory access time, and requires high number of repetition codes, that consequently reduces the speed. Medium Noise with MultiCon setup is able to show a TP rate of 90% with speed of 2.10 *kilo-bpmc* with a DI of 0.85. This is ∼3x faster compared to the cache attack. Figure 12 shows the results for high external noise scenario. High noise reduces the efficacy of baseline and MultiCon attack setups and requires a high number of repetition codes to overcome. MultiCon is the only attack setup that is able to achieve a 90% TP rate with 0.85 DI and at speed of 0.38 *kilo-bpmc*. All baseline attack setups fail to achieve efficacy with 90% TP rate.

The speed of attack is the product of average time to make a memory access (*time*) by receiver application and number of repetition codes. Figure 13 shows the receiver application's average time for memory access (*time*) to receive a single bit once using various channels and noise situations. For example, on average, a cache contention attack takes ∼48 cycles to measure memory access. In the presence of high noise, the *time* increases to ∼63 cycles due to unwanted cache misses and contention on memory controller queues. The MultiCon attack shows comparable average memory access (*time*) to cache attack. The NoC attack shows smallest *time* and memory contention attack has highest *time*. Figure 13 summarizes the average access *time*. Repetition codes are required to overcome noise challenges, and the number of repetition codes depends on the timing variations (*delta*). Although cache and MultiCon attacks have similar *time*, MultiCon high timing variations are less sensitive to external noise and require fewer repetition codes. Figure 14 shows the number of repetition *codes* required by baseline and MultiCon attack setups under low, medium, and high noise to achieve a high threshold of TP rate (i.e., 90%). Figure 14 confirms that MultiCon significantly lowers the number of repetitions due to its high timing-variations, and cache, memory controller and NoC attacks all require high repetitions to achieve the target TP rate of 90%.

VI. Related Work

Timing variation attacks are extensively studied to develop software SCA [2] [3] [4] [11] [12] [13] [14] [5] [6] [15] to leak security-sensitive information. Cache attacks such

as Prime+Probe [2] and Flush+Reload [3] are shown to leak private keys of AES and RSA cryptosystems. In the Prime+Probe attack, the receiver application operates on a cache set level granularity and has a high average time to access memory (i.e., a cache set where a cache set contains multiple cache lines). In the Flush+Reload attack, the receiver application operates at cache line-level granularity, resulting in a faster attack. Memory-controller contention attacks [16] [6] [17] [18] are studied in literature for denial-of-service (DoS) and timing-based SCAs. A prior work [16] shows a DoS attack that targets memory controllers in a multicore machine. It exploits the unfair allocation of shared resources of a memory controller. Other works [6] [18] also demonstrate the timing-variations on a memory controller for practical attacks (i.e,. covert-communication attack). Shared NoC hardware is also targeted [11] [5] for timing-variation attacks. Ed. Suh [11] shows the possibility of timing attacks that exploit the NoC hardware. In recent work, ConNOC [5] proposes a novel placement of code and data to exploit shared NoC hardware efficiently and demonstrates it on real hardware. Prior works [19] [20] have also used noise to show that it reduce the timing-variations of timing attacks, and [7] [5] have successfully shown the use of repetition codes to amplify the timing-variations. Such attacks are mitigated using isolation and obfuscation based mitigation schemes [21].

VII. Conclusion

This paper proposes a multi-channel timing-based side-channel attack (called MultiCon) that aggregates timing-variations on shared cache, network-on-chip, and memory-controller hardware in a multicore processor. It implements the proposed attack and baseline single-channel attacks (cache, NoC, and memory controller contention attacks) on a real Tile-Gx72 processor and shows that the efficacy of attacks is drastically reduced in the presence of external noise. It uses repetition codes to increase the efficacy of the attack setups but at the cost of speed. MultiCon with 90% TP rate (efficacy) has a speed that is greater than 3× faster as compared to baseline attacks in the presence of external noise.

VIII. Acknowledgments

This research was supported by the National Science Foundation under Grant No. CNS-1929261. This research was also supported in part by the Semiconductor Research Corporation (SRC).

References

[1] S. Röttger and A. Janc, "A spectre proof-of-concept for a spectre-proof web," Mar 2021.

[2] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of aes," in *Topics in Cryptology – CT-RSA 2006* (D. Pointcheval, ed.), (Berlin, Heidelberg), pp. 1–20, Springer Berlin Heidelberg, 2006.

[3] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*, (San Diego, CA), pp. 719–732, USENIX Association, Aug. 2014.

[4] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment* (J. Caballero, U. Zurutuza, and R. J. Rodríguez, eds.), (Cham), pp. 279–299, Springer International Publishing, 2016.

[5] U. Ali and O. Khan, "ConNOC: A practical timing channel attack on network-on-chip hardware in a multicore processor," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2021.

[6] Y. Wang, A. Ferraiuolo, and G. E. Suh, "Timing channel protection for a shared memory controller," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 225–236, 2014.

[7] D. Skarlatos, M. Yan, B. Gopireddy, R. Sprabery, J. Torrellas, and C. W. Fletcher, "Microscope: Enabling microarchitectural replay attacks," *IEEE Micro*, vol. 40, no. 3, pp. 91–98, 2020.

[8] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, p. 15–31, Sept. 2007.

[9] "STRESS - tool to impose load on and stress test systems." https://linux.die.net/man/1/stress, 2019.

[10] Y. Massoud, J. Kawa, D. MacMillen, and J. White, "Modeling and analysis of differential signaling for minimizing inductive cross-talk," in *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, (New York, NY, USA), p. 804–809, Association for Computing Machinery, 2001.

[11] Y. Wang and G. E. Suh, "Efficient timing channel protection for on-chip networks," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pp. 142–151, 2012.

[12] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 955–972, USENIX Association, Aug. 2018.

[13] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1–19, 2019.

[14] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 973–990, USENIX Association, Aug. 2018.

[15] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring(s): Side channel attacks on the CPU On-Chip ring interconnect are practical," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 645–662, USENIX Association, Aug. 2021.

[16] T. Moscibroda and O. Mutlu, "Memory performance attacks: Denial of memory service in Multi-Core systems," in *16th USENIX Security Symposium (USENIX Security 07)*, (Boston, MA), USENIX Association, Aug. 2007.

[17] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for cross-cpu attacks," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 565–581, USENIX Association, Aug. 2016.

[18] A. Shafiee, A. Gundu, M. Shevgoor, R. Balasubramonian, and M. Tiwari, "Avoiding information leakage in the memory controller with fixed service policies," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 89–101, 2015.

[19] C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, S. Mangard, and K. Römer, "Hello from the other side: Ssh over robust cache covert channels in the cloud.," in *NDSS*, vol. 17, pp. 8–11, 2017.

[20] S. A. Crosby, D. S. Wallach, and R. H. Riedi, "Opportunities and limits of remote timing attacks," *ACM Trans. Inf. Syst. Secur.*, vol. 12, jan 2009.

[21] U. Ali, A. R. Sahni, and O. Khan, "Protecting on-chip data access against timing-based side-channel attacks in multi-core systems," in *IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*, 2022.