Guard-NoC

A protection against side-channel attacks for MPSoCs

Reinbrecht, Cezar; Aljuffri, Abdullah; Hamdioui, Said; Taouil, Mottaqiallah; Forlin, Bruno E.; Sepulveda, Johanna

DOI
10.1109/ISVLSI49217.2020.000-1

Publication date
2020

Document Version
Accepted author manuscript

Published in
2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

Guard-NoC: A protection against Side-Channel Attacks for MPSoCs

Cezar Reinbrecht, Abdullah Aljuffri, Said Hamdioui, Mottaqiallah Taouil
Delft University of Technology
Faculty of EE, Mathematics and CS
m.taouil@tudelft.nl

Bruno E. Forlin
Universidade Federal do Rio Grande do Sul
Instituto de Informatica
bruno.eforlin@inf.ufrgs.br

Johanna Sepúlveda
Airbus Defense and Space
Munich, Germany
johanna.sepulveda@airbus.com

***Abstract*—Multi-Processor System-on-Chips (MPSoCs) are popular computational platforms for a wide variety of applications due to their energy efficiency and flexibility. Like many other platforms they are vulnerable to Side Channel Attacks (SCAs). In particular, Logical SCAs (LSCAs) are very powerful as sensitive information can be retrieved by simply observing system properties that depend on the victim's software execution on the MPSoC. Unfortunately, many of the current protection mechanisms are either platform dependent or are effective only against a reduced set of attacks. In this work, we present *Guard-NoC*, a secure Network-on-Chip (NoC) architecture able to protect MPSoCs against a wide variety of LSCAs. The secure NoC employs three application-independent strategies to hide and isolate sensitive information: i) blinding the execution time of operations; ii) masking the execution time of operations; and iii) dual communication strategy (i.e., use packet and circuit switching simultaneously). Our results show that our secure NoC is resilient against practical LSCAs and leaks almost no information while having a minimal area and power overhead.**

***Index Terms*—Network-on-Chip, Countermeasure, Side-Channel Attack, Hardware Security**

I. Introduction

Multi-Processors System-on-Chips (MPSoCs) play a major role in many electronic devices such as servers, smartphones and many other Internet-of-Things devices. The wide spread adoption of MPSoCs in critical applications has turned them into an interesting attack target. Attackers might exploit one or more of the following main components of MPSoCs [1]: *node*, *communication*, and *interface*. Each component present vulnerabilities that can be used by the so-called side-channel attacks (SCAs) [2]. SCAs are successful attack techniques whose goal is to retrieve secret data by analyzing physical (e.g., power dissipation) or logical effects (e.g., timing) produced during the normal operation of the system. Today, logical SCAs (LSCAs) are a major threat in the semiconductor industry as they can be performed completely in software, and often remotely [3]. Besides, they are hardly detectable because these attacks only observe the system's behavior.

A great number of hardware and software countermeasures have been proposed to avoid LSCAs in MPSoCs. With respect to MPSoC *node* protection, most software countermeasures have focused on the avoidance of timing leakage. In [4], the authors modified an implementation of Advanced Encryption System (AES) to hide the time behavior of cache misses. In [5], the authors presented an instruction scheduler as part of a compiler to prevent timing leakage of cache-dependent operations. In [6], the authors proposed the concept of static and dynamic software diversification to prevent timing and access leakage. On a similar line of thought, the authors in [7] added random permutations during AES encryption to mask the cache access patterns. On the other hand, the hardware countermeasures for node protection are based typically on isolation strategies. In [8], the authors use cache partitioning to dedicate (parts of) caches for secure computation. The industry further advanced this concept by creating secure zones, also known as Trust Execution Environments (TEE), which isolate the sensitive tasks completely; examples are Arm Trust Zone [9], Intel SGX [10], and Sanctum (RISC-V distribution) enclave [11]. Recently, the authors in [12] and [13] have used machine learning techniques trained with high-performance counters to detect a specific set of cache access attacks, while the authors in [14] used attack models to build a lightweight hardware detector. With respect to MPSoC *communication*, only hardware solutions have been presented. In [15] and [16], the authors proposed different protection mechanisms against NoC timing attacks. The former one integrates Quality-of-Service (QoS) mechanisms to isolate sensitive traffic, while the latter one adds random arbitration of packet switching and adaptive routing to mask the communication leakage. In [17], the authors proposed an MPSoC with several security mechanisms inside the NoC. Their NoC targets to secure the communication of the system by having dynamic configuration of secure zones, trusted routing, and some cryptographic capabilities to the network interface. As they only focus on communication, the nodes are still vulnerable to LSCAs. With respect to the MPSoC *interface*, no logical attacks have been reported so far. However, attacks on JTAG and test interfaces of integrated circuits already were published in [18, 19]. The above countermeasures against logical attacks targeting *node* and *communication* clearly are limited. They either focus on a subset of logical attacks and in addition are typically design dependent or application dependent. Therefore, there is a clear need for an efficient solution that is able to protect all parts of an MPSoC against LSCAs.

In this paper, we propose to embed countermeasures for the whole MPSoC in the communication infrastructure, referred to Guard-NoC. The main contributions of the paper are:

- The design of a unique router which embeds blinding and masking strategies in the network interface to protect the nodes against processor and cache attacks.
- The usage of dual switching mode to protect the communication infrastructure against NoC timing attacks.
- The evaluation of the proposed NoC against popular LSCAs in a hardware emulation platform (FPGA).
- Trade-off analysis considering performance and hardware overheads when using Guard-NoC.

This paper is organized as follows. Section II presents information regarding NoCs and LSCAs. Section III introduces Guard-NoC. Section IV presents the experiments and results. Finally, Section V concludes this paper.

II. BACKGROUND

In this section, we present background information regarding Networks-on-Chips and Logical Side-channel attacks.

A. Network-on-Chip (NoC)

The NoC transmits packets between a source IP (which injects the packet) and destination IP (which receives the packet). Routers are used to move the packets through the network. Each processing node communicates with a router through a network interface (NI). The NI implements the communication protocol and wraps information into packets during sending and unwraps them during receiving. The intra-chip communication has a tremendous impact on overall system performance and area and power costs [20]. It has a central role in the system, which makes the NoC also an attractive component to implement security mechanisms. The NoC comprises four main parameters [20]:

i) Topology, defines the way the routers are interconnected;

ii) Routing, specifies the routing of packets between a sender (source) and a receiver (destination);

iii) Switching mode, defines the commutation of information. There are two possible switching modes in an NoC, circuit switching and packet switching. In circuit switching, a dedicated path is created from the source node to the destination node by pre-configuring all the routers in the path before the packet injection to the NoC. All remaining communication flows are unable to use the pre-defined route. Resources can only be released when the information reaches the destination. On the other hand, in packet switching, the message transverses into small parts (defined as flits). Each flit travels separately in the NoC, following the path of the header. The routers configure the switches only when headers pass. In this strategy, the paths are created on the fly during the transmission, which results in less performance penalties;

iv) Flow control, synchronizes the transmissions between routers.

In this work, we target to protect a NoC that uses mesh topology, deterministic XY routing algorithm, dual switching mode (packet and circuit switching), and handshake flow control. The flit granularity is 32 bits. Our target NoC-based MPSoC is shown in Fig. 1.

B. Logical Side-channel Attacks

Based on the leakage source, the authors in [21] classified logical side-channel attacks into three types: a) **timing**; b) **access**; and c) **trace**.

*a) **Timing Attacks**:* The authors in [22] pioneered in 1996 the concept of using different timing responses of chip components to infer hidden information. Only in 2005, Bernstein was able to perform a practical remote timing attack on servers running AES [3]. An improvement of such an attack was introduced by the authors in [23], the so-called cache collision attack. The attacker manipulates the input message of the encryption algorithm to force collisions of cache addresses (cache hits) when the key hypothesis is correct. In [21], the authors proposed a variation of this attack called differential collision cache attack. It analyzes a pair of encryptions, where the collision effects are exploited of the second run of each pair. In addition to the cache, the floating point unit of the processor has been targeted by timing attacks as presented in [24]. With respect to communication, the authors in [15] and [16] proposed NoC attacks. NoC attacks are triggered by an attacker injecting messages in the NoC. As the routers are shared, the communication collisions between malicious and sensitive traffic may reveal information of the system (e.g., component location, network topology) and application (e.g., communication affinity, transmission pattern, communication volume). Recent work, like [25], has applied such technique to realize a timing practical attack based on cache collision.

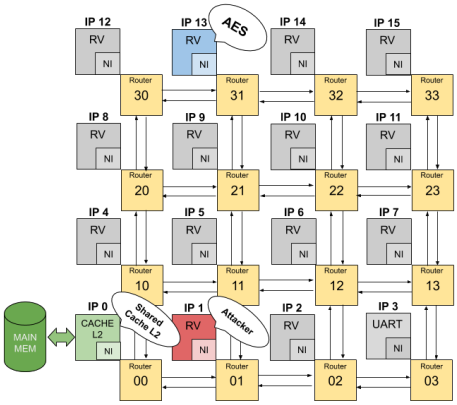


Fig. 1: Reference MPSoC Architecture.

*b) **Access Attacks**:* At the moment, access attacks are only applied to caches by observing the cache addresses used by a sensitive operation of the victim. The first access attacks were presented in [26]. Among them, the Prime+Probe attacks are still popular today. These attacks consist of three steps: i) Prime - attacker fills (parts of) the cache with data; ii) attacker triggers a sensitive operation (e.g. encryption); and iii) Probe - attacker reads the data written in the Prime step and evaluates which addresses were used by the victim based on the observation of cache misses. A limitation of Prime+Probe is that it can only be successfully applied when the attacker has good control of the addresses accessed in the shared cache, which is not a realistic assumption for all systems, such as virtualized platforms. To deal with this limitation, new attacks were proposed where a lower degree of cache control is required. Examples are Flush+Reload [27] and Flush+Flush [28] attacks. In contrast to Prime+Probe, these attacks perform the prime step by cleaning the data of the cache rather than filling it with own data by e.g. using the CLFLUSH instruction in x86 architectures. Regarding communication attacks, in [29], the authors demonstrated that it is possible to use an NoC timing attack to identify the end time of each AES round, and subsequently, apply Prime+Probe at the end of the first round to improve the attack considerably.

*c) **Trace Attacks**:* In trace attacks, an adversary obtains a profile of the resource activity of the target platform and deduce sensitive information from it, as described in [30]. This information is most commonly collected by special monitors inside the system like High Performance Counters. If the MPSoC has well-established security policies, this attack is not practical, and hence not considered in this work.

III. GUARD-NOC

Guard-NoC is a secure NoC against LSCAs. In this section we motivate the use of a secure NoC as a protection for an MPSoC, followed by the threat model adopted in this

paper. Thereafter, we describe the proposed NoC architecture. Finally, the protection mechanisms are described.

A. Motivation

The Network-on-Chip is a central component of an MPSoC architecture which handles all communication between the nodes. MPSoCs usually integrate security features such as cryptographic hardware cores for supporting confidentiality and authentication services. However, during the operation of a cryptographic core (trusted element), the secret key may passively be revealed through LSCA. In case an MPSoC application secures data by using the embedded cryptographic core, both plaintext and ciphertext information is exchanged through the NoC. Depending on the type of security function, part of the execution of the cryptographic task (e.g., an AES encryption) will use the NoC to accesses some valuable information stored in the main memory (e.g., S-Box data). In summary, the NoC is part of critical operations in the system, from memory accesses by the elements to specialized service requests/responses. Therefore, this work proposes an NoC architecture that affects the relation between nodes to hide potential timing leakages. We achieve this by applying blinding and masking countermeasures in the router. In addition, we adopt dual switching to avoid attackers infering timing information. In the following subsections, we provide more details.

B. Threat Model

Guard-NoC considers the following threat model:

- There are trusted and non-trusted nodes in the system.
- Trusted nodes run inside a secure zone and have their own isolated local resources (similar to ARM Trust zone [9]). IPs 8, 9, 12 and 13 are used as trusted nodes (see Fig. 1), but in general any other mapping is possible.
- Sensitive applications are only executed on trusted nodes. Oppositely, external applications can only run on non-trusted nodes, as they may contain malicious intentions.
- The last level cache is shared between the trusted and non-trusted nodes and is the gateway to the main memory of the system. Our target platform has level 2 as last level.
- System monitors and debug information contain sensitive and therefore can only be accessed by trusted nodes.

C. Hardware Architecture

Fig. 1 shows the MPSoC platform considered in this paper. It consists of 16 nodes interconnected by a 4x4 grid of routers. Of those nodes, 14 contain a RISC-V processor (RI5CY core from Pulpino platform [31]), one shared L2 cache (node IP 0) and one UART for external communication (node IP 3). The NoC configuration follows the description given in subsection II-A, with the only difference that Guard-NoC routers can handle both circuit and packet switching (i.e., dual switching). In addition, they include an extra component responsible for the security of the nodes called *Obfuscation Function* (see Fig. 2).

D. Protecting Nodes

The protection of the nodes is provided by the *obfuscation module* block which is shown in more details in Fig. 2. It is a dedicated hardware unit that is included between the network interface and local input channel (IC local) of each router

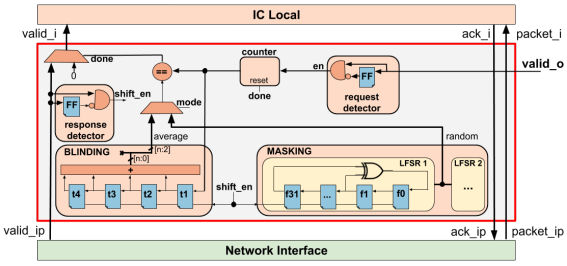


Fig. 2: Obfuscation Module.

that obfuscates the timing. By altering the delay of packets, different timing behavior of cache memories, hardware accelerators or even applications running in processors will be observed by the attacker, making the attacks much harder. The obfuscation module has two strategies, which are blinding and the masking. Blinding focuses on mitigating the leakage behavior, while masking aims to hide it by adding random noise. Each strategy is described in more details next.

1) Blinding Strategy: The blinding strategy manipulates the response time of the node and tries to make this constant. We propose three blinding techniques referred to as *average*, *bucket* [32], and *worst-case*.

Average Blinding: This technique averages the response time of the previous four responses (See t1 till t4 in Fig. 2). This blinding strategy is composed of three phases: i) Initialization: during this phase, an incoming packet asks for a service identified by signal valid_o (i.e., valid_o signal is high at local output channel of the router). This operation triggers the start of a counter and forward the request to the node. ii) Update: the response packet is ready to transmit when the encryption is completed, i.e., valid_ip from network interface is high. The value of the counter will be stored and a new average response time will be calculated based on the last 4 services, which are the last four encryptions for this node. iii) Wait+Release: When the counter reaches the average time the packet is released to the network and the counter is reset, with the exception when the operation time is higher than the average.

Bucket Blinding: *Bucket* blinding was initially proposed by Kopf et al. in [32], but only a theoretical model has been presented. Our solution is the first practical implementation of such strategy. It defines a set of fixed time responses and selects one as actual response time. For example, if eight buckets are defined, the time behavior will vary only between these eight possibilities. The drawback of this method is that designers must know in advance which applications will run on the node to have meaningful bucket values.

Worst-case Blinding: This method uses the worst case timing as a fixed response time. Each time an encryption takes longer than the worst execution time, the network interface will update this as the new worst-case time. For applications where the worst and best case timings are close, it can be an interesting alternative.

2) Masking Strategy: The masking countermeasure delays the responses by a random amount of time and hence can be seen a noise source. One of the best sources to achieve randomness is to use True Random Number Generators [33]. Due to its high cost, we rather propose the usage of pseudo-random generators in hardware. These pseudo-random algorithms can be implemented by a Linear Feedback Shift Register (LFSR). An LFSR circuit is composed of a cyclic shift register and

exclusive-or operations, as shown in the masking module in Fig. 2. The masking strategy works in a similar way as the blinding strategy, but without the initialization phase. During the update phase, a random value is generated by computing the XOR of the outputs of two LFSRs (i.e., LFSR1 and LFSR2 in the Fig. 2). Two LFSRs increases the period of the generated random numbers as described in [34]. Based on the desired amount of noise, only parts of the LFSR bits are used. Once the counter that is activated in the update phase reaches the generated random value, the packet will be released into the network ending the Wait+Release phase.

E. Protecting the Communication

Communication attacks are prevented in Guard-NoC by the usage of different switching mechanism. Although dual switching has been previously used in NoCs to achieve a high throughput [35], the usage of this technique for security purposes is novel. The idea is to use packet switching for secure packets and circuit switching for common packets. Secure packets are required when a node inside the Secure Zone communicates to a node outside this zone. In packet switching, the message is divided in flits (small parts) and transmitted independently router by router. Hence, this traffic has a minimal impact on traffic generated by other nodes in the NoC. On the other hand, circuit switching configures the network in such a way that the whole message is transmitted at once; therefore, it needs to reserve all the routers on the communication path upfront. The fine grained information transmitted in packet switching mode allows the attacker to understand precisely when traffic is passing through the local router. Consequently, by forcing attackers to use circuit switching this can be prevented. Overall, circuit switching affects performance-wise the remaining traffic much more, but security-wise it makes it much harder for the attacker to get any timing related information.

IV. Experimental Results

This section describes the experiment setup and evaluates the security, performance and hardware overhead.

A. Setup

The platform used in the experiments is based on the MPSoC architecture presented in Fig. 1. It integrates a memory node (IP_0), an UART interface (IP_3) and fourteen processing nodes, each consisting of a RISC-V processor and a private 8 kB direct-mapped cache with 4 bytes cache lines. Processing node IP_{13} executes a software implementation of the AES encryption algorithm (i.e., T-table AES). The shared L2 cache located at node IP_0 is a 16-way 256kB set-associative cache with a line size of 16 bytes. All these nodes are interconnected though a mesh-based 4×4 NoC. All security and performance evaluations were obtained through RTL simulations, while the hardware costs were obtained by synthesizing the designs with Cadence Design System tools (i.e, Genus Synthesizer) using 65 nanometer technology. Three different types of attacks were used to analyze the efficiency of Guard-NoC under different countermeasures. The rank analysis is the chosen metric for comparison, which evaluates how far is the guessed key from the correct key. According to the security policies presented in the subsection III-C, malicious applications can not run in trusted nodes where secret information resides. An attacker, however, may attack the services that are provided by these trusted elements instead. Therefore, we provide three attack use cases. For the sake of brevity, we describe one attack use case for each LSCA type:

Case 1 - Node-based Attack for Processors (Timing Attack of Bernstein): IP 1 requests encryption services from IP 13 and records the execution time of each encryption. IP 13 takes different times to compute AES when different inputs are applied because it uses look-up tables stored in memory. As the L1 cache memory is not large enough to store the AES T-tables completely, IP 13 exchanges messages with the shared cache L2 located at IP 0. IP 1 can observe the impact of the memory hierarchy on the overall encryption time. Hence, the adversary can perform Bernstein's attack [3]. We selected this attack to represent timing attacks as most timing attacks are derived from this one. As the victim in this case 1 is processor IP 13, the blinding and masking evaluations considers the protection on its router.

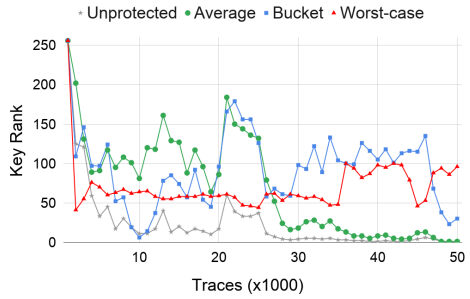
Case 2 - Node-based Attack for Caches (Access Attack Prime+Probe): IP 1 requests encryption services from IP 13. Before any request, an array with enough size to fill completely the set of cache L2 (IP 0) is read by IP 1 to prepare the cache, known as prime phase. After the encryption is finished by node IP 13, the same array is read again, and the time to access each L2 cache address is observed. In case the response time is slow, i.e., a cache miss occurred, the attacker knows that the memory position was used by the encryption algorithm on IP 13. This process is repeated until enough information is collected to successfully get the key. The Prime+Probe attack requires that an attacker has many privileges in the system. Therefore, if this attack does not succeed due to our countermeasures, other attacks like Flush+Reload [36] or Flush+Flush [28] will also likely fail as they target more restricted environments (i.e. less privilege such as virtualized machines). As the victim in this case 2 is the cache L2 (IP 0), the blinding and masking evaluations considers the protection on its router.

Case 3 - Communication-based Attack for NoCs (NoC Timing Attack): IP 1 requests encryption services from IP 13. Meanwhile, IP 1 starts injecting dummy packets to IP 5 and monitors the transmission delay. Since there is an interaction between the shared cache L2 (IP 0) and the crypto-processor (IP 13), a delay in the transmission can be observed by the attacker when the shared cache replies (i.e. send the data asked by IP 13). This provides the attacker additional information (such as the duration of each encryption round).

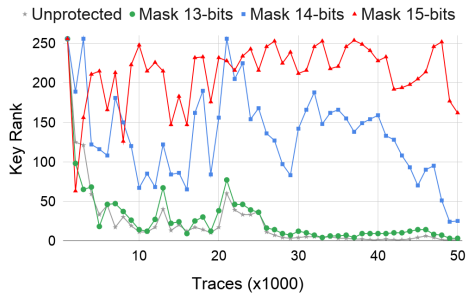
B. Security Evaluation

In this experiment, we analyze the security of the MPSoC for blinding, masking and dual switching schemes.

1) Blinding Analysis: Here, we evaluate the three blinding strategies. They are *average*, *bucket*, and *worst-case*. As blinding focuses only on protecting the nodes, only attack cases 1 and 2 are considered. Fig. 3(a) shows the results of blinding for the first attack case, i.e., the timing attack of Bernstein. The graph compares the rank evolution of the unprotected (grey line) against each countermeasure (colored lines). In the unprotected scenario, the rank decreases as more traces are provided. This means that the attacker is getting closer to guessing the correct key. However, for the bucket and worst-case blinding strategies the rank evolution results are not conclusive. The countermeasures can be considered

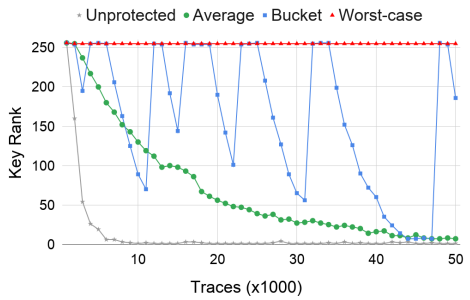


(a) Blinding



(b) Masking

Fig. 3: Key-rank analysis of Bernstein attack.



(a) Blinding

(b) Masking

Fig. 4: Key-rank analysis of Prime+Probe attack.

secure although their ranking analysis indicate that an attack may be theoretically possible (i.e., smooth decrease in the curves). In particular, the *average* blinding represents the weakest protection, as the key rank converges to a very low value; although it did not reach 1 it can be brute-forced.

Similarly, Fig. 4(a) shows the results of blinding for the second attack case, i.e., the Prime+Probe access attack. In this case, the attacker eliminates candidates after each encryption performed, and hence the key rank decreases much faster. However, since the countermeasures lead to misinterpretation by the attacker regarding the used cache lines, the Prime+Probe methodology can fail and the attacker could completely ignore the correct key. When this happened in the experiments, we restarted the processes starting from a rank of 255. The *average* blinding countermeasure was unsuccessful in avoiding the attack; it only made it harder when compared to the unprotected case. The other two countermeasures were able to successfully prevent the attack. The *bucket* countermeasure confuses the attacker in deciding which accesses resulted in cache misses and hits. As a result, the analysis restarted several times. In the *worst-case* blinding, all accesses were observed as cache misses and hence could not be used in the attack. Therefore, we can conclude that the *bucket* and *worst-case* countermeasures can fully prevent Prime+Probe attacks, while the *average* countermeasure only makes the attack harder.

2) Masking Analysis: Fig 3(b) shows the rank analysis of Bernstein's attack for three different masking configurations; they are 13-bit, 14-bit, and 15-bit. The masking strategy was only able to protect against the masking of 14 or 15 bits. Comparing with blinding, masking is a more powerful protection, but requires a high amount of noise to work properly. Hence, it will affect the overall performance more.

Fig 4(b) presents the rank analysis of all masking variations for attack case 2 (prime+probe). We evaluate the masking with lower values based on 6, 7, or 8 bits random noise. Similarly as observed for attack case 1, the randomization causes many difficulties to the attacker. The three options resulted in several restart operations in the Prime+Probe analysis. The portions of the graph where the rank is low, do not represent a threat. The attacker never knows when he is close to the result and when a false positive is reached. Therefore, we can conclude that the three solutions are secure and prevent Prime+Probe attacks. It is difficult to conclude if the smaller version (6-bit) is secure enough when compared to the other two options.

3) Dual-switching Analysis: The third attack case, i.e., the communication-based NoC attack, evaluates the countermeasure that protects the communication. In this attack scenario, the attacker performs an NoC timing attack to understand the communication behaviour (i.e., when accesses occur) between the secure node performing AES encryptions and the shared cache L2. Our tests on the unprotected NoC revealed that the attacker has a 100% accuracy in guessing when the accesses took place in a controlled environment (i.e., no other background traffic is used as this causing noise to the measurements). When background traffic is considered (i.e., 5% of the bandwidth used by third-party traffic in the attacked path), the attacker accuracy reduced to 45%. However, the work in [29] stated that an accuracy of at least 25% is enough to perform some logical attack successfully. The tests on Guard-NoC showed a drop in the accuracy to only 1% in a controlled environment. Therefore, Guard-NoC successfully avoids NoC timing attack by using the dual-switching countermeasure.

C. Performance and hardware overheads

Table I presents the performance, hardware area and power overhead of the countermeasures. We analyzed the performance overhead of all countermeasures using AES-128. These results, in addition to the security evaluation results, could be used by a designer to evaluate the best trade-off between security and performance/area/power. The performance overhead

TABLE I: Evaluation of Guard-NoC Countermeasures.

Performance Overhead					
Countermeasure		AES encryption	Countermeasure		AES encryption
Blinding	Average	2.77%	Masking	13-bit	6.7%
	Bucket	4.25%		14-bit	13.45%
	Worst-case	12.61%		15-bit	26.94%
Hardware Overhead					
Component				Area	Power
Guard-NoC Router				16%	18%
MPSoC with Guard-NoC				0.8%	0.9%

of the blinding schemes is marginal, except for the worst-case blinding countermeasure. However, this 12.6% overhead is still reasonable. The performance overhead of the masking depends on the size of the random noise source. However, as 14-bit (see previous subsections) is sufficient also the overhead of 13.45% is very acceptable. The area overhead is marginal as well and is calculated by adding the extra area of the obfuscation module to the router (see Figures 4 and 5). Note therefore that this is not the area overhead with respect to the whole platform. In case the whole MPSoC is considered, the overhead of the obfuscation module is less than 0.8%. Similarly, the power overhead is 0.9% of the complete platform. Note that this number is also pessimistic, as it assumes the case where the protection schemes are always running.

V. Conclusion and Discussion

This paper presented Guard-NoC, a secure NoC resilient against logical side channel attacks that protects MPSoCs. The countermeasures are especially desirable for platforms like servers, where resources are shared among different users. From our work we conclude the following:

Time-to-market: By embedding the security in the NoC, designers do not have to change/modify the nodes (IPs) and only focus on the design of the routers. This facilitates the integration process, since design companies create complex MPSoCs by employing several third-party components (IPs).

Security Updates: Security upgrades can be executed and evaluated more easily as only the routers have to be modified. For example, in case the system has to be protected against other threats such as malware, intelligent intrusion detection systems can be embedded in the routers.

Adaptive Security: During run-time, the system manager (often an operating system) could set different security policies for each node. For example, define *average* blinding for the non-sensitive processors and *mask 14-bits* for the AES processor. Ideally, the performance degradation of the countermeasures should affect only non-trustable applications.

System Protection: Contrary to previous Networks-on-Chip security solutions [15, 17], Guard-NoC is capable of not only protecting the intra-chip communication, but also the whole computational platform including processors, accelerators and memories.

Crypto-libraries: Guard-NoC has demonstrated that even a naive software implementation of AES encryption can be secure. Hence, the system can avoid complex software libraries for security, such as the ones mentioned in [37]; then saving performance and memory space.

Acknowledgments

This work was labelled by the EUREKA cluster PENTA and funded by Dutch authorities under grant agreement PENTA-2018e-17004-SunRISE.

References

[1] A. Vajda, *Multi-core and Many-core Processor Architectures.*, 2011.
[2] T. Kim and et al., "STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud," in *USENIX*, 2012.
[3] D. J. Bernstein, "Cache Timing Attacks on AES," Available at: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf, April 2005.
[4] C. Rebeiro, M. Mondal, and D. Mukhopadhyay, "Pinpointing cache timing attacks on aes," in *VLSID*, 2010.
[5] D. Stefan and et al., "Eliminating Cache-Based Timing Attacks with Instruction-Based Scheduling," in *ESORICS*, 2013.
[6] S. Crane and et al., "Thwarting Cache Side-Channel Attacks Through Dynamic Software Diversity," in *NDSS*, 2015.
[7] E. Brickell *et al.*, "Software mitigations to hedge AES against cache-based software side channel vulnerabilities." in *IACR Archive*, 2006.
[8] H. Yun and P. Valsan, "Evaluating the Isolation Effect of Cache Partitioning on COTS Multicore Platforms," in *OSPERT*, 2015.
[9] B. Ngabonziza and et al., "Trustzone explained: Architectural features and use cases," in *CIC*, 2016.
[10] M. Schwarz, S. Weiser, and D. Gruss, "Practical enclave malware with intel SGX," *CoRR*, 2019.
[11] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *USENIX*, 2016.
[12] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Appl. Soft Comput.*, 2016.
[13] M. Mushtaq and et al., "Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters," in *HASP*, 2018.
[14] C. Reinbrecht *et al.*, "LiD-CAT - A Lightweight Detector for Cache Attacks," in *ETS*, 2020.
[15] W. Yao and E. Suh, "Efficient timing channel protection for on-chip networks," in *NOCS*, 2012.
[16] J. Sepulveda *et al.*, "NoC-Based Protection for SoC Time-Driven Attacks," *IEEE ESL*, 2015.
[17] M. A. Kinsy *et al.*, "Hermes: Secure heterogeneous multicore architecture design," in *HOST*, 2017.
[18] B. Yang, K. Wu, and R. Karri, "Secure scan: A design-for-test architecture for crypto chips," *IEEE TCAD*, 2006.
[19] J. Da Rolt *et al.*, "New security threats against chips containing scan chain structures," in *HOST*, 2011.
[20] N. E. Jerger and L.-S. Peh, *On-Chip Networks*. Morgan and Claypool, 2009.
[21] A. Bogdanov *et al.*, "Differential Cache-Collision Timing Attacks on AES with Applications to Embedded CPUs," in *CT-RSA*, 2010.
[22] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *CRYPTO*, 1996.
[23] J. Bonneau and I. Mironov, "Cache-Collision Timing Attacks Against AES," in *CHES*, 2006.
[24] M. Andrysco and et al., "On Subnormal Floating Point and Abnormal Timing," in *IEEE SP*, 2015.
[25] C. Reinbrecht *et al.*, "Earthquake - A NoC-based Optimized Differential Cache-collision Attack for MPSoCs," in *DATE*, 2018.
[26] D. Osvik and et al., "Cache attacks and countermeasures: The case of aes," in *Topics in Cryptology - CT-RSA 2006*, 2006.
[27] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-channel Attack," in *USENIX*, 2014.
[28] D. Gruss, C. Maurice, and K. Wagner, "Flush+Flush: A Stealthier Last-Level Cache Attack," *CoRR*, 2015.
[29] C. Reinbrecht *et al.*, "Timing attack on NoC-based systems: Prime+Probe attack and NoC-based protection," *MICPRO*, 2017.
[30] O. Acıiçmez and Ç. Koç, "Trace-Driven Cache Attacks on AES," in *ICICS*, 2006.
[31] M. Gautschi *et al.*, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE TVLSI*, 2017.
[32] B. Köpf and M. Dürmuth, "A provably secure and efficient countermeasure against timing attacks," in *CSF*, 2009.
[33] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *DAC*, 2007.
[34] G. Xiao-chen and Z. Min-xuan, "Uniform Random Number Generator Using Leap Ahead LFSR Architecture," in *ACM CCS*, 2009.
[35] M. B. Stuart, M. B. Stensgaard, and J. Sparsø, "The ReNoC Reconfigurable Network-on-Chip: Architecture, Configuration Algorithms, and Evaluation," *ACM TECS*, 2011.
[36] D. Gullasch, E. Bangerter, and S. Krenn, "Cache Games - Bringing Access-Based Cache Attacks on AES to Practice," in *IEEE SP*, 2011.
[37] G. Irazoqui *et al.*, "Did we learn from LLC side channel attacks? A cache leakage detection tool for crypto libraries," *CoRR*, 2017.