Lightweight Security Mechanisms for MPSoCs

Anderson Camargo Sant'Ana
PUCRS - Porto Alegre, Brazil
anderson.santana.001@acad.pucrs.br

Henrique Martins Medina
PUCRS - Porto Alegre, Brazil
henrique.medina@acad.pucrs.br

Kevin Boucinha Fiorentin
PUCRS - Porto Alegre, Brazil
kevin.fiorentin@acad.pucrs.br

Fernando Gehm Moraes
PUCRS - Porto Alegre, Brazil
fernando.moraes@pucrs.br

ABSTRACT

Computational systems tend to adopt parallel architectures, by using multiprocessor systems-on-chip (MPSoCs). MPSoCs are vulnerable to software and hardware attacks, as infected applications and Hardware Trojans respectively. These attacks may have the purpose to gain access to sensitive data, interrupt a given application or even damage the system physically. The literature presents countermeasures using dedicated routing algorithms, cryptography, firewalls and secure zones. These approaches present a significant hardware cost (firewalls, cryptography) or are too restrictive regarding the use of MPSoC resources (secure zones). The goal of this paper is to present lightweight security mechanisms for MPSoCs, using four techniques: spatial isolation of applications; dedicated network to send sensitive data; traffic blocking filter; lightweight cryptography. These mechanisms protect the MPSoC against the most common software attacks, as Denial of Service (DoS) and spoofing (man-in-the-middle), and ensures confidentiality and integrity to applications. Results present low area and latency overhead, as well as the effectiveness of using the mechanisms to block malicious traffic.

CCS CONCEPTS

• **Security and privacy → Hardware attacks and countermeasures**;

KEYWORDS

MPSoC, attacks, security, spatial isolation, dedicated NOC, lightweight cryptography.

ACM Reference format:
Anderson Camargo Sant'Ana, Henrique Martins Medina, Kevin Boucinha Fiorentin, and Fernando Gehm Moraes. 2019. Lightweight Security Mechanisms for MPSoCs. In *Proceedings of 32nd Symposium on Integrated Circuits and Systems Design, Sao Paulo, Brazil, August 26–30, 2019 (SBCCI '19),* 6 pages.
https://doi.org/10.1145/3338852.3339876

1 INTRODUCTION

Computational systems tend to adopt parallel architectures, by using multiprocessor systems-on-chip (MPSoCs) devices [22]. These systems contain a large number of processing and storage elements, which can be interconnected by a Network-on-Chip (NoC). This trend in the semiconductor industry for using MPSoCs is mainly applied to applications that require processing power or massive interconnectivity of devices, e.g., the internet of things (IoT).

MPSoCs are vulnerable and can be attacked. According to Fiorin et al. [10] security attacks to an embedded system can be classified as software, physical or invasive, and side channel attacks (SCA). Thus, the MPSoC design requires security techniques to prevent potential attacks from intruders. These attacks compromise the integrity of data processed by the MPSoCs. Intruders can have different goals, from extracting information from the system to even unconfiguring or damaging the system.

According to Sepúlveda et al. [18], software-based attacks account for about 80% of attacks on embedded systems. Software attacks may come from infected applications or the operating system. Examples of attacks made from software include, but are not limited to: viruses, trojans, worms, denial of service (DoS), extraction of sensitive information, hijacking, and spyware.

Physical attacks also occur in MPSoC. An example of an attack reported in the literature are those made by Hardware Trojans (HT) [21] [14].

The attacks listed above, both software and hardware, may have the purpose of finding an encryption key used in the MPSoC to gain access to sensitive data. SCA obtain confidential information indirectly. SCA techniques include, for example, evaluation of the signature of the energy consumed in the system, the execution time of a particular application, and the electromagnetic radiation emitted by the system.

Due to the various types of attacks reported in the literature, having a system resilient to attacks becomes increasingly necessary for the electronics industry. As a consequence, several techniques have appeared aiming to increase the security level of Integrated Circuits (IC). Among the techniques used in hardware level are: secure zones [7], encryption [13] and firewalls [9].

The *goal* of this paper is to present lightweight security mechanisms for MPSoCs. These mechanisms have a small impact on the system performance and area, and protect the MPSoC against the most common software attacks, as DoS and spoofing (man-in-the-middle), and ensures to the applications confidentiality and integrity.

This paper is organized as follows. Section 2 presents related works in the field of security for MPSoC. Section 3 presents the

Table 1: Related works classification.

Proposal	Attacks	Countermeasures	Cost
Sepúlveda [19]	DoS and Timing Attack	adaptive routing and random arbitration	Power, Area and Performance
Caimi [7]	DoS, Timing Attack and Spoofing	Secure Zones	Area and Performance
Fernandes [8]	DoS and Timing Attack	Secure Zones and Cryptography	Performance
Ancajas [1]	HT	cryptography,package authentication, and task migration	Power, Area and Performance
Reinbrecht [17]	Distributed Timing Attack	Traffic Monitor and routing adaptive	Power and Area
Fernandes [9]	DoS and Hijacking	Firewalls	Area
Azad [2]	DoS and Spoofing	Firewalls	Area and Performance
This work	Memory integrity, DoS, Spoofing	Spatial isolation, filter and lightweight crypt.	Area and Performance

vulnerabilities in MPSoCs and the threat model. Section 4 details the reference architecture, without the security mechanisms. Section 5 details the main contribution of this paper, the lightweight security mechanisms. Section 6 presents the results and Section 7 draws the conclusions and directions for future work.

2 RELATED WORKS

Previous works [1, 17, 19] show that MPSoCs can be attacked at the software and hardware levels. These attacks are prevented using countermeasure techniques. Many countermeasures are explored in the state-of-the-art to add security to MPSoCs (Table 1). Most works show the implementation of a specific attack and its impact on the system, analyzing countermeasures' cost. Further, they present the area and power overhead to add safety to the system.

According to [7, 8, 20], secure zones is a technique that protects applications by reserving communication resources, computing resources, or both. At the computing level, processors are prevented from running tasks from different applications, preventing malicious applications from accessing local memory for information extraction. At the communication level, two main techniques are adopted. The first one is the creation of secure disjoint zones, where communication is protected by encryption [8, 20]. The second one is the creation of opaque secure (*OZS*) zones, where traffic not belonging to the *OZS* is deviated using rerouting mechanisms [6].

Most works related to security for MPSoCs use cryptography [1, 11, 15]. Encryption provides integrity, confidentiality, and authentication for sensitive information. Thus, using cryptography, secure communication channels are created [13]. Note that the encrypted data shares the communication links with other data flows. These flows can be malicious, seeking data through SCA. SCA can be mitigated by random arbitration on the router and adaptive routing of the NoC [19].

Another way to aggregate security in MPSoCs is to use firewalls to filter incoming and outgoing data according to the implemented security policy [2, 9]. These firewalls are strategically placed to protect Processing Elements (PEs) from various forms of attacks. The drawback of firewalls is the silicon area due to the memory required to store security policies.

Table 1 summarizes the main related works cited in this Section, positioning our proposal w.r.t. the state-of-the-art.

3 THREAT MODEL

The baseline architecture assumed in this work is a homogeneous NoC-based many-core, named MPSoC along this work. Each PE contains a 32-bit RISC processor, a NI (Network Interface) and a local memory accessed by the processor and NI module.

The resource sharing in MPSoC introduces vulnerabilities to applications running on it. It is possible to explore these vulnerabilities in attacks, such as confidentiality and integrity; timing attack; denial of service (DoS); spoofing; hijacking.

Confidentiality and integrity may be harmed when tasks belonging to different applications execute in the same processor. This is a common feature of systems where processors execute multitasking operating systems (OS). If a given task, when scheduled by the OS, has access to the memory contents of the processor, the attack becomes feasible.

Timing attacks are classified as side-channel attacks (SCA). Malicious applications may infer the contents of a given flow being transmitted through the NoC by statistical evaluation of the switching activity in the links [17].

Denial of service (DoS) is the most common attack in NoC-based MPSoCs, due to the shared links used to transmit the applications' flows. DoS attack may block part of the NoC by sending packets to destinations that are not able to consume the packets. Block a given processor by sending unexpected messages to a task (e.g. an interruption to deliver a message that was not expected by the processor); or disturb the latency of applications through a communicating intensive flow in the same path of an application having quality-of-service constraints.

An example of spoofing attack is the man-in-the-middle attack. For example, consider the scenario presented in Figure 1(a), where T_A sends messages to T_B, and a malicious task T_M is allocated. In Figure 1(b) the attack starts, with T_M sending a message to the processor executing T_A, which changes the address of T_B. After modifying the task location table, T_A sends messages to T_M instead of T_B, Figure 1(c). Consequently, T_M may extract confidential data by reading the message contents or send a different message to T_B. This attack was successfully performed on the reference platform

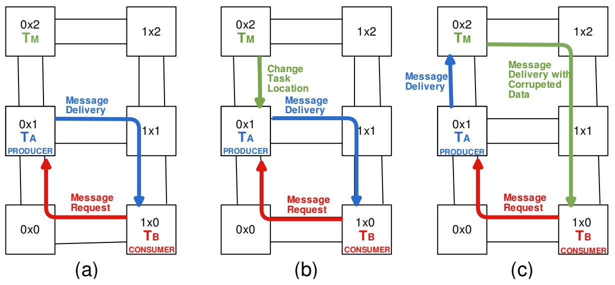

Figure 1: Example of man-in-the-middle attack. (a) T_A communicates with T_B. (b) T_M starts the attack (c) T_M has access to the flow's communication.

described in the next section, demonstrating the vulnerability of MPSoCs.

Another attack described in the literature is Hardware Trojans (HTs). HTs may extract sensitive information. For example, Prasad et al. [16] report an HT able to intercept packets, transmitting them to malicious applications. HTs may also interfere with the NoC performance, by injecting packets at a high injection rate to a given task.

Our proposal includes lightweight security mechanisms targeting the protection against most of the attacks above mentioned. It is out of the scope of this work to propose mechanisms protecting the NOC against DoS that interfere with the applications' performance neither attacks generated from HTs.

4 REFERENCE ARCHITECTURE

Figure 2(a) presents a 6x6 MPSoC instance, and Figure 2(b) the main components of the PE (Processing Element): a processor, a network interface with Direct-Memory-Access capabilities (DMNI), local memory and the NoC router.

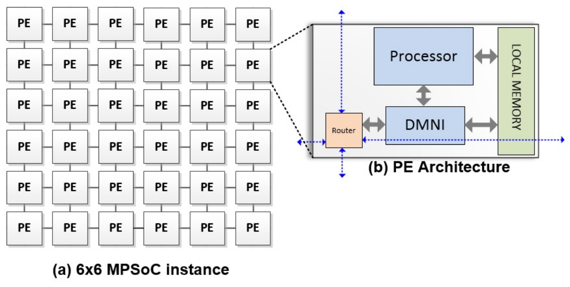

Figure 2: Baseline MPSoC architecture.

All PEs have the same hardware, being the differentiation made by software. Each PE may assume the following roles:

- Slave PE – *SP*: execute applications' tasks.
- Manager PE – *MP*: manage the SPs, executing functions such as application admission, mapping, remapping, DVFS control. Manager PEs only execute management functions.

A multi-tasking OS runs at each SP. The communication mechanism between tasks is the message passing, and a round-robin scheduler controls the execution time slice of each task. The OS supports two communication APIs: *serviced-oriented* and *raw*. The serviced-oriented mechanism leaves the charge of creating the packet structure to the OS. For example, at the task level is possible to write $SEND(*msg, task_T)$, and the OS encapsulates the message $*msg$ into a packet with all required fields (for example, $task_T$ address, packet size, the function of the packet, the message data). On the other side, the raw communication leaves the task the function to create the packet. This mechanism is useful for, e.g., communicating with external devices, as shared memories.

Assumptions related to secure components of the system include:

- The OS of each SP is not modified during its loading (integrity of the OS);
- The admission of new applications is also protected (integrity of applications), without modification in the object code [5];
- Applications do not have access to the *MP*. The manager processor only exchange messages with the OSs running in the SPs;
- There is no HTs in the MPSoC.

5 LIGHTWEIGHT SECURITY MECHANISMS

The proposal of this work is based on four lightweight security mechanisms:

a) spatial isolation of applications;
b) dedicated network for sending management messages related to the mapping and key distribution;
c) traffic blocking filter;
d) lightweight cryptography.

5.1 Spatial Isolation of Applications

The spatial isolation of applications avoids computation sharing at the processor level. The goal is to restrict the task mapping in such a way that tasks can only share the same PE *iff* they belong to the same application, thus preventing a malicious task to access a memory region reserved for another task or performing a DoS attack on the sensitive task. This mechanism ensures the integrity of the tasks memory contents.

Figure 3(a-b) shows different scenarios for mapping two applications and Figure 3(c) the applications' task graphs. In Figure 3(a) we have two distinct applications (app_1 and app_2) sharing the same SP ($0x1$). In this scenario, the malicious task, T_M, can access the local memory, or access the router, so it can steal information and perform an attack. In Figure 3(b), app_1 is mapped without sharing the processors with other applications, ensuring the spatial isolation of the application.

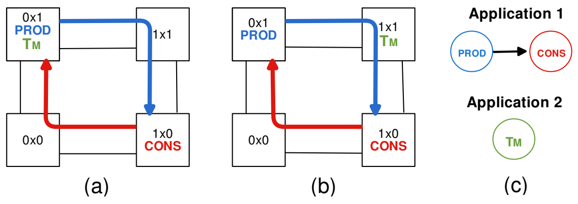

Figure 3: Mapping scenarios. (a) two applications are sharing the same PE (0x1). (b) app_1 mapped without sharing the processors with tasks belonging to other applications (c) applications' task graph.

The mapping heuristic runs on the MP. The implemented countermeasure has blocked the allocation of distinct applications in the same PE, thus avoiding direct access to memory by malicious tasks. A vector, where each index represents a processor of the MPSoC, assigns the index of the application when the processor receives a given task of an application. The mapping algorithm checks if the processor is free, otherwise it checks if the task to be mapped is different from the current application. If it is different, the mapping chooses another PE, and if it is the same, it can map on this same processor.

This countermeasure protects the integrity and confidentiality at the PE level. However, it is still possible to execute attacks such as DoS or man-in-the-middle. There is no hardware cost to implement this countermeasure as it is implemented in software.

5.2 Dedicated Network for Secure Messages

If the transmission of sensitive data to the SPs shares the NoC with flows belonging to general applications, the system security is vulnerable. Thus, to provide security to sensitive data, this work proposes the adoption of a simple dedicated NoC to send sensitive data to SPs. The adoption of a dedicated NoC isolates sensitive data from application data.

The dedicated network, named *HNoC*, is a serial Hamiltonian path that runs through all PEs, and only the MP may inject data into this NoC. Figure 4 illustrates the Hamiltonian path traversing a 3x3 MPSoC. The reasons to adopt this topology include: (*i*) small area footprint; 2 ports instead of 5 ports of a standard mesh; (*ii*) no need to add input buffers; (*iii*) the amount of data to be transmitted is small, justifying the serial path.

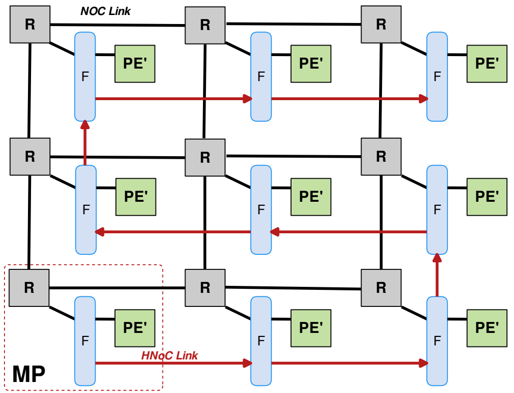

Figure 4: Hamiltonian path, in a 3x3 MPSoC. The *HNoC* adopts 32-bit links. R: router; F: filter; PE': DMNI/processor/local memory.

In the context of this work, the *HNoC* router is named *filter*. Its structure is discussed in subsection 5.3. It acts as a firewall, but its structure is much simpler than a standard firewall because the amount of information to store (secure policies) is small, reducing the *filter* area.

The MP may insert into the HNoC packets to set security policies, as cryptography keys and the applications' ID. The communication on HNoC is a two-step process. The first step is the injection of configuration packets to a set of *filters* with the operation that will be executed in the sequence. The second step is the payload data injection, that will be consumed by all *filters* waiting for data. The advantage of this multicast transmission comes from the fact that configuration packets are smaller than data packets. Thus, data packets are sent once to all *filters* waiting for the same data.

5.3 Traffic Blocking Filters

The *filter* is placed between the DMNI module and the router, as shown in Figure 5. The bottom part of the Figure corresponds to the Hamiltonian path. The incoming data is stored in a register (*reg_IN*), and the control logic (*CL*) evaluates the data contents. This data may be a configuration or a payload message. For a configuration message, *CL* evaluates the local filter address with the message contents. If they match, it means that next messages contain data to be stored in the internal registers (App_{ID} and App_{key}). The current *filter* implementation receives five payload messages: 4 to the 128-bit application key (App_{key}), and one for the application identifier (App_{ID}). If the upstream *filter* has consumed the previous message, the current message goes to the output register (*reg_OUT*), notifying the presence of a new message.

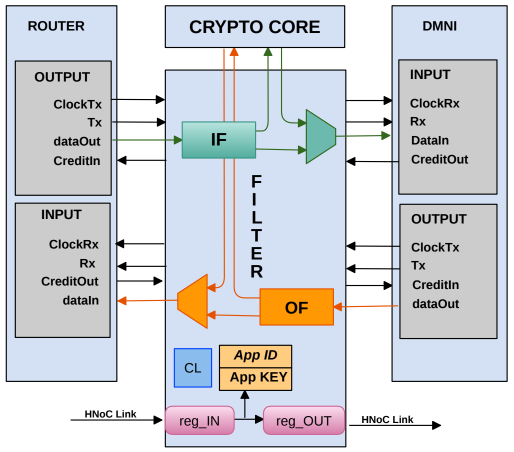

Figure 5: Filter used as HNoC router, to manage packet discarding, and connection to a crypto core. *OF*: output filter; *IF*: input filter; *CL*: control logic.

Both values, App_{ID} and App_{key}, are unique for each application. When a new application enters into the system, the MP generates unique App_{ID} and App_{key} values.

The top part of Figure 5 has two functions: block incoming or outcoming traffic and encrypt/decrypt the messages (next subsection). Any application, using the service-oriented or raw communication APIs must insert the App_{ID} in a given flit of the packet. Thus, two scenarios may arise. The first one is a task trying to forge the App_{ID} to execute an attack (as the previously described man-in-the-middle). In this situation, the packet is discarded in the *output filter* since the packet App_{ID} does not match with the contents of the App_{ID} register, and the malicious flow does not enter into the NoC. Note that the task that tried to inject the malicious packet does not perceive that the packet was discarded, and continues to act as the attack is happening. The second scenario corresponds to discard incoming flows. It is possible, for example, that an IO device tried to send packets to this filter. If the IO device forged successfully the App_{ID}, passing through the filter, the OS verifies if the packet comes from an IO device and if the task expected it. If not, it is discarded at the OS level and not at the filter level.

Coupling two simple mechanisms, spatial isolation of applications with traffic blocking filters, avoids DoS, spoofing and hijacking attacks.

5.4 Lightweight Cryptography

The crypto core is decoupled from the filter implementation, enabling the user to adopt different encryption mechanisms. The current work adopts two encryption methods: AES (IP core from [12], with a parallel implementation of the rounds) and a lightweight cryptography algorithm, SIMON [4], implemented according to its specification. Encryption algorithms, like AES, seek to achieve high levels of security, with little or no concern regarding power consumption and silicon area. The area of lightweight cryptography targets the security of devices where these features are limited. Released in 2013 by the National Security Agency (NSA), the Simon and Speck algorithm family aims to bring alternatives to the area of lightweight cryptography. Speck implementations were designed

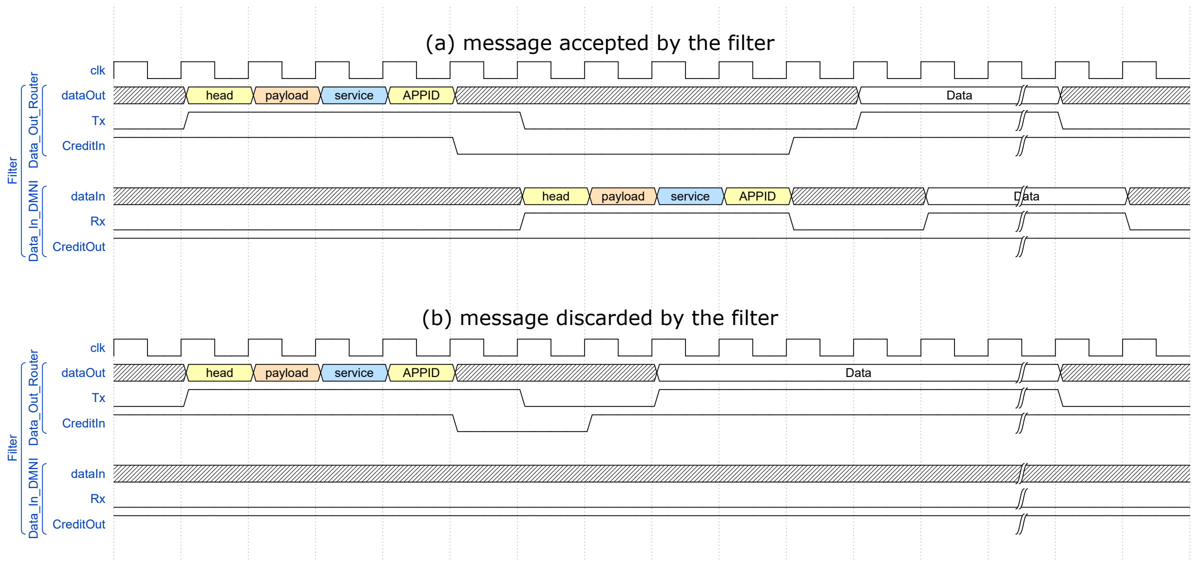

Figure 6: Waveform illustrating the operation of the filter blocking unwanted messages.

for better performance in software, while Simon is more efficient for hardware implementations [3].

Table 2 compares the performance of both encryption methods, by simulation and synthesis. Both IPs were implemented for a 128-bit key and a 128-bit block to cipher. The Table presents the latency to cipher one 128-bit block, the silicon area, and the dissipated power. As expected there is a trade-off between performance and area. The AES block is faster but presents a larger area. On the other side, SIMON, presents a small area, being suitable to be used in all *filters* without penalizing the PE area.

Table 2: Comparison between Simon and AES crypto cores - 65nm technology.

	Simon	AES
Latency (clock cycles)	140	19
Area (μm)	22,371	105,316
Power (μW)	16.033	399.233

According to the threat model Section, the use of cryptography protects applications for confidentiality, integrity, and SCAs. Confidentiality and integrity are served by the previous mechanisms, spatial isolation and the filter with the application identification. Thus, the main function of the cryptography is to protect the application against SCA. If the latency penalty of packet transfers can be accepted by the application, lightweight cryptography is the mechanism to be adopted.

6 RESULTS

This Section presents initially the actuation of the filter to block malicious traffic, then latency and area are evaluated. The MPSoC is modeled in synthesizable VHDL, and clock-cycle accurate simulation enables the performance evaluation using ModelSim tool.

6.1 Actuation of the Security Mechanisms

This section presents a scenario using three of the four security mechanisms: (*i*) spatial isolation of applications; (*ii*) dedicated network; (*iii*) traffic blocking filter.

Figure 6(a) presents a packet with a correct App_{ID} entering in the PE, while Figure 6(b) presents the action of the filter blocking a malicious packet. The upper four signals in both figures correspond to the router output port signals and the lower three signals to the output filter ports (Figure 5).

The beggining of the simulation in both scenarios is the same, the filter stores the first four flits (the fourth flit contains the App_{ID}), and the credit signal ($CreditIn$) goes to zero, interrupting the flits reception. At this moment, the App_{ID} is compared against the App_{ID} stored in the filter (sent through the HNoC during the mapping).

If a correct App_{ID} is received, the following actions occur: (*i*) the filter notifies valid data to the DMNI (signal $RX = 1$), and the first four flits are consumed by the DMNI (*dataIn* bus); (*ii*) the $CreditIn$ goes to $'1'$, and the remaining flits consumed.

If the App_{ID} does not match, the action executed by the filter is to assert the $CreditIn$ signal, without notifying the DMNI. This simple action corresponds to a virtual consumption, resulting in discarding any flow not targeted to the App_{ID}.

The filters block any attempt to inject packets with a wrong App_{ID}. However, as previously mentioned, peripherals attached to the NoC (as shared memories or hardware accelerators) may generate a packet with forged identification. Our current mechanism to discard such packets is a dedicated communication API, which is responsible for exchanging data with peripherals.

6.2 Latency results

This section evaluates the impact of the security mechanisms in the application latency, compared to the baseline MPSoC. The goal of the simulated scenarios is to define worst-case and typical latency overheads, using applications with a communication intensive profile (*prod_cons*, two tasks) and a computation intensive application (*MPEG*, five tasks).

The simulated scenario uses a 3x3 MPSoC, with a hop distance between communicating tasks equal to one. Due to the software stack to create and transmit the packets, and the corresponding functions to receive them, the number of hops has a negligible impact on the latency. Both applications run 50 iterations, and the average iteration latency is obtained from the average values from iteration 10 to 40, considering steady values.

Table 3 presents the average iteration latency for: (*i*) baseline MPSoC; (*ii*) MPSoC with filter and encryption disabled; (*iii*) MPSoC with AES encryption.

Results present distinct behaviors, according to the application profile. For the synthetic application (*prod_cons*), the average latency increased by 13.63% and 46.51% without and with the AES encryption, respectively, compared to the baseline MPSoC. The overheads in the average iteration latency of the real application (*MPEG*) are 0.1% and 2.55%. The reason for explaining such small overheads comes from the fact that most of the time the application is computing the frame decoding tasks, and not transferring data.

Table 3: Average iteration latency, results in clock cycles.

	baseline	with Filter	with AES
Prod_Cons	**1,877**	**2,133**	**2,750**
MPEG	**220,660**	**220,883**	**226,288**

These results expose the overhead of the proposal in terms of performance. Using the filter mechanism without enabling the encryption, the latency overhead varied between 0.1% (*MPEG*) and 13.63% (*prod_cons*), a small cost considering the increase of security added to the system. The cryptography protects the system against SCAs, and as shown for a real benchmark, the overhead was also small, 2.55%.

6.3 Area results

Table 4 illustrates the required area for the router, filter and both modules together. The filter corresponds to 16.33% of router area. From Table 2, the SIMON and AES cores correspond to 37.75% and 177.73% of the router area, respectively.

Table 4: Area consumption, CORE65GPSVT library (μm^2).

Instance	#Cells	Cell Area	Total Area
Router Buffer 16	5,906	43,735	59,255
Filter	1,229	6,638	9,677
Router+Filter	7,593	53,324	73,154

Thus, considering the adoption of the SIMON core, the overhead in the communication infrastructure is approximately 60%, a small cost considering the benefits of the approach.

7 CONCLUSIONS

The work presented in this paper added security to MPSoCs in a two-step process. During the application admission, the mapping heuristic applies spatial isolation, and a manager processor generates a unique application identifier and a cryptography key, transmitting them using a secure channel (dedicated NoC) to the PEs that will execute the application. Then, at runtime, a filter verifies the application identifier of all packets arriving or leaving the PEs. Further, if necessary, packets are encrypted using AES or lightweight cryptography (SIMON), protecting the traffic against SCAs. The cost of the approach is a small area (≈60% in the communication infrastructure) and a latency overhead smaller than 3% for a real benchmark, with and without cryptography.

As future work we can enumerate: (*i*) evaluate the overheads with real benchmarks; (*ii*) optimize the SIMON core to reduce its latency; (*iii*) add a DOS detection mechanism at the OS level by monitoring the reception latency; (*iv*) add new countermeasures when communicating with peripherals.

ACKNOWLEDGMENTS

Author Fernando Gehm Moraes is supported by FAPERGS (17/2551-196-1) and CNPq (302531/2016-5), Brazilian funding agencies.This paper was achieved in cooperation with Hewlett Packard Brasil LTDA. using incentives of Brazilian Informatics Law (Law nº 8.2.48 of 1991).

REFERENCES

[1] Dean Michael Ancajas, Koushik Chakraborty, and Sanghamitra Roy. 2014. Fort-NoCs: Mitigating the Threat of a Compromised NoC. In *DAC*. ACM, 158:1–158:6.
[2] Siavoosh Payandeh Azad, Behrad Niazmand, Gert Jervan, and Johanna Sepúlveda. 2018. Enabling Secure MPSoC Dynamic Operation through Protected Communication. In *ICECS*. IEEE, 481–484.
[3] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. 2013. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404. (2013). https://eprint.iacr.org/2013/404.
[4] Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. 2015. The SIMON and SPECK lightweight block ciphers. In *DAC*. ACM, 1–6.
[5] Luciano L. Caimi, Vinicius Fochi, and Fernando Gehm Moraes. 2018. Secure Admission of Applications in Many-cores. In *ICECS*. IEEE, 761–764.
[6] Luciano L. Caimi, Vinicius Fochi, Eduardo Wächter, and Fernando Gehm Moraes. 2018. Runtime creation of continuous secure zones in many-core systems for secure applications. In *LASCAS*. IEEE, 1–4.
[7] Luciano L. Caimi, Vinicius Fochi, Eduardo Wächter, Daniel Munhoz, and Fernando Gehm Moraes. 2017. Activation of secure zones in many-core systems with dynamic rerouting. In *ISCAS*. IEEE, 1–4.
[8] Ramon Fernandes, César A. M. Marcon, Rodrigo Cataldo, Jarbas Silveira, Georg Sigl, and Martha Johanna Sepúlveda. 2016. A security aware routing approach for NoC-based MPSoCs. In *SBCCI*. IEEE, 1–6.
[9] Ramon Fernandes, Bruno S. Oliveira, Johanna Sepúlveda, César A. M. Marcon, and Fernando Gehm Moraes. 2015. A non-intrusive and reconfigurable access control to secure NoCs. In *ICECS*. IEEE, 316–319.
[10] Leandro Fiorin, Cristina Silvano, and Mariagiovanna Sami. 2007. Security Aspects in Networks-on-Chips: Overview and Proposals for Secure Implementations. In *DSD*. IEEE Computer Society, 539–542.
[11] Catherine H. Gebotys and Robert J. Gebotys. 2003. A Framework for Security on NoC Technologies. In *ISVLSI*. IEEE, 113–120.
[12] Hemanth. 2004. *aes_crypto_core*. https://opencores.org/project,aes_crypto_core
[13] H. Isakovic and A. Wasicek. 2013. Secure channels in an integrated MPSoC architecture. In *IECON*. IEEE, 4488–4493.
[14] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. 2011. Introduction to differential power analysis. *J. Cryptographic Engineering* 1, 1 (2011), 5–27.
[15] Bruno Scherer Oliveira, Henrique Martins Medina, Anderson C. Sant'Ana, and Fernando Gehm Moraes. 2018. Secure Environment Architecture for MPSoCs. In *SBCCI*. IEEE, 1–6.
[16] N. Prasad, Rajit Karmakar, Santanu Chattopadhyay, and Indrajit Chakrabarti. 2017. Runtime mitigation of illegal packet request attacks in Networks-on-Chip. In *ISCAS*. IEEE, 1–4.
[17] Cezar Reinbrecht, Altamiro Amadeu Susin, Lilian Bossuet, and Johanna Sepúlveda. 2016. Gossip NoC - Avoiding Timin Side-Channel Attacks through Traffic Management. In *ISVLSI*. IEEE, 601–606.
[18] Johanna Sepúlveda, Guy Gogniat, Daniel Florez, Jean-Philippe Diguet, Cesar Zeferino, and Marius Strum. 2014. Elastic security zones for NoC-based 3D-MPSoCs. In *ICECS*. IEEE, 506–509.
[19] Martha Johanna Sepúlveda, Jean-Philippe Diguet, Marius Strum, and Guy Gogniat. 2015. NoC-Based Protection for SoC Time-Driven Attacks. *Embedded Systems Letters* 7, 1 (2015), 7–10.
[20] Martha Johanna Sepúlveda, Daniel Flórez, Vincent Immler, Guy Gogniat, and Georg Sigl. 2017. Efficient security zones implementation through hierarchical group key management at NoC-based MPSoCs. *Microprocessors and Microsystems - Embedded Hardware Design* 50 (2017), 164–174.
[21] Mohammad Tehranipoor and Farinaz Koushanfar. 2010. A Survey of Hardware Trojan Taxonomy and Detection. *IEEE Design & Test of Computers* 27, 1 (2010), 10–25.
[22] Hassan M. G. Wassel, Ying Gao, Jason Oberg, Ted Huffmire, Ryan Kastner, Frederic T. Chong, and Timothy Sherwood. 2013. SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip. In *ISCA*. ACM, 583–594.