UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CEZAR RODOLFO WEDIG REINBRECHT

Architectural Channel Attacks in NoC-based MPSoCs and its Countermeasures

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Altamiro Amadeu Susin

Porto Alegre
July 2017

CIP — CATALOGING-IN-PUBLICATION

Reinbrecht, Cezar Rodolfo Wedig

Architectural Channel Attacks in NoC-based MPSoCs and its Countermeasures / Cezar Rodolfo Wedig Reinbrecht. – Porto Alegre: PPGC da UFRGS, 2017.

146 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2017. Advisor: Altamiro Amadeu Susin.

1. MPSoC. 2. Hardware security. 3. Network-on-chip. 4. Side-channel attack. 5. NoC timing attack. I. Amadeu Susin, Altamiro. II. Título.

“The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards.”

— GENE SPAFFORD

GREETINGS

I would like to thank everyone who, in one way or another, contributed to the research and writing of this thesis for my doctorate graduation in the Post-graduation Program in Computation (PPGC) of the Federal University of Rio Grande do Sul (UFRGS).

To my advisor, Prof. Dr. Altamiro Amadeu Susin, for his dedication, friendship, guidance and for his vision of the future that inspires many of his students. In spite of my partial dedication, Prof. Susin put a great effort to provide me everything I needed to succeed in the doctorate. Also, Prof. Susin looked for research opportunities during my doctorate, which culminated in the partnership with Dr. Johanna Sepulveda.

To my co-advisor, Profa. Dr. Johanna Sepúlveda, who played a vital role in the development of this research. During the development of the thesis, Dr. Sepulveda has contributed in many ways: i) by defining the directions of the research of the thesis, which was derived by its original publication "NoC-based Protection for SoC time-driven Attacks "at IEEE Embedded Systems Letters; ii) by proposing and discussing the threat model, attacks and security countermeasures; and iii) by engaging my intership at the Embedded System Security and Hardware Architecture group from Hubert Curien Laboratory at University of Lyon (France), headed by her partner Prof. Dr. Lilian Bossuet. In order to support this collaboration she contributed by co-writing the granted project: "Development and evaluation of traffic management techniques to avoid side-channel attacks on network-on-chip based multi-processors-system-on-chip" supported by The CNPQ Brazilian scholarship (1.12.2015 - 31.07.2016). Besides, I thank Dr. Sepúlveda for all patience and support during the hard deadlines and experiments.

To the other teachers who guided me and taught me during the postgraduate course: Tiago Balen, Marinho Barcellos, Sérgio Bampi, Ricardo Reis, Luigi Carro, Alexandre Bonatto, Gabriel Nazar, Rafael Iankowski, and Everton Carara.

I would like to thank the fellows of scientific initiation who contributed greatly to the development of the work, and I quote here Bruna Carvalho, Bruno Forlin, Paulo Kipper, Pedro Portugal, Ana L. Brodt, and Jefferson Johner. I thank also the friends of Lapsi in the post-gratuation, Fabio Irigon, Igor Hoelscher, Tiago Waszak, and Luft.

To my friends, who felt my absence, but nevertheless they continued always supporting me.

To my mother and sister, Lisete Beatriz Wedig Reinbrecht and Stéfani Karine Wedig Reinbrecht, who are responsible for the person I am and for the values and princi-

ples that I bring with me. They always supported me and helped me in what I needed to move on and pursue my dreams. To my relatives,

To my beautiful wife, Paula Cardoso Rodrigues Reinbrecht, who was always by my side, acting as my inspiration, and given me courage to face all adversities. Thank you for supporting even in the moments of absence, of anguish and of stress. Together we achieved another important step, and from now on we will face many others. Thank you for being my half.

To all, my sincere gratitude!

ABSTRACT

Multi-Processors Systems-on-Chips (MPSoC) became the established hardware platform for a wide variety of applications and devices. Even more devices and systems will be interconnected by the Internet. The Internet link already brings several security concerns because all sensitive information stored on these devices can be reachable by external agents, and this prognostics will only increase the security issues. One of the most dangerous attacks is the Side Channel Attack (SCA). This type of attack explores features of the target system that reveals some secret or valuable data. This threat can be implemented physically through specialized instrumentation coupled directly to the device, or logical from architectural behavior accessed remotely through the network. The present thesis defines this particular logical SCA as a sub category called Architectural Channel Attack (ACA). This research project revised the bibliography to identify, analyze and explore the potential vulnerabilities of MPSoCs. The most vulnerable parts recognized were the shared cache and the Network-on-Chip (NoC). Within this knowledge, this thesis developed four new attacks aiming MPSoCs - Hourglass, Firecracker, Arrow, and Earthquake. Besides, the proposition that the hardware can provide security being transparent to applications resulted in a proposal of a hardware countermeasure, the Gossip NoC. The proposed attacks executed in a real MPSoC environment in an FPGA, breaking the Advanced Encryption Standard (AES). These evaluations were the first practical demonstration of an ACA performed in a NoC-based MPSoC entirely. The efficiency of different countermeasures, the Gossip NoC and three other ones from the literature, was evaluated under these attacks. Results showed that i) the shared cache and the NoC are critical vulnerabilities of complex MPSoCs; ii) the proposed attacks optimize the traditional cache ACAs found in literature making possible to attack even in limited environments; iii) the Earthquake makes the differential collision strategy feasible; iv) the NoC is a suitable candidate to implement security mechanisms, since it can access all elements in the system; v) the Gossip NoC avoids only one type of attack, but a protection mechanism for such complex systems demands multiple countermeasure strategies integrated to be a complete solution.

Keywords: MPSoC. hardware security. network-on-chip. side-channel attack. NoC timing attack.

Ataques de Canal Arquitetural em MPSoCs baseados em NoCs e suas Contramedidas

RESUMO

Sistemas em Chip Multi-Processados (do inglês, MPSoCs) tornaram-se a plataforma de hardware estabelecida para uma ampla variedade de aplicações e dispositivos. Cada vez mais dispositivos e sistemas serão interligados pela Internet. A conexão com a Internet já traz várias preocupações de segurança, porque todas as informações confidenciais armazenadas nesses dispositivos podem ser acessadas por agentes externos, e esse prognóstico só aumentará as questões relacionadas à segurança. Um dos ataques mais perigosos é o Ataque de Canal Lateral (do inglês, SCA). Este tipo de ataque explora características do sistema de destino (informação indireta) que revela alguns dados secretos ou valiosos. Esta ameaça pode ser implementada fisicamente através de instrumentação especializada acoplada diretamente ao dispositivo, ou lógica pelo comportamento arquitetural que é acessado remotamente através da rede. Esta tese define este SCA lógico em particular como uma subcategoria chamada Ataque de Canal Arquitetural (do inglês, ACA). Este projeto de pesquisa revisou a bibliografia para identificar, analisar e explorar as potenciais vulnerabilidades dos MPSoCs. As partes mais vulneráveis reconhecidas foram as Caches compartilhadas e a Rede-em-Chip (do inglês, NoC). Uma vez adquirido este conhecimento, esta tese desenvolveu quatro novos ataques para MPSoCs - Hourglass, Firecracker, Arrow, e Earthquake. Além disso, a proposição de que o hardware pode fornecer segurança sendo transparente para aplicações culminou em uma proposta de uma contramedida de hardware, o Gossip NoC. Os ataques propostos foram executados em um ambiente real de MPSoC em um FPGA, quebrando a criptografia AES. Estes experimentos práticos foram a primeira demonstração de um ACA realizado em um MPSoC baseado em NoC. A eficiência de diferentes contra-medidas foi avaliada sob estes ataques. Os resultados mostraram que i) a Cache compartilhada e a NoC são vulnerabilidades críticas em MPSoCs; ii) os ataques propostos otimizam os ACAs tradicionais de Cache encontrados na literatura; iii) o Earthquake torna viável a estratégia de colisão diferencial; iv) a NoC é uma candidata adequada para implementar mecanismos de segurança; v) a Gossip NoC evita apenas um tipo de ataque.

Palavras-chave: MPSoC, Segurança, Ataque de Canal Lateral, Rede-em-Chip, Sistema-em-Chip, Timing Attack, NoC Timing Attack, NoC Segura..

LIST OF ABBREVIATIONS AND ACRONYMS

3DIC	Three Dimension Integrated Circuit
ACA	Architectural Channel Attack
AES	Advanced Encryption Standard
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
ASLR	Address Space Layout Randomization
BP	Bad Positive
CAESAR	Competition for Authenticated Encryption: Security, Applicability, and Robustness
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
DES	Data Encryption Standard
DPA	Differential Power Analysis
DSP	Digital Signal Processing
DTA	Distributed Timing Attack
EM	Electromagnetic
EtM	Encrypt-then-MAC
FP	False Positive
FPGA	Field Programmable Gate Array
GCM	Galois Counter Mode
GPU	Graphical Processor Unit
HAL	Hardware Abstraction Layer
IC	Integrated Circuit
ICC	International Chamber of Commerce
IDE	Integrated Development Environment

IoE	Internet-of-Everything
IoT	Internet-of-Things
IP	Intelectual Property
IT	Information Technology
ITRS	International Technology Roadmap for Semiconductor
KSM	Kernel Samepage Merging
L1	Level one
L2	Level two
LLC	Last Level Cache
Malware	Malicious Software
MLS	Multi-Level Security
MPSoC	Multi-Processor System-on-Chip
NI	Network Interface
NIST	National Institute of Standards and Technology
NoC	Network-on-Chip
OS	Operating System
P+P	Prime+Probe
PA	Power Analysis
PUF	Physically Unclonable Functions
QoS	Quality of Service
RR	Round-Robin
SCA	Side Channel Attack
SER	Secure-enganced-router
SLAT	Second Level Address Translation
SNP	Spatial Network Partitioning
SoC	System-on-Chip

SPI	Serial Peripheral Interface
SSL	Secure Sockets Layer
TA	Timing Attack
TDM	Time-Division Multiplexing
TNP	Temporal Network Partitioning
TP	True Positive
TPS	Transparent Page Sharing
TSV	Through Silicon Via
UART	Universal Asynchronous Receiver Transmitter
VLIW	Very Large Instruction Word
VM	Virtual Machine
WFRL	West First Routing Logic
XOR	Exclusive-Or

LIST OF FIGURES

LIST OF TABLES

LIST OF CODES

CONTENTS

1 INTRODUCTION

Systems-on-Chip (SoC) is a computational system assembled in a single chip. SoCs are composed of software and hardware components, such as operating systems, processors, memories, accelerators (co-processors), among several elements (MEYR, 1997). SoC is the hardware platform of several types of applications, from conventional electronic devices (appliances) to high-performance servers. This complete solution inside the same silicon has fewer costs to the final product because it does not need extra components. Furthermore, few components result in fewer energy requirements, a key feature for the current technology trends, such as the mobile segment.

Although SoCs became a standard on electronic devices, the market has been increasing the demand for more performance, power efficiency and flexibility to run different applications. As a result, a new architectural concept was proposed as a promising platform, the Multi-Processor System-on-Chip (MPSoC) (CHEN et al., 2009). This system brings the potential of high parallelism as the novelty. MPSoCs are SoCs composed of several processing elements. These processors can be assembled with identical or different architectures, being homogeneous or heterogeneous respectively. According to ITRS 2015 (International Technology Roadmap for Semiconductor of 2015), by 2029 just the mobile segment will integrate over 300 elements, between application processors and graphical processor units (GPUs) (ITRS, 2016). Therefore, the MPSoC will improve the performance through the parallelism, software flexibility through the heterogeneity and power reduction through less external hardware requirements.

In the recent past, the Internet was mostly used as a means of collecting information and communications. Progressively it evolved into a commercial instrument, where the users could book services, acquire goods, pay bills and so on. Then, different types of cameras and sensors began to be connected to the Internet resulting in novel uses and applications. At present the Internet of Things allows objects to be sensed and controlled remotely across existing network infrastructure, creating opportunities for more direct integration between the physical world and computer-based systems and resulting in improved efficiency, accuracy, and economic benefit.

As the list of elements that can be connected to the Internet keeps on increasing a new term has been proposed: The Internet of Everything (IoE). The term IoE expands on the concept of the “Internet of Things” in that it connects not just physical devices but quite literally everything by getting them all on the network (CHANDHOK, 2014).

IoE works to connect more devices to the network, stretching out the edges of the network and expanding the roster of what can be connected. IoE has a major play in all industries, from retail to telecommunications to banking and financial services. However, placing all this personal information on the web or accessible via the web poses a severe challenge for security (KOMAR; EDELEV; KOUCHERYAVY, 2016). Almost every day we can hear news of hackers stealing valuable information, and it is imperative that appropriate measures are taken to prevent these problems, but despite this more and more people are willing to "venture" in the IoE since the benefits offered by the many capabilities provided by the IoE are overwhelming. Power consumption , electromagnetic radiation (EM) or hot spots Nowadays, Systems-on-Chip already is an object of attacks, and Side Channel Attack (SCA) is one of the most used techniques (KARRI et al., 2001). This type of attack employs features of the target system that reveals indirect information (leakage) about some secret or important data. This attack can be made physically through specialized instrumentation coupled directly to the device, extracting thermal (HUTTER; SCHMIDT, 2014), power (KOCHER; JAFFE; JUN, 1999; GEBOTYS; GEBOTYS, 2003; MASOOMI; MASOUMI; AHMADIAN, 2010; ORS et al., 2004) or electromagnetic (GANDOLFI; MOURTEL; OLIVIER, 2001) information. The collected information can be used to infer cryptographic keys, source code of proprietary software, digital certificates and other sensitive information (BAYON et al., 2012) (MORADI; MISCHKE; PAAR, 2013). New attacks have explored increasingly timing, access patterns, scheduling and faults to implement SCAs. These characteristics are not physical but logical from architectural behavior. This thesis defines this particular type of SCA as a subcategory called Architectural Channel Attack (ACA). Attacks that targets the behavior features are more suitable and effective for complex hardware systems, like SoCs and MPSoCs. The main reasons are:

- ACAs do not demand high specialized instrumentation;
- ACAs has no need to access the target device directly;
- High parallelism running on MPSoCs adds a significant noise for the physical measurements, such as power, EM, etc.;
- More structural complexity represent more potential leakage sources.

Cache memory is the most explored resource of SoCs for attacks, where timing, access pattern or trace are examples of leakage sources (BOGDANOV et al., 2010). Most cache attacks take advantage of the fact that the cache miss and cache hit are key-

dependent events with some cryptographic systems (SEPULVEDA et al., 2015). First cache attack proposals retrieved the key of different ciphers implementations, such as Data Encryption Standard (DES), RSA or Advanced Encryption Standard (AES) (KOCHER, 1996) (KELSEY et al., 1998) (TSUNOO et al., 2003) (BERNSTEIN, 2005) (OSVIK; SHAMIR; TROMER, 2006). Trace-based attacks are not considered in the scope of this research because they require a great level of information of the system, considered not possible (BOGDANOV et al., 2010). Therefore, we aimed access-based and timing-based cache attacks.

Regarding access-based attack, it infers the memory positions accessed during a cryptographic cipher operation by the time to access the data. One of the most efficient techniques is the Prime+Probe, proposed by Osvik et al. (OSVIK; SHAMIR; TROMER, 2006). In the same work, Osvik introduced the Evict+Flush and an asynchronous approach. The works of (XINJIE et al., 2008) (YOUNIS et al., 2015) (LIU et al., 2015) (OREN et al., 2015) optimized the access-based technique by Osvik or applied in different computational environments.

A timing-based attack observes the total execution time of a cryptographic cipher operation. The number of misses and hits of the cache result in a difference between latency responses. The timing attack proposed by Bernstein (BERNSTEIN, 2005) is a generic timing-based attack. The works of (NEVE; SEIFERT; WANG, 2006) (WEISS; HEINZ; STUMPF, 2012; WEISS et al., 2014) (SPREITZER; PLOS, 2013) (SPREITZER; GéRARD, 2014) (IRAZOQUI et al., 2014) optimized the timing-based technique of Bernstein. Then, some authors, like (BONNEAU; MIRONOV, 2006) and (BOGDANOV et al., 2010), proposed an optimization of the timing attack by manipulating the input data to provoke cache hits. This method is called collision timing attack. Bogdanov (BOGDANOV et al., 2010) presented a differential collision attack that explores the collisions generated by pairs of encryptions in sequence.

All the methods mentioned above was performed remotely to attack servers or remote computers. These attacks had to consider the target computer system and communication behavior besides the SoC architecture. Hence, the accuracy of the attack is compromised, due to communication and system interferences. Moreover, these attacks on MPSoC based devices face much more challenges, due to novel architectures features. One feature is the secure zone (SEPULVEDA; FLOREZ; GOGNIAT, 2015) (SEPULVEDA et al., 2014), where it is not possible to run a spy application in the victim CPU. The attack of Osvik uses the spy process strategy. Another architecture feature is the

usage of specific application IPs in the system. These specialized hardware functions are shared among several processors. Then, its behavior becomes not exclusive from the target process. For instance, a shared IP responsible for the network interface could increase the timing noise of remote protocols, like the Secure Sockets Layer (SSL) used by Bernstein's and Bogdanov's attacks. This work presents for the first time Architectural Channel Attacks implemented entirely inside the chip improving the attacks efficiency and decreasing data storage requirements.

Regarding attacks running inside the system, previous works considered that traffic patterns from Network-on-Chips (NoCs) could be exploited inside the MPSoC (YAO; SUH, 2012) (WASSEL et al., 2014) (SEPULVEDA et al., 2015). They mention that the NoC could leak information through the latency or throughput of the transfers. This ACA was called NoC timing attack. Each work also proposed a protection mechanism designed in the NoC architecture. Therefore, these authors did a remarkable contribution to the field of hardware security, pointing out the vulnerabilities and protection potentials of the NoC.

However, the state-of-the-art of ACAs did not comprehended attacks performed entirely inside the SoC or MPSoC, only describing the possibility of internal vulnerabilities. This research project has investigated the main ACAs related to cache and NoC inside MPSoCs to understand its feasibility and propose new threats. As a consequence, chapter six presents four developed attacks that breaks the AES cryptography. The first is the Hourglass, an adaptation of the timing-based attack from Bernstein, combined with a NoC timing attack. Then, the second and third are Firecracker and Arrow attacks, inspired in the Prime+Probe attack from Osvik integrated to a NoC timing attack. The fourth is Earthquake attack; that implements a differential collision attack with NoC timing attack.

The knowledge of MPSoCs vulnerabilities became possible to propose a security mechanisms. Chapter seven presents a secure enhanced NoC architecture, the Gossip NoC. This NoC targets sufficient protection with minimum area and performance penalties.

In summary, the present thesis sustains that MPSoCs, as the promising platform for future systems, have critical vulnerabilities in the shared cache and NoC, which can be worst if both are combined. Moreover, this thesis enforces that structures inside the NoC can implement some MPSoC security, and more research around this area should be addressed. Therefore, this research aims to deeply study ACAs in NoC-based MPSoCs, to develop and present with practical experiments MPSoCs vulnerabilities; and to show

that the NoC can perform protection mechanisms.

1.1 Motivation

Our project has technical, scientific and socio-economic relevance. Security in computer systems is a growing concern of society and the widespread use of SoCs in several electronic applications, such as smartphones and tablets, makes this issue even more critical.

The technical and scientific relevance rely on the fact that security is a key topic on enabling the electronics evolution. This subject is a target of recent studies of several research groups. Moreover, ACA has recently been proposed as a feasible attack, which leads to the fact that most devices are vulnerable. Our work in this topic will contribute to the scientific community with the first deep study of ACA implemented in MPSoCs and NoC-based protections. It is expected that the results of this project could be adopted by the national and international industry.

The socio-economic relevance stems from the importance of information security issue for national sovereignty, for businesses and citizens. A Recent article published in the portal Computer World, entitled "the Brazilian information security market has reached the significant milestone of US $ 1 billion investment", evidence such relevance.

1.2 The Thesis

The architecture behavior of shared cache and network-on-chip, isolated or combined, can reveal secrets of MPSoCs through Architectural Channel Attacks (ACAs); though protection mechanisms implemented in the NoC has the potential to avoid it through traffic management.

1.3 Objectives

The general thesis objective is to study ACAs in NoC-based MPSoCs, to develop and present with practical experiments MPSoCs vulnerabilities; and to show that the NoC can perform protections mechanisms. The following items are specific goals to achieve the general objective:

- To review the literature and identify potential threads for NoC-based MPSoCs;
- to develop an environment to run experiments and collect results;
- To adapt and perform state-of-the-art attacks on an MPSoC;
- To develop new attacks in NoC-based MPSoCs;
- To propose security mechanisms inside NoC architecture;
- To implement, characterize and validate the proposals under the attacks;
- To analyze results and present the final considerations.

1.4 Methodology

Four parts comprehend the methodology of the present thesis work. The first one referred to the research on Architecture Channel Attacks. The second part investigated protection strategies implemented in the NoC architecture. The third one was the development of an experimental environment to evaluate attacks and countermeasures. The final part evaluated the attacks and the protection mechanisms, resulting in a comparative analysis of the vulnerabilities and hardware security.

The ACA exploration started with a review of the literature where commonly cache memories were the primary target of logical attacks in complex computational systems. Another valuable information found in published works was that the NoC had become a potential source of leakage to attack MPSoC architectures (NoC timing attacks). Then, the NoC timing attack was analyzed to develop a complete workflow on how to perform such attack. At the same time, the understanding of the techniques from the state-of-the-art did culminate in four new attacks proposals.

The knowledge of the MPSoC vulnerabilities made possible to develop and evaluate countermeasures against most dangerous MPSoCs attacks. To be aligned with the thesis, the protection mechanism proposed was a NoC architecture, aiming a low area and power solution. It resulted in the Gossip NoC.

An MPSoC platform was developed to evaluate with a significant level of confidence the attacks and some countermeasures (the Gossip and others from literature). The environment runs entirely in hardware in an FPGA. It was developed an external software also to support compilation, boot sequence and execution interface.

Finally, the last step comprised all experiments to validate and extract metrics of the proposals. Besides, it was performed a comparison of state-of-the-art and the propos-

als. The analysis of the security mechanisms had shown the potential of the NoC as the leading actor in MPSoC security.

1.5 Structure of the Thesis

The present thesis is divided into eight chapters plus the conclusion and appendix. The chapter two describes the basic concepts required to understand the main topics of the thesis. It organizes the concepts in three fields: i) hardware security; ii) cryptographic engineering; and iii) Multi-processors Systems-on-Chips. Then, chapter three and four presents the state-of-the-art, where the first chapter focus on the architectural attacks, and the second on the countermeasures. Both chapters show works that explored vulnerabilities and protections in the cache and in the NoC of complex hardware systems. Chapter five introduces a study on the theme of the NoC timing attack. This type of attack was already cited by different authors ((YAO; SUH, 2012) (WASSEL et al., 2014) (SEPULVEDA et al., 2016) (SEPULVEDA et al., 2015) and (STEFAN; GOOSSENS, 2011)), but no detailed analysis and study has been performed yet. This chapter provides an overview of the technique and defines a threat model and methodology, to enable the replication of such technique. Some concerns about the attack are also presented and evaluated. After understanding the NoC timing attack, it is presented in chapter six the developed attacks of this thesis. The attacks focus on three side channel approaches: i) timing-based leakage; ii) access-based leakage; and iii) collision-based leakage. Chapter seven presents the secure enhanced NoC proposed. The architecture and functionality of Gossip NoC are explained in details. Different attacks and countermeasures are evaluated in chapter eight. The experiments were developed in a real MPSoC architecture running on an FPGA. The thesis ends with the conclusion chapter. The appendix presents the software development environment developed for the experiments with the real MPSoC, called MPSoC Glass.

2 BASIC CONCEPTS

This chapter presents the basic concepts regarding the field of the thesis. It comprises: i) security in hardware systems; ii) cryptographic engineering and iii) Multi-Processors Systems-on-Chips (MPSoCs). Security implemented in hardware is a recent approach. The principal motivations to embedded such mechanisms are presented. Regarding cryptographic engineering, several encryption technologies have allowed applications and devices to provide the suitable level of security. Considerations on performance and protection have driven such area. We present in details one of the most used encryption, the Advanced Encryption System (AES). The last part of this chapter introduces the Multi-Processors Systems-on-Chips. They are the architecture evolution of the System-on-Chip (SoC) concept. MPSoCs assemble several processing elements besides peripherals, specific purpose IPs, memories, and interconnection. Hence, his parallel nature achieves high performance, energy efficiency, and software flexibility.

2.1 Hardware Security

To implement security mechanisms directly in hardware is a recent trend. The primary motivations regard integrated circuit counterfeiting, secure cryptography (authentication, key storage, and generation, etc.), hardware Trojans, and Side Channel Attacks (SCA).

IC Counterfeiting: Fake or pirate chips have become a relevant issue for the integrated circuit industry. The counterfeiting and piracy for G20 nations were $923 billion to $1.13 trillion in 2013, and it was estimated $1.9 to $2.81 trillion in 2022 according to the International Chamber of Commerce (ICC) (ICC,). Current methods to detect counterfeit are performed during the test of the IC. However, it may not be effective against non-conventional types (e.g. cloned, overproduced, and tampered devices) (GUIN; DIMASE; TEHRANIPOOR, 2014). Recently, a different approach has been proposed, the application of Physically Unclonable Functions (PUF). The PUF uses manufacture variability to build a structure that responds with uniqueness for each chip. Each PUF receives a challenge as input, and outputs a unique response (BOSSUET; TORRES, 2017), providing a fingerprint for each chip. Then, the authenticity of the piece can be checked.

Secure Cryptography Besides the counterfeit problem, PUFs can solve the major concerns regarding authentication and secret key generation (SUH; DEVADAS, 2007). However, PUF is not the most recommended solution for this case. Given a challenge to a PUF, the response is always the same, but some errors may happen. The algorithm used to correct and check such response is computation intense, and it can be prohibitive for designs, whose application requires at run-time such correctness. Another approach has been established as the suitable solution, the authenticated encryption. This method performs the cryptography of huge blocks of data, generating the ciphertext and its respective tag. The tag has a correspondence with the encrypted data, being possible to authenticate the encryption. Six different authenticated encryption modes (namely OCB 2.0, Key Wrap, CCM, EAX, Encrypt-then-MAC (EtM), and GCM) have been standardized in ISO/IEC 19772:2009 (ISO/IEC,). So far, the most used authenticated encryption is the GCM (Galois Counter Mode). At the moment, a competition called CAESAR (Competition for Authenticated Encryption: Security, Applicability, and Robustness) aims to define a new standard, which will be implementable by software or hardware (MAIMUT; REYHANITABAR, 2014).

Hardware Trojans Hardware Trojans are malicious hardware elements inserted during IC manufacturing. Since the majority of ICs today have its manufacturing outsourced, this threat has appeared, and it has increased at a fast rate. There are already reports of trojans inserted in military equipment (ADEE, 2008). Trojans can change the functionality of an IC and affect the primary objective of the device, or even disable some function (TEHRANIPOOR; WANG, 2012). Another source of Trojan is the Intellectual Property component, which is a third-party hardware used to add some functionality to the project. Conventional test methods for ICs are unable to detect such undesirable elements because it can only check the expected functionality of the developed design (unless it is a soft-IP). A possible solution is to analyze the traces of current (or other physical features) to verify differences caused by the trojan. This technique is similar to the attack known as Side Channel Attack, that collects information through behavior tips.

Side Channel Attacks Attacks that uses leakage information from an implemented target hardware are known as Side-channel Attacks. It was first proposed in 1996 by (KOCHER, 1996), and since then, this technique has been used to extract cryptographic key material of encryption algorithms running on microprocessors, DSPs, FPGAs, ASICs and high-

performance CPUs (KOCHER; JAFFE; JUN, 1999; GEBOTYS; GEBOTYS, 2003; MASOOMI; MASOUMI; AHMADIAN, 2010; ORS et al., 2004; TSUNOO et al., 2003). Countermeasures in hardware include noise insertion, out-of-order memory access, data scrambling. However, most recent attacks have been exploring logical leakages, such as the timing behavior. Solutions by software have been proposed to avoid logical threats, like random allocation of processes in memory, out-of-order execution, dummy logic insertion, etc. The Side Channel Attacks is the main topic of this thesis. Thus next section details this subject, and proposes a new classification for the logical SCAs.

2.1.1 Side Channel Attacks

A public threat in cryptographic engineering is side channel analysis, or Side-Channel Attacks (SCAs) (BOSSUET; TORRES, 2017). SCAs are widely used as passive attacks because they make it possible to retrieve secret information (such as secret keys) with relatively few measurements and sometimes to use the inexpensive equipment. SCAs are non-invasive attacks, which means that the external agent only observes the device in normal operation without causing any physical harm. It can use information from sensors of power consumption, time delay or electromagnetic radiation. Typical targets of side-channel attacks are security ICs used in embedded devices or smart cards. Since these designs are specific purpose ICs, they are easier to analyze. In general, it is a simple device running a single process at a low to moderate clock frequency with direct access to the side-channel of interest and with (precise) control over synchronization and measurements. Examples of SCA techniques are:

- Power Analysis (PA) (GEBOTYS; GEBOTYS, 2003)(ORS et al., 2004);
- Differential Power Analysis (DPA) (KOCHER; JAFFE; JUN, 1999)(MASOOMI; MASOUMI; AHMADIAN, 2010);
- Electromagnetic Analysis (GANDOLFI; MOURTEL; OLIVIER, 2001);
- Temperature and Heat-fault Attacks (HUTTER; SCHMIDT, 2014); and
- Fault-based Attacks (KARRI et al., 2001).

Recently, some authors proposed the observation of the logical behavior from the ICs instead of the physical, since system actions lead to similar vulnerabilities. Firstly, the victims of these logical attacks were the System-on-Chips (SoCs), whose complex architecture created several sources of leakages. Commonly, most vulnerabilities remain

in the shared resources, optimized features, and increased functionality (TIRI, 2007). Examples of logical SCAs are:

- CPU Timing Attacks (TSUNOO et al., 2003)(ANDRYSCO et al., 2015);
- Cache Timing Attacks (KOCHER, 1996)(BERNSTEIN, 2005)(NEVE; SEIFERT; WANG, 2006)(PERCIVAL, 2005).
- Collision Cache Attacks (BONNEAU; MIRONOV, 2006)(BOGDANOV et al., 2010); and
- Access-based Attacks (OSVIK; SHAMIR; TROMER, 2006) (BENGER et al., 2014) (ZHANG et al., 2014) (IRAZOQUI et al.,) (GULLASCH; BANGERTER; KRENN, 2011) (YAROM; FALKNER, 2014).

The strategy of sharing resources aims to reduce the amount of hardware needed to implement a particular functionality, but it also creates opportunities to attackers. The main elements shared inside any SoC are the memories and the communication infrastructure. MPSoCs architectures follow the same concept, but on a more complex scale. Thus, it is expected that more source of leakages will be available for the attackers. MPSoCs have two primary shared resources, the memories, and the Network-on-Chip. Memory attacks already are a well-known SCAs. Recently, Networks-on-Chip attacks have been cited by (SEPULVEDA et al., 2012; SEPULVEDA et al., 2015), which considered draining, extraction of data and denial-of-service. It was proven that communication behavior could reveal patterns during application execution, which can be used to figure out sensitive information. NoCs are entering in the list of vulnerable components inside complex hardware systems. Therefore, attacks and countermeasures have to be investigated.

2.1.2 Proposed SCA Classification

Side Channel Attacks comprise a great variety of techniques. As described before, the main difference between each SCA is the leakage source. The leakage source defines the type of victim, the methodology employed, and the equipment required. The majority of SCAs explores physical features of devices. Application specific integrated circuits (ASICs) are the typical target of such attacks. However, the demand for more flexibility, performance, and less power has been increasing. As a consequence, complex hardware architectures have been proposed. This increase in complexity has brought challenges regarding the physical-based attacks, but opportunities to logical-based ones.

Therefore, since there are already several physical-based SCAs, and there is a potential emerging of several new logical-based SCAs, this thesis proposes an SCA classification. The proposed classification divide the attacks into two sub-categories by the leakage nature, physical and architecture. Table 2.1 organizes the current known SCAs according to the new subcategories.

Table 2.1: Proposed Side Channel Attacks Classification

Side Channel Attacks		Target			
Sub-Category	Leakage Source	Processor	Cache	NoC	ASIC
Physical Channel Attacks	Power	X			X
	Diferential Power	X			X
	Electromagnetic	X			X
	Temperature	X			X
	Fault	X	X		X
Architectural Channel Attacks	Timing	X	X	O	X
	Collision		X	O	
	Access		X	O	

The Xs presented in table 2.1 refers to the attacks described in the state-of-the-art. The Os are the contribution of this thesis in the state-of-the-art.

2.2 Cryptographic Engineering

The cryptographic engineering becomes a valuable tool for current technology. The increase of device integration brought all personal information to the digital world. As a result, these sensitive data has become a target of attacks. Therefore, different cryptographic algorithms have been used to protect and maintain users privacy. Different algorithms can be employed, depending on the characteristics of the target application, hardware support, and relevance of the information. The most popular solution is the Advanced Encryption Standard (DAEMEN; RIJMEN, 2002), introduced in 2001 by Vincent Rijmen e Joan Daemen in a competition of NIST (National Institute of Standards and Technology). Another great cipher is the RSA (RIVEST; SHAMIR; ADLEMAN, 1978). Due to hardware cost, RSA is used more frequently where it requires higher security, like bank terminals. Other cryptographic algorithms have been used in commercial applications, but considering the popularity, AES is the target study case of this thesis.

2.2.1 Advanced Encryption Standard - AES

Advanced Encryption Standard is the preferred cryptography in many (mainly commercial) applications. This symmetric encryption operates inputs of 128 bits and keys of 128, 192 or 256 bits. This cipher algorithm uses iterations, called rounds, to perform a series of linked operations. In the case of a key of 128 bits, it is completed in 10 rounds. These activities refer to a substitution-permutation network because it replaces inputs by specific outputs and then shuffles the bits.

2.2.1.1 Encryption Process

The input is a plaintext of 128 bits organized as a block of 16 bytes, represented as $Plaintext \rightarrow P_i$, where $0 <= i <= 15$. AES arranges this block as a matrix of four columns and four rows:

$$Plaintext = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix}$$

The same structure is applied to the key, for example, a 16 byte key is represented as $Key \rightarrow K_i$, where $0 <= i <= 15$:

$$Key = \begin{bmatrix} k_0 & x_1 & k_2 & k_3 \\ k_4 & x_5 & k_6 & k_7 \\ k_8 & x_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{bmatrix}$$

Before starting the encryption, it is performed the key expansion, which transforms the key in several keys to be used for each round, called subkeys. This process can be represented as $expanded(K_i) \rightarrow k_i^{round}$, where $0 <= i <= 15$ and $0 <= round <= 10$. Figure 2.1 shows this process. Then, four operations are executed at each round, with an exception on the last one (figure 2.1):

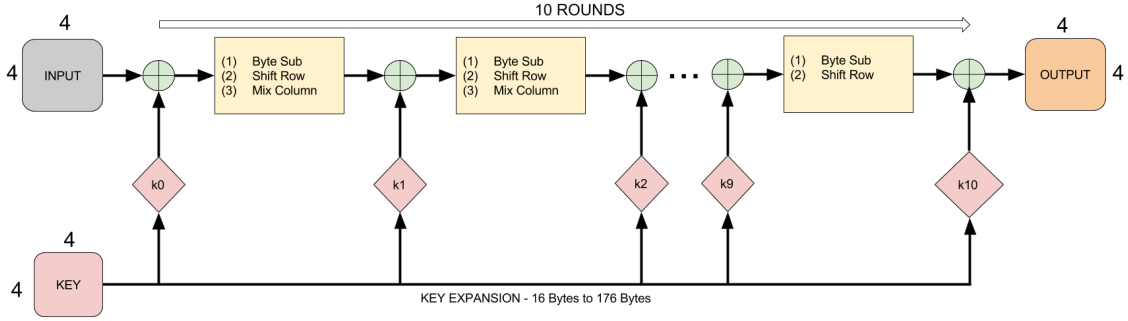
- AddRoundKey: The 16 bytes of the plaintext or intermediate state (x) are considered as 128 bits and are XORed to the 128 bits of the subkey (k^{round}). If this is the last round, then the output is the ciphertext. Otherwise, the resulting 128 bits are

interpreted as 16-bytes and continue with the following operations.

- SubBytes: The 16 input bytes are substituted according to a fixed table (S-box) given in design. The result is a new matrix of four rows and four columns.
- ShiftRows: Each of the four rows of the matrix is shifted to the left, in a circular manner (no data lost).
- MixColumns: Each column of four bytes is now transformed using a unique mathematical function. This function takes as input the four bytes of one column and outputs four entirely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

All these four operations can be represented as an iterative set of equations. Each equation presented at 2.1 represents the computation of each byte (of all 16 bytes) in the intermediate value. These intermediate bytes are iterated through these equations for ten rounds to accomplish the 128 AES algorithm. To calculate each part, the intermediate value from the current round is used as an index of the S-Box table, represented as the S. After changing its value, a circular shift operation is performed, where the number besides the S represents the amount of shift (01 is one shift left, 02 is two shift left, and so on). When everything is ready, one can compute the AddRoundKey operation from the next round to be ready for the next iteration. Considering this algorithm, before start with these equations the inputs must perform an AddRoundKey in advance.

Figure 2.1: AES-128 encryption diagram, representing the main operations executed over the iterative process of ten rounds.



$$
\begin{aligned}
(x_0^{r+1}) &\leftarrow 01.S[x_0^r] \oplus 01.S[x_5^r] \oplus 02.S[x_{10}^r] \oplus 03.S[x_{15}^r] \oplus k_0^{r+1} \\
(x_1^{r+1}) &\leftarrow 01.S[x_0^r] \oplus 02.S[x_5^r] \oplus 03.S[x_{10}^r] \oplus 01.S[x_{15}^r] \oplus k_1^{r+1} \\
(x_2^{r+1}) &\leftarrow 02.S[x_0^r] \oplus 03.S[x_5^r] \oplus 01.S[x_{10}^r] \oplus 01.S[x_{15}^r] \oplus k_2^{r+1} \\
(x_3^{r+1}) &\leftarrow 03.S[x_0^r] \oplus 01.S[x_5^r] \oplus 01.S[x_{10}^r] \oplus 02.S[x_{15}^r] \oplus k_3^{r+1} \\
(x_4^{r+1}) &\leftarrow 01.S[x_1^r] \oplus 01.S[x_6^r] \oplus 02.S[x_{11}^r] \oplus 03.S[x_{12}^r] \oplus k_4^{r+1} \\
(x_5^{r+1}) &\leftarrow 01.S[x_1^r] \oplus 02.S[x_6^r] \oplus 03.S[x_{11}^r] \oplus 01.S[x_{12}^r] \oplus k_5^{r+1} \\
(x_6^{r+1}) &\leftarrow 02.S[x_1^r] \oplus 03.S[x_6^r] \oplus 01.S[x_{11}^r] \oplus 01.S[x_{12}^r] \oplus k_6^{r+1} \\
(x_7^{r+1}) &\leftarrow 03.S[x_1^r] \oplus 01.S[x_6^r] \oplus 01.S[x_{11}^r] \oplus 02.S[x_{12}^r] \oplus k_7^{r+1} \\
(x_8^{r+1}) &\leftarrow 01.S[x_2^r] \oplus 01.S[x_7^r] \oplus 02.S[x_8^r] \oplus 03.S[x_{13}^r] \oplus k_8^{r+1} \\
(x_9^{r+1}) &\leftarrow 01.S[x_2^r] \oplus 02.S[x_7^r] \oplus 03.S[x_8^r] \oplus 01.S[x_{13}^r] \oplus k_9^{r+1} \\
(x_{10}^{r+1}) &\leftarrow 02.S[x_2^r] \oplus 03.S[x_7^r] \oplus 01.S[x_8^r] \oplus 01.S[x_{13}^r] \oplus k_{10}^{r+1} \\
(x_{11}^{r+1}) &\leftarrow 03.S[x_2^r] \oplus 01.S[x_7^r] \oplus 01.S[x_8^r] \oplus 02.S[x_{13}^r] \oplus k_{11}^{r+1} \\
(x_{12}^{r+1}) &\leftarrow 01.S[x_3^r] \oplus 01.S[x_4^r] \oplus 02.S[x_9^r] \oplus 03.S[x_{14}^r] \oplus k_{12}^{r+1} \\
(x_{13}^{r+1}) &\leftarrow 01.S[x_3^r] \oplus 02.S[x_4^r] \oplus 03.S[x_9^r] \oplus 01.S[x_{14}^r] \oplus k_{13}^{r+1} \\
(x_{11}^{r+1}) &\leftarrow 02.S[x_3^r] \oplus 03.S[x_4^r] \oplus 01.S[x_9^r] \oplus 01.S[x_{14}^r] \oplus k_{14}^{r+1} \\
(x_{15}^{r+1}) &\leftarrow 03.S[x_3^r] \oplus 01.S[x_1^r] \oplus 01.S[x_6^r] \oplus 02.S[x_{11}^r] \oplus k_{15}^{r+1}
\end{aligned}
\tag{2.1}
$$

2.2.1.2 Decryption Process

The decryption process follows the same algorithm. However, each step has to be made on the contrary. The expansion of the key remains the same. Then, the ciphertext (the input of this process) goes through the AddRoundKey step, but the first sum with the last part of the key (opposite way). The MixColumn and the ShiftRow also perform its operations in opposite way. In the end, the process outputs the plaintext recovered.

2.2.2 Performance-oriented AES

All operations performed by AES can be implemented using just logical and arithmetic operations. However, to obtain better performance, the cipher can be optimized for software implementations using a table with the operations pre-computed, as presented

in (DAEMEN; RIJMEN, 2002). The pre-computed operations comprise the execution of SubBytes, ShiftRows and MixColumns for all possibilities (entries of 0 to 255), resulting in four tables of 1 kB, called the T-tables (T_0, T_1, T_2 and T_3). There is one more table (T_4) for the last round that does not use the MixColumns operation.

The performance-oriented AES has two main phases. The first phase generates the subkeys by key expansion ($expansion(K) \rightarrow k^{round}$). Each subkey is used in the AddRoundKey step to provide the next round input matrix. Each byte of this input matrix is related to an index of the T-tables, where its content represents all operations performed for such byte. As a consequence, the output of the T-tables consulting are XORed resulting in a new output matrix, that can be represented as an intermediate state as follows x_i^{round}, where $0 <= i <= 15$ and $0 <= round <= 9$. In summary, each intermediate state is used for the next round computation, which executes a XOR with the next round subkey (AddRoundKey operation) and the accessed T-tables values (SubBytes, ShiftRows and MixColumns operations) generating the next intermediate state. This mathematical iterated operation can be observed in 2.2.

$$\begin{aligned}
(x_0^{r+1}, x_1^{r+1}, x_2^{r+1}, x_3^{r+1}) &\leftarrow T_0[x_0^r] \oplus T_1[x_5^r] \oplus T_2[x_{10}^r] \oplus T_3[x_{15}^r] \oplus k_0^{r+1} \\
(x_4^{r+1}, x_5^{r+1}, x_6^{r+1}, x_7^{r+1}) &\leftarrow T_0[x_4^r] \oplus T_1[x_9^r] \oplus T_2[x_{14}^r] \oplus T_3[x_3^r] \oplus k_1^{r+1} \\
(x_8^{r+1}, x_9^{r+1}, x_{10}^{r+1}, x_{11}^{r+1}) &\leftarrow T_0[x_8^r] \oplus T_1[x_{13}^r] \oplus T_2[x_2^r] \oplus T_3[x_7^r] \oplus k_2^{r+1} \\
(x_{12}^{r+1}, x_{13}^{r+1}, x_{14}^{r+1}, x_{15}^{r+1}) &\leftarrow T_0[x_{12}^r] \oplus T_1[x_1^r] \oplus T_2[x_6^r] \oplus T_3[x_{11}^r] \oplus k_3^{r+1}
\end{aligned} \tag{2.2}$$

The last round is computed by repeating the equation 2.2 with $r = 9$, except that $T_0, ..., T_3$ is replaced by T_4. The resulting x_i^{10} is the ciphertext.

2.2.3 Crypto-libraries

In this section, we also analyze the commercial implementations of AES in software. Three widely used crypto libraries are described, namely OpenSSL (PROJECT,), PolarSSL (POLARSSL,), and Libgcrypt (LIBGCRYPT,). All these AES solutions use the performance oriented approach. However, each one differs in the last round. Also, some of them do already contain methods to reduce or nullify cache-based side channel leakage.

OpenSSL: It performs the last round using a table T_4, where the S-box and the ShiftRow are previously computed.

PolarSSL: PolarSSL executes the last round using an S-Box table. It provides more security than OpenSSL since the granularity of such table is in bytes not words. The computation effort increases a little, mainly to perform the ShiftRow operation.

Libgcrypt: This library calculates the S-Box values used in the last round during the encryption. The timing leakage generated by cache access can be mitigated since there is no table, but a high computation effort is inserted. Depending on the sensitivity of the attacker, this methodology could not be secure.

2.3 Multi-processors Systems-on-Chip

New market demands, like high performance and low energy consumption, resulted in flexible platforms known as Multi-processors Systems-on-Chip (MPSoCs). MPSoCs are a complete system containing multiple processing elements on the same integrated circuit (WOLF; JERRAYA; MARTIN, 2008). Besides the processors, the system comprises co-processors, hardware accelerators, memories and a Network-on-Chip (NoC). NoCs are the usual interconnection solution to integrate several components, whose features could vary in router architecture, topology, etc. MPSoCs are the key enabler technology for new computation paradigms that demands high performance or energy efficiency. Fields that already apply MPSoCs are machine learning, IoT/IoE, high-bandwidth communication, augmented reality, and video encoding/decoding. The characteristics that define each MPSoC architecture are:

- the architecture model;
- the memory model;
- the communication model - Network-on-Chip; and
- the software architecture.

2.3.1 Architecture Model

The architecture model of an MPSoC is defined by the processing elements involved. If only the same type of architecture is used, the MPSoC is homogeneous. In this case, it is easy to program parallel applications and manage system resources. Besides, a homogeneous system can provide dynamic allocation and migration of tasks at run-time. However, some applications require specific hardware elements to obtain the proper performance or avoid extra computational effort.

Another approach integrates different architectures to perform a particular computation. The objective is to employ the suitable hardware for each application improving energy consumption. These elements can be processors (DSP, VLIW, general purpose, ...), hardware accelerators, co-processors, or application specific IPs. The heterogeneity affects the programmability directly, causing different constraints due to distinct instruction sets. Application Programming Interfaces (APIs) have been proposed to facilitate programmable issues overcoming such drawback.

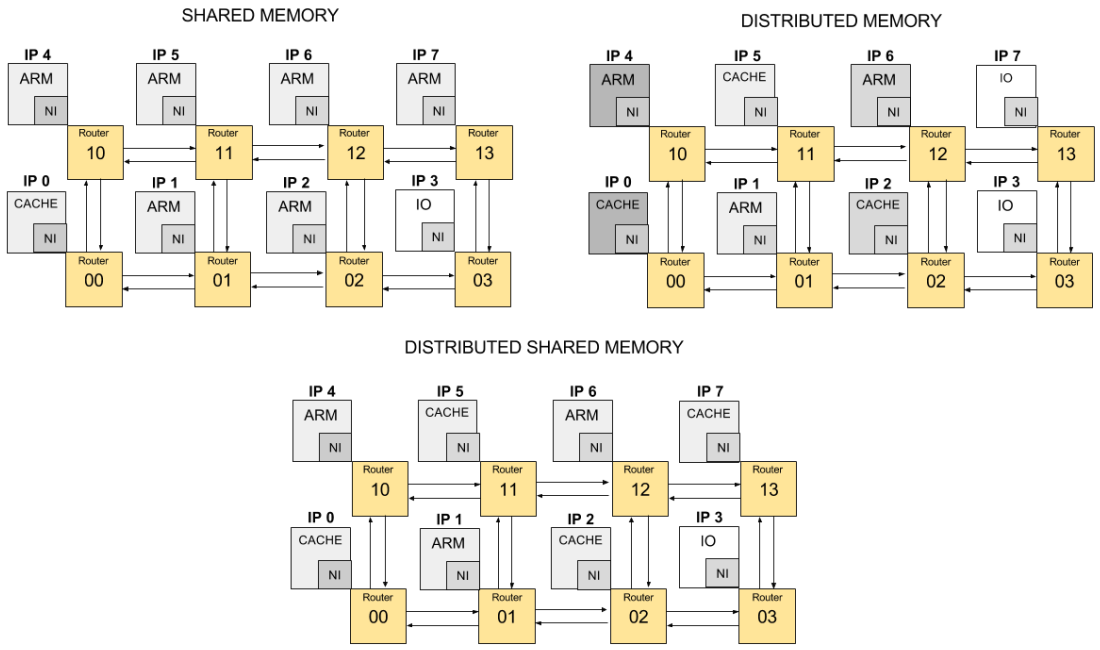
Recently, a new concept of architecture model has emerged. Processors with the same instruction set and different micro-architectures have been integrated. High-performance implementations are mixed with low power (low performance) ones. The applications run in the low power cores as default, and when necessaire the high performance works. This strategy stays in the middle of the homogeneous and heterogeneous concept. It enables easy programmability and energy efficiency since the high-performance core shut down when idle. An industrial example is big.little technology from ARM (ARM,).

2.3.2 Memory Model

MPSoC architectures are based on memory hierarchies. Several levels of cache can be integrated. Some of the cores may incorporate a processor and an L1 cache. When a cache miss occurs on L1, the cache coherency mechanism initiates an access to the shared L2 cache, located, usually, on another distant core. According to Girao et al. (GIRAO; BARCELOS; WAGNER, 2009), the memory hierarchy can be categorized as i) shared memory; ii) distributed memory; iii) distributed shared memory. Figure 2.2 shows these three strategies.

The shared memory is also defined as centralized. It has one memory element

Figure 2.2: Memory organization strategies in MPSoCs: Shared, Distributed, and Shared Distributed.



shared among all elements, typically the cache L2 or L3. This approach allows all processors to access the same address space, requiring an operating system to manage the accesses. The problem is the bottleneck created since the memory is a frequent use resource.

The distributed memory implements several memories in different nodes of the NoC. Each memory is exclusive for each processing element. Hence, the address space is not shared, and the data can be isolated between processes. However, the interaction between processes requires intense communication.

The third approach implements several memories spread in the NoC, but with the address space shared. Thus, the processors can access different memories, and use them to synchronize the tasks. The drawback consists on memory synchronization since all processors must be informed when data has been updated.

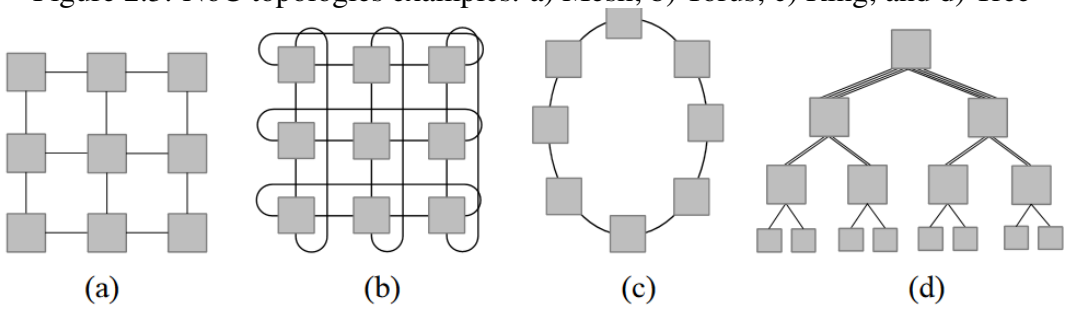
2.3.3 Communication Model - Network-on-Chip

The Network-on-Chip is the proper interconnection for MPSoCs architectures. The main benefits are the scalability, high bandwidth, and design reusability. By employing a set of routers and links, the NoC transmits packets between a pair of source IP

(which injects the packet) and destination IP (which receives the packet). The network interface (NI) is the part that links a core to a router. The NI implements the communication protocol by packing and unpacking the data and controlling the data injection and ejection to/from the NoC. Intra-chip communication structure has a large impact on overall system performance, and area and power costs. It has a central role in the system, with high influence among all elements, which makes the NoC also attractive to implement security mechanisms. There are several proposed NoC architectures in the literature. Each proposal exploits different characteristics, described as follows.

Topology: The way the routers are arranged is defined as the network topology. Traditional topologies are the mesh, torus, ring, and tree (figure 2.3). The mesh is the most used organization, being a homogeneous structure. Each router is connected to four adjacent routers (with exception to the corners). Each port connection is identified as north, south, east and west plus the local connected to the node.

Figure 2.3: NoC topologies examples. a) Mesh; b) Torus; c) Ring; and d) Tree



Nowadays, MPSoC architectures have also been employing hierarchical topologies. Different levels of communication are defined in design time, organizing the cores in groups, also known as clusters. Hence, the inter-cluster communication can differ from the intra-cluster communication. The most common strategy uses bus-based connection inside the cluster, and a mesh NoC to connect them. The reason to exploit the hierarchical strategy is the communication locality from the applications. As a consequence, the costs of the interconnection architecture can be reduced without compromise performance.

Routing: Any communication in the NoC has the node that sends a message (source) and the node that receives it (destination). Both have a unique network address that it is used to define the path of the packets. The process that determines which path the message will take is known as routing. The routing algorithm can follow four strategies: i) arithmetic,

ii) source-based, iii) table-based, and iv) adaptive.

The arithmetic routing algorithm follows a fixed rule, computed at each router. For example, the XY algorithm always routes the message to the X axis first (horizontal), and then routes in the Y-axis (vertical). Hence, each router computes the difference from the source to the destination to define the route. As a result, the routers requires an arithmetic logic inside to calculate the routes. The source-route defines all message route at the source, typically by the network interface. However, a centralized manager or centralized rule is required to avoid routing issues, like deadlock or starvation. The table-based uses tables at each router, where depending on the target, the table defines the output port. It is a very flexible approach but consumes much more area. The adaptive can route through more than one path possibility. The normal condition to choose a different path is the network congestion. When an output port from a router is blocked, the router can change it to another route. Any routing strategy can also define its algorithm based on a non-deterministic technique (e.g. random number generator). This approach tries to avoid channel leakages in the communication, creating an unpredictable scenario.

Switching Mode: Two switching modes can be implemented in a NoC, the circuit switching, and the packet switching. The circuit uses the analogy of a closed circuit to transfer the data. This concept means all routers must be configured previously from the source node to the destination node. The circuit switching requires a distinct packet or protocol to close the circuit before any transmission. The benefit is that the maximum throughput can be achieved since no congestion will take place after the configuration. The drawback regards the latency inserted, due to the circuits blocks a whole path.

On the other hand, there is the packet switching. In packet switching, the message is split in small parts, defined as flits, and each flit travels separately in the NoC, following the header. The switch of the routers is configured only when the header pass. In this strategy, the paths are set during the transmission, which results in less latency. However, if congestion is met, the throughput is penalized.

Time-division-multiplexing: An important feature regarding traffic congestion managing is the time-division multiplexing (TDM). Using TDM is possible to create the concept of virtual channels. In this technique, the physical link is shared by different messages, organized in time slots. Each time slot corresponds to a virtual port on the router. The router requires this TDM manager, and one input buffer separate for each virtual channel

to support this strategy. The benefit regards the avoidance of congestion and deadlocks. The drawback concerns the area overhead because buffers are the most expensive resource inside a router.

Flow Control: A flow control has to be implemented to synchronize the transmissions between routers. A well-defined protocol is used to inform when a transmitter wants to send, and when a receiver can receive. The primary objective is to avoid loss of data in the network caused by congestion (buffers full), or duplication of data (second capture). The typical protocols used are the handshake (valid/ack protocol), and the credit-based.

Technology: Recently, different semiconductor technologies have been proposed to modify the way we know to integrate the hardware components. One technology is the 3D Integrated Circuits (3DIC). Various integrated circuits are stacked during manufacture. The 3DIC provides a vertical link, known as Through-Silicon Via (TSV) that communicates with the different dies. Consequently, more hardware can be integrated into the same area, and the TSVs enable their communication. As a result, 3D NoCs have been proposed in literature obtaining good results.

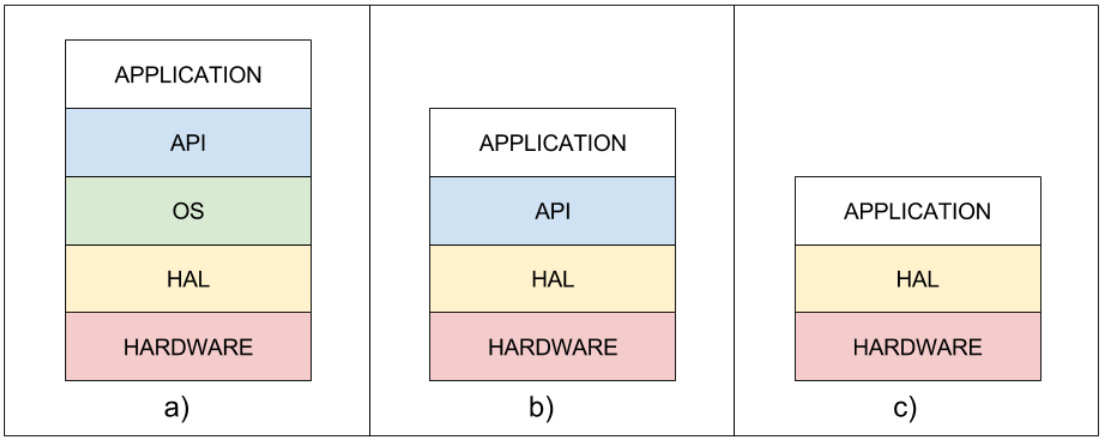
Another concern in the semiconductor industry is the scaling of the wire in high-end technologies. The scaling factor of the transistor and the wire became different, and now the IC limitation is defined by the interconnections (wires). A new process has been proposed to attenuate the electrical wire concerns in current designs, the silicon photonic. Silicon photonic presents an alternative to building logical links without electrical concerns, increasing the performance, and reducing the power. The work in (ORCUTT et al., 2011) presents a methodology to manufacture photonic elements together with conventional CMOS technology monolithically. This feature allowed one to design architectures with these two technologies in the same die. As a result, this technology becomes transparent to the design step and manufacture.

2.3.4 Software Architecture

The software architecture is the structure required to run applications on the target hardware platform. According to (JERRAYA; WOLF, 2004), it comprises three parts: i) the Hardware Abstraction Layer (HAL); ii) the Operating System (OS); and iii) the Application Programming Interfaces (APIs). These possibilities are represented at Figure

2.4.

Figure 2.4: Software Architectures. a) Full; b) Partial; c) Bare-metal.



The HAL performs the configuration of all hardware components, being responsible for initializing the hardware components, and boot the system to a bare-metal application (without OS) or an OS. If the system uses an OS, specific functions of the hardware may be provided by APIs. The API translates complicated system tasks to simple functions. Hence, applications can use better the hardware features. Examples of APIs already implemented for MPSoCs are the OpenMP, MPI, OpenCL, OpenCV and MCAPI (ROSA, 2016).

2.3.5 MPSoC Examples

A summary of the MPSoCs found in industry and academy is shown in table 2.2. This overview was presented in (ROSA, 2016), and shows the main characteristics of each one. The last architecture regards to our reference architecture, MPSoC Glass.

2.3.6 Reference Architecture - MPSoC Glass

The reference architecture is the MPSoC Glass, a heterogeneous MPSoC interconnected by a mesh NoC using a shared memory organization. The structure and components can be observed in figure 2.5. MPSoC Glass uses the NIOS II processor from Altera. In the same manner as presented by ARM with big.Little technology (ARM,), our architecture uses different NIOS II implementations, the economy core (NIOS II/e

Table 2.2: Summary of MPSoC architectures.

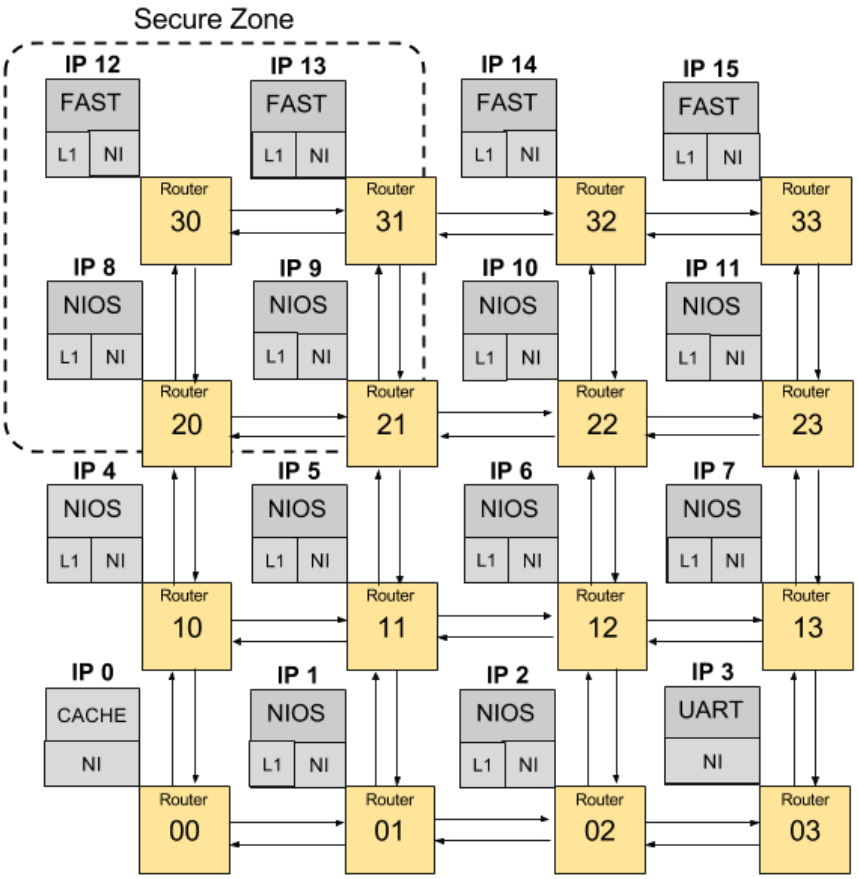
	Target Application	Memory Organization	Communication Organization	Application Programming
Cell	Gaming	Shared	NoC (Ring)	-
Tomahawk	Communication Multimedia	Shared	NoC	C-based #pragmas
Faust	Software Defined Radio	Distributed	NoC	Data Flow
Magali	Software Defined Radio	Distributed	NoC	Data Flow
X-GOLD SDR20	Software Defined Radio	Shared	Bus	-
COBRA	Software Defined Radio	Shared	Crossbar Bus	MPA tool
Flextiles	General Purpose	Distributed Shared	NoC	Thread-based
P2012	General Purpose	Shared	NoC	-
Glass	General Purpose	Shared	NoC	IDE C language

from (ALTERA,)), and the fast core (NIOS II/f from (ALTERA,)). The NIOS II fast core aims high-performance applications, system management and primary tasks of the system. Minor tasks and parallel applications are intended to be executed by a group of NIOS II economy cores. The NIOS II economy core has a low area and power, being suitable to be replicated many times in an MPSoC. Consequently, the concentration of the economy core in the MPSoC is much higher than the fast cores, being four fast NIOS II and ten economy NIOS II. Other components integrated into the MPSoC Glass are a shared cache, and an UART interface resulting in a total of sixteen parts.

The memory organization has two cache levels in the hierarchy. The level 1 (L1) is a local instruction and data direct mapped cache. The level 2 (L2) is a shared cache, 16-way set associative. The shared cache L2 has a direct link to the external main memory. The sizes of the caches and cache lines are parametrized for each design. Depending on the target usage, a different setup can be set.

The NoC is composed by a 5-port router, with input buffers of eight flits, each flit of 32 bits. The topology is mesh, 4x4, and it uses the XY as the routing algorithm. Also, MPSoC Glass has a built-in security zone, defined at design-time. Elements inside secure zones are considered trustful among them because their communication are exclusive (SEPULVEDA; FLOREZ; GOGNIAT, 2015) (SEPULVEDA et al., 2014). The

Figure 2.5: Reference Architecture - MPSoC Glass. Four fast NIOS II core, ten economy NIOS II core, one UART interface, and one shared cache memory.



processors inside the secure zone are not allowed to download or run untrusted software. Others (processors outside secure zone) can execute any application, even external ones downloaded by the system. However, some features of the NoC architecture can change based on design constraints.

The software architecture is simplified for the first version of MPSoC Glass. Each processor has an HAL (Hardware Abstraction Layer) to initialize the necessary components, and a bare-metal code that runs the tasks directly in the HAL. A developed library set provides all communication features through the NoC and crypto-functions as services, being similar to an API solution. An IDE (Integrated Development Environment) was developed to automate parts of the development flow, such as compilation and upload of the binaries in the several processors of the system. Information regarding the library and the MPSoC Glass IDE are described in the Appendices of the thesis. Future works aim the improving of MPSoC Glass IDE and the integration of Operating System and commercial APIs.

2.3.6.1 Hardware Costs

The Glass MPSoC was implemented in a Cyclone IV GX from Intel FPGA (Altera). According to synthesis, the system can reach 120MHz as the maximum operation frequency. Results of area and power can be observed by table 2.3.

Table 2.3: FPGA Cyclone IV GX synthesis results of the MPSoC Glass components.

	Logic	**Registers**	**Power (miliwatts)**
NIOS II/e - Economy Core	**590**	**299**	**35,62**
NIOS II/f - Fast Core	**3002**	**2292**	**101.9**
Processor NI	**660**	**646**	**14.99**
Timer	**213**	**216**	**4.48**
Cache	**5412**	**677**	**405.38**
Cache NI	**113**	**184**	**20.33**
UART	**49**	**33**	**0.8**
UART NI	**190**	**228**	**4.55**
One Single Router	**1506**	**685**	**38.49**
4x4 NoC (Routers and Links)	**22423**	**7418**	**643.44**

2.4 Considerations

This chapter has presented the essential knowledge to understand the area of hardware security. The Advanced Encryption Standard (AES) was explained in details to elucidate the concepts used by the following chapters. Besides, considerations about AES libraries. The last part gave an overview about MPSoC architecture, describing the main elements that compose these systems.

About the MPSoC overview, it is important to reinforce that the examples found in industry and academy already has established that this kind of platform will be heterogeneous with a NoC as the communication structure. However, solutions in the application layer, like APIs and operating systems are still an open issue, where no standard has been widely accepted. Therefore, the research object of this thesis, about MPSoC vulnerabilities and protections, must keep tracking the progress on the application layer of MPSoCs, adapting the context when necessary.

3 STATE-OF-THE-ART

This chapter introduces the state-of-the-art in the field of architecture channel attacks. This survey aims to put in evidence only the works that can apply to MPSoCs. Regarding MPSoC vulnerabilities, the shared resources are the primary targets of the architectural-channel attacks. Thus, only cache and NoC attacks are presented.

This chapter shows the state-of-the-art regarding the countermeasures against ACAs in MPSoCs. The main victim's of ACAs in MPSoCs are the shared cache and the NoC. Consequently, this chapter presents only the countermeasures for these components. To avoid cache attacks, the main practical techniques found in literature are software-based. The software solutions explores modifications in the crypto-library, the compiler or the operating system. Regarding NoC protection mechanisms, only hardware-based approaches have been proposed.

3.1 Cache Attacks

Regarding cache attacks, it is possible to explore three strategies. One regards the memory accesses during the encryption, called access-based cache attack. Another approach is to study the time that cache inserts on the computation of a task, called timing-based cache attacks. The last one analyses the cache transactions, defined as trace-based attacks. Trace-based attacks are not considered in this overview since they require a significant level of information of the victim.

3.1.1 Timing-based Cache Attacks

Kocher (KOCHER, 1996) and Kelsey et al. (KELSEY et al., 1998) first mentioned Timing-based cache attacks. According to Kocher, cryptographic algorithms running on a platform always present timing leakages that can be used for performing ACAs. Tsunoo et al. (TSUNOO et al., 2003) present for the first time a practical timing attack on caches, breaking the DES algorithm. This work opened a new front on ACAs.

Bernstein (BERNSTEIN, 2005) adapts Tsunoo's attack to break AES cryptography. Neve et al. (NEVE; SEIFERT; WANG, 2006) provides implementation details regarding the attack of (BERNSTEIN, 2005) and proposes an extension to it. Weißet

al. (WEISS; HEINZ; STUMPF, 2012; WEISS et al., 2014) implements the attack in virtualized embedded environments. Spreitzer and Plos (SPREITZER; PLOS, 2013) and Spreitzer and Gérard (SPREITZER; GéRARD, 2014) perform the attack on mobile phone devices. Apecechea et al. (APECECHEA et al.,) and Irazoqui et al. (IRAZOQUI et al., 2014) demonstrate the attack in virtualized environments used in commercial cloud computing systems.

3.1.1.1 Bernstein's Attack

This attack exploits timing variations of the AES-128 performance-oriented implementation (T table implementation described in subchapter 2.3). The first round of AES follows the equation described in 2.2, The index of the Table is calculated from the exclusive-or (xor) between a byte from the plaintext and a byte from the key. For instance, $T_0[k[0] \oplus n[0]]$, where $k[0]$ is the first byte from the key and $n[0]$ is the first one from the plaintext. Since these look-up tables are stored in the main memory, each different index accessed provoke a cache miss. If some index value is repeated, it occurs a cache hit. The occurrence of cache misses increases considerably the delay of AES operation. Therefore, all AES computation is well correlated with the time of the accesses to the cache.

Each cache access has a strong relation between the plaintexts and key bytes. Then, knowing the plaintext used, and analyzing the time behavior for each byte position, it is possible to identify patterns between encryptions with a known and unknown keys. Bernstein uses four phases to accomplish that objective: i) Profile; ii) Attack; iii) Correlation; iv) Exhaustive key search.

Profile Phase The first phase aims to extract the statistical behavior of the encryption time. This step can be made in an online or an offline manner. The profile is considered online when the extraction can be done on the same platform of the victim. The profile is offline, when it is made in different platform, but with similar behavior.

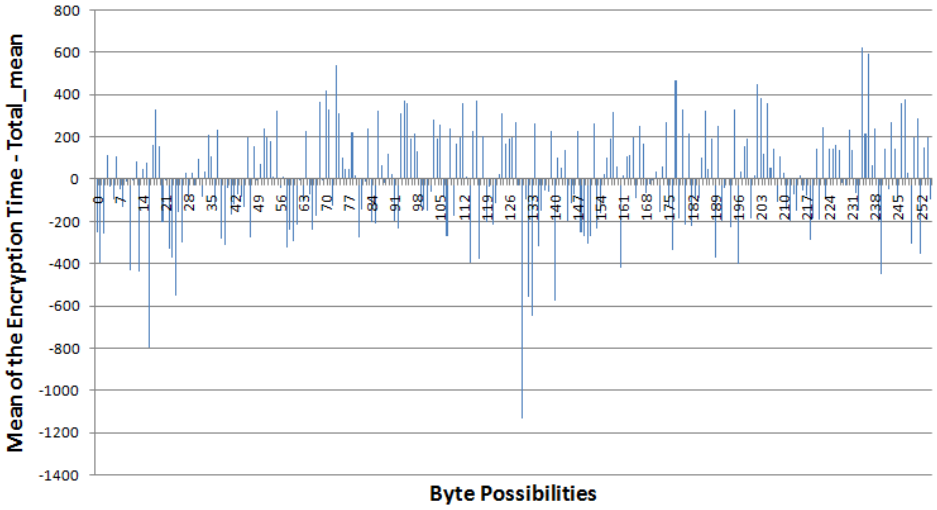
The attacker has to generate statistical signatures for each byte position of the plaintext (16-byte positions in total). Each target position is encrypted with a known key several times. The process takes into account all possibilities for that position (from 0 to 255). The timing to perform the encryption of each value is accumulated. Then, dividing by the number of occurrences, one calculates the average time for each possibility. For instance, ten thousand encryptions were performed to evaluate the first position of the plaintext. For each value, the execution time was accumulated to calculate its average and

stored in a table. At equation 3.1, the accumulation of encryption times are represented by the sums of t, while the occurrences are given by letter n. After collecting all, the results are normalized by the mean time of all encryptions (independently of the byte), resulting in the vector v at equation 3.1 presented below. The table represented by the letter v stores these final results, where b accounts for each byte possibility (0 to 255), and i the byte position of the plaintext (0 to 15).

$$v[i][b] = \frac{t[i][b]}{n[i][b]} - \frac{\sum_i \sum_b t[i][b]}{\sum_i \sum_b n[i][b]} \tag{3.1}$$

Then, the values in table v are plotted in a graph, which is defined as the signature for the respective position of the plaintext, in this example position zero. The same process is done for all byte positions of the plaintext. The signature of position zero is showed in figure 7.2.

Figure 3.1: Profiling Phase - Signature of the position 0.



Attack Phase The attack phase is performed on the victim, where random plaintexts are sent for encryption. In the same manner, as in profile phase, all times are computed to obtain the normalized average time for each possibility of each byte position. As a result, a new signature table is generated, known as the table $v^{'}$, since the key now is unknown.

Correlation Phase In this phase, both signature tables (v and $v^{'}$) are correlated. The objective of the correlation is to reveal some patterns between the plaintext and the key.

The correlation follows the equation 3.2 below:

$$c[i][j] = \sum_{j=0}^{255} v[i][j].v^{'}[i][j \oplus b] \tag{3.2}$$

The character b is the guessed key byte, which has to be tested along the computation of several correlations. The result of the correlation phase, $c[i][j]$, is sorted, and the higher values are used to send the key candidates to the last phase of the attack.

Exhaustive key search: Since the correlation phase usually results in multiple key candidates per key byte, an exhaustive key search on the remaining keyspace has to be performed.

3.1.1.2 Neve's Optimization

This optimization enabled full key recovery on Bernstein technique, something the original attack could not guarantee. The work achieved this result by investigating the timing attack, concluding that performing the attack locally (inside the hardware platform) the patterns in cache eviction could reveal all bytes of the key. To do so, Neve et al. proposed two strategies:

1. To select the plaintext to be encrypted in attacking phase: By choosing the plaintext, it is possible to force test conditions, where the correlation with the signatures can be amplified;
2. To extend the analysis to the second round of AES: It is possible to use the behavior of the second round to improve correlation results. It is possible to guess knowing the plaintext.

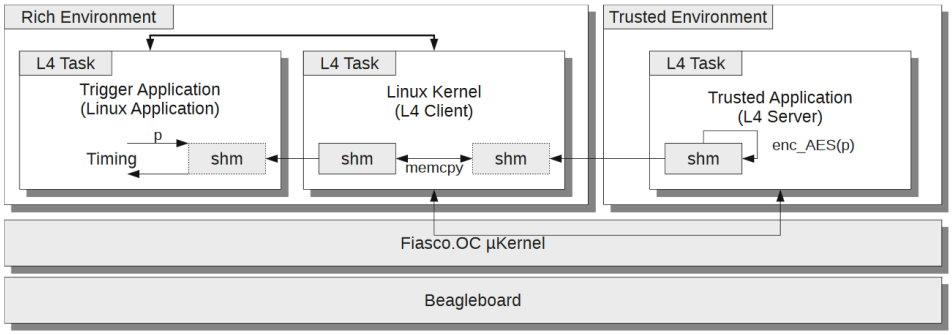
3.1.1.3 Application: Virtualized Embedded Environments

Weiß et al. (WEISS; HEINZ; STUMPF, 2012) implemented the attack of Bernstein in virtualized embedded environments targeting applications of stock market and financial transactions. The work showed that the timing attack is possible even when the victim runs in a trusted environment (figure 3.2).

Weiß et al. added to the profiling phase one more step. The first one executes the encryption outside the target system, defined as offline profiling. Then, the second step runs on the target system but without knowing the key. The objective is to discover

the time to encrypt in the trusted environment. The proposed architecture, observed in figure 3.2, executes the malicious software in the rich operating system (OS), while the encryption runs in the trusted OS. The application has to make a system call for the rich OS (client) request the encryption for the trusted OS (server). Then, with the signatures generated offline, and the time behavior extracted in the target system, the authors were able to perform the attacking phase.

Figure 3.2: Rich operating system making an system call to the trusted operating system. *Source:* (WEISS; HEINZ; STUMPF, 2012).



Later, in (WEISS et al., 2014) Weiß et al. implemented the same strategy in virtualized environments that used multi-cores as the hardware platform. Besides, the work focused on a real-time kernel, called PikeOS, typically used in avionics and automotive equipment. This environment is challenging, because the operating system isolates the trusted services altogether, what is called partitions (figure 3.3). Moreover, the sophisticated scheduler of PikeOS to guarantee real-time conditions inserts timing noises in the process. So, experiments tested the efficiency of the attack under different scheduler configurations.

By comparing the results of the attack against single- and multi-core systems, the paper concludes that to dedicate a core to perform the crypto task has a huge impact on the vulnerability of such systems. The exclusive core reduces computational noises, improving the correlation of the attacking phase.

3.1.1.4 Application: Mobile phone devices

Spreitzer and Plos (SPREITZER; PLOS, 2013) investigated the Bernstein's attack in a real environment on three mobile devices: Acer Iconia, Galaxy SIII, and Google Nexus S. Table 3.1 shows the number of measurements to implement the attack, for both phases, profile, and attack.

Results revealed that a high number of measurements are required for the attack.

Figure 3.3: Communication between partitions inside PikeOS. *Source:* (WEISS et al., 2014).

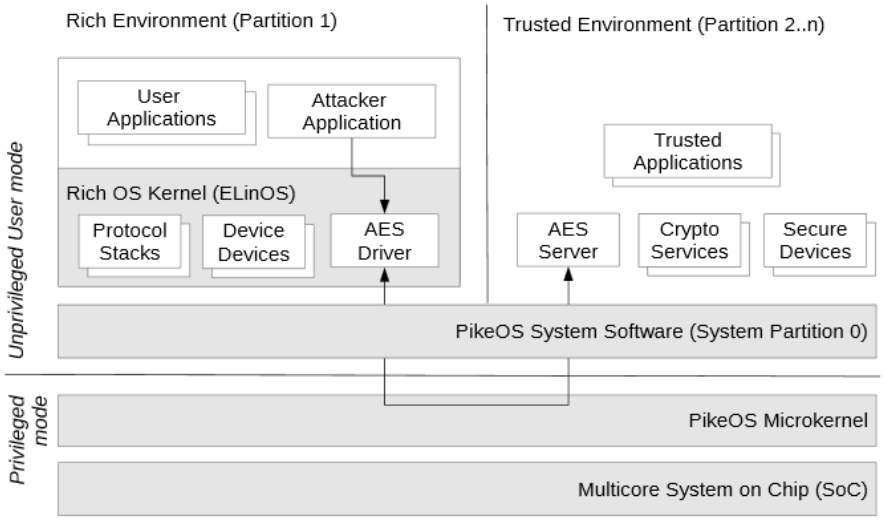


Table 3.1: Number of measurements to implement the attack on different mobile phone devices.

Device	Samples		Remaining Key Space
	Profile Phase	Attack Phase	
Acer Iconia A510	2^{30}	2^{27}	73 bits
	2^{30}	2^{29}	78 bits
Google Nexus S	2^{30}	2^{29}	65 bits
	2^{29}	2^{28}	69 bits
Samsung Galaxy SIII	2^{30}	2^{29}	58 bits
	2^{30}	2^{30}	61 bits

To compute the 2^{30} encryptions, the Google Nexus S takes about 6 hours. It could drain the battery drastically, creating difficulties to perform the attack to recover all bytes of the key. However, it is very feasible to reveal part of the key, and the Bernstein timing attack can be considered a relevant thread for real mobile devices.

Another work that exploited mobile environments was presented by Spreitzer and Gérard (SPREITZER; GéRARD, 2014). Their work investigated the recovery of different parts of the key, exploring the granularity. Besides, it enhanced Bernstein's attack through a technique proposed by Aly and ElGayyar (ALY; ELGAYYAR, 2013). The method adds new timing information in the correlation phase. This novel information consists of the overall minimum encryption time (global minimum) and the minimum encryption time for a particular plaintext byte at a specific position (local minimum). Then, the time to perform the exhaustive key search can be drastically reduced.

3.1.1.5 Application: Virtualized Cloud Environments

Apecechea et al. (APECECHEA et al.,) and Irazoqui et al. (IRAZOQUI et al., 2014) applied Bernstein attack in commercial cloud computing systems, evaluating the security of virtualization tools and crypto-libraries implementations.

Firstly, Apecechea et al. and Irazoqui et al. presented some challenges and opportunities in virtual machine architectures. They cite five commercial features for memory management important for the attack:

- Address Space Layout Randomization (ASLR): At each execution of a process, the memory location is changed to avoid byte overflow attacks. This can affect the behavior of the execution time of the processes.
- Second Level Address Translation (SLAT): The translation of virtual machine addresses to physical ones can be done directly in hardware through a particular TLB (translation look-aside buffer). This improves performance, but creates a vulnerability, since VM addresses will be linked directly to physical ones.
- Kernel Samepage Merging (KSM): The related information can be merged in memory, so different VMs access the same addresses. Hence, it opens a leakage, where a VM can observe the behavior of other VM.
- Transparent Page Sharing (TPS): The same page can be shared for different VMs. This page sharing can reveal the link of logical addresses of one VM to another.

Then, the work mentioned that most common crypto libraries have different implementations of the T table AES algorithm, and its security should be studied. The experiments targeted the Amazon EC2 and the Rackspace cloud services. Both use the XEN VMM to provide virtualization. And, since companies are typically deploying VMware as another virtualization technology to reduce IT costs, it was included in the evaluations as well. Results showed that:

- The latest versions of the libraries were secure against last round attacks of AES.
- The first round of all libraries were vulnerable to cache timing attack.
- There is a considerable increase of noise when moving from native machine to virtual machine execution.
- There is no difference in the attack results from a single VM to a cross-VM scenario.
- Processes running together inserts noise to the attack.
- It was possible to recover a minimum of 30 bits for the cross-VM scenarios.

3.1.2 Access-based Cache Attacks

Access-based cache attacks were first introduced by Osvik et al. in a paper that propose three new attack strategies (OSVIK; SHAMIR; TROMER, 2006). Osvik et al. proposed that it is possible to observe the used sets of the cache by accessing the same positions after the encryption. The technique was called Prime+Probe, and it has been widely studied by the security community (XINJIE et al., 2008) (LIU et al., 2015) (CRANE et al., 2015). Prime+Probe preconditions are practical if the attack can be implemented in the victim platform.

3.1.2.1 Prime+Probe Attack

The Prime+Probe is an access-based cache attack proposed by Osvik (OSVIK; SHAMIR; TROMER, 2006). The attacker analyses the access of the cache after each encryption, to detect which sets of the access were used. The positions accessed are directly related to the T table indexes. Hence, knowing the plaintext used, the attacker can evaluate what the possibilities of the key are.

Osvik et al. implemented Prime+Probe at systems composed of only a single processor. The attacker employed a spy process to run on the same core as the target cipher algorithm (victim process). Therefore, the attacker could have access to the same resources of the victim process, such the cache L1. Therefore, some preconditions need to be met to accomplish the Prime+Probe attack successfully:

- Attacker knows the cache configuration;
- Attacker knows the location of the AES lookup tables in memory, to know the cache positions;
- Attacker generates the encryption plaintext; and
- Attacker can access the cache.

Prime+Probe is performed in five stages. At the first stage, the attacker prepares the cache (Prime). The second stage is the encryption of a random known plaintext. The third stage is used by the attacker to read the cache and to extract information of accesses (Probe). The fourth step is the analysis of the collected information. As a result, the attacker reduces the key search space drastically. Finally, the fifth stage is an exhaustive search key step, where all the remaining possibilities are tested. Each one of the five stages is described below:

Prime The attacker must write in the cache a malicious vector, whose size have to be at least the size of the target T table of AES. Besides, the attacker needs to know the address that the destination T table will be located in the cache.

Encryption At this step, the attacker requests an encryption with a random known plaintext. The encryption process will execute and access the cache accordingly to T table index. According to AES performance oriented algorithm, the indexes are given by the exclusive-or between the bytes of the plaintext and the key, such as $index \leftarrow plaintext[0] \oplus key[0]$ for the first access in T_0.

Probe After AES encryption occurs the Probe stage, where the attacker retrieves the malicious vector. Since the AES will use some memory positions during execution, when the attacker retrieves the malicious vector, cache misses taking place. The misses can be identified by the higher response time during the attackers' vector reading. So, when a cache miss is detected, the attacker translates the vector index (address) to the respective cache set. The information of the used sets for each encryption and the plaintext, generated as well by the attacker, are stored for the analysis process.

Analysis The values generated after the first round follow the expression $x_i^0 = p_i \oplus k_i (i = 0, ..., 15)$. Therefore, by testing the data acquired in the Probe stage, it is possible to identify the sets that the crypto-processor has not used. That is the non-accessed indexes. By assuming that $x_i^0 \neq p_i \oplus k_i (i = 0, ..., 15)$ and knowing the plaintext byte p_i, is possible to prove that $k_i \neq p_i \oplus x_i^0 (i = 0, ..., 15)$. Hence, possible key candidates can be removed. This strategy reduces the key search space within the brute force process.

Exhaustive key search In the final stage, remaining key bits have to be verified through any brute-force search space algorithm.

3.1.2.2 Xinjie et al. Optimization

Xinjie et al. (XINJIE et al., 2008) offer a novel analysis strategy to implement Prime+Probe. Instead of using the accessed lines of the cache, it targets the non-accessed lines. The objective is to reduce the number of traces required to reveal the key.

The analysis stage employed in this work is based on the algorithm presented in (XINJIE et al., 2008). We perform the first round analysis, where only the accesses during

the first AES round are used for the analytical test. The values generated after the first round follow the expression $x_i^0 = p_i \oplus k_i (i = 0, ..., 15)$. Therefore, by testing the data acquired in the Probe stage, it is possible to identify the sets that the crypto-processor has not used. That is the non-accessed indexes. By assuming that $x_i^0 \neq p_i \oplus k_i (i = 0, ..., 15)$ and knowing the plaintext byte p_i, is possible to prove that $k_i \neq p_i \oplus x_i^0 (i = 0, ..., 15)$. Hence, possible key candidates can be removed. This strategy reduces the key search space within the brute force process.

3.1.2.3 Application: Last Level Caches

Liu et al. (LIU et al., 2015) showed that the Prime+Probe technique could attack even the last level cache (LLC). This work presented a way to get the accesses of the cache properly, even when the attacker only shares the higher hierarchy of the cache, the LLC. It developed two techniques to break an ElGamal cipher decryption: i) Probe cache sets without knowledge of the virtual address mapping, and ii) identify victims security-critical accesses using temporal access patterns.

3.1.3 Collision-based Cache Attacks

Collision-based cache attacks are a variation of the Bernstein timing attack. The objective of this attack is to explore the same accesses to the same T table. For example, in the first round of AES, the elements x_0, x_4, x_8, x_12 access the same table T_0. If more than one is equal, it represents that the second one will force a cache hit, and the overall time to perform the AES will decrease. Since, each x_i is given by the calculation of one byte of the plaintext and one of the key, such as $p_i \oplus k_i$, knowing the plaintext it is possible to identify key candidates.

The first approach of this technique was proposed by Bonneau and Mironov (BONNEAU; MIRONOV, 2006), where they presented how to perform this attack in the first and last round of AES. Later, Bogdanov et al. (BOGDANOV et al., 2010) modified the Bonneau and Mironov attack to a differential one. Bogdanov explored the collision between pairs of plaintext, generated to provoke at least five collisions. Spreitzer and Plos (SPREITZER; PLOS, 2013) did experiments with Bogdanov technique and evaluated the applicability of the attack on mobile devices.

3.1.3.1 Bonneau and Mironov Attack

Bonneau and Mironov presented for the first time the cache collision attack, and they describe three approaches: i) first round, ii) last round, and iii) expanded last round.

The first round collision attack explores the same access that may happen to the same T table. So, the attack aims to find all combination of plaintext and key that result in the same index of the target T table. This condition can be represented as $p_i \oplus k_i = p_j \oplus k_j$, where i and j represent the byte positions that access the same table. Since the attacker knows the plaintext used, he only needs the information of the impact on AES encryption time caused by each possible collision. To do so, Bonneau and Mironov worked on the relation that if there is a collision then $p_i \oplus p_j = k_i \oplus k_j$. Several encryptions for each possible pair p_i and p_j were performed, where the low average time was defined as a delta time related to the key expression $\Delta = k_i \oplus k_j$. Through these calculated times, it is possible to identify the access to the same set of the memory, due to the collision identification. The drawback is that it was not possible to guess exactly which address was accessed, only the set. As a consequence, the attacker was capable of retrieving only 68 bits from the key. Therefore, this first round approach was considered impractical by the authors.

To overcome such drawback, Bonneau and Mironov proposed an attack to the last round of AES. Typically, the last round requires the usage of an extra T table, known as table T_4, because last round does not compute the MixColumn operation. Hence, the T_4 accesses are exclusive for the last round and occurs as follows:

$$\begin{aligned}
(c_0, c_1, c_2, c_3) &\leftarrow (T_4[x_0^{10}] \oplus k_0^{10}, T_4[x_5^{10}] \oplus k_1^{10}, T_4[x_{10}^{10}] \oplus k_2^{10}, T_4[x_{15}^{10}] \oplus k_3^{10} \\
(c_4, c_5, c_6, c_7) &\leftarrow (T_4[x_4^{10}] \oplus k_4^{10}, T_4[x_9^{10}] \oplus k_5^{10}, T_2[x_{14}^{10}] \oplus k_6^{10}, T_3[x_3^{10}] \oplus k_7^{10} \\
(c_8, c_9, c_{10}, c_{11}) &\leftarrow (T_4[x_8^{10}] \oplus k_8^{10}, T_4[x_{13}^{10}] \oplus k_9^{10}, T_2[x_2^{10}] \oplus k_{10}^{10}, T_3[x_7^{10}] \oplus k_{11}^{10} \\
(c_{12}, c_{13}, c_{14}, c_{15}) &\leftarrow (T_4[x_{12}^{10}] \oplus k_1 2^{10}, T_4[x_1^{10}] \oplus k_{13}^{10}, T_2[x_6^{10}] \oplus k_{14}^{10}, T_3[x_{11}^{10}] \oplus k_{15}^{10}
\end{aligned} \tag{3.3}$$

As observed in equation 3.3, the result of the operations are stored in c_i, which is the ciphertext. For any two ciphertext bytes c_i and c_j, it holds that $c = k_i^{10} \oplus T_4[x_u^{10}]$ for some u and $c = k_j^{10} \oplus T_4[x_w^{10}]$ for some w. Regardless of the actual values of u and w, whenever $x_u^{10} = x_w^{10}$, a cache collision occurs on T_4. Suppose $x_u^{10} = x_w^{10}$ and $T_4[x_u^{10}] = T_4[x_w^{10}] = \alpha$. Then it will hold that $c_i = k_i^{10} \oplus \alpha$ and $c_j = k_j^{10} \oplus \alpha$. Hence, if there is a collision in the last round, one can assume that $c_i \oplus c_j = k_i^{10} \oplus k_j^{10}$. So, the attacker

performs several encryptions, annotating the low average time of the encryption for each combination of ciphertext bytes that could generate a collision. This timing information is represented as $\Delta = c_i \oplus c_j$ and each possible combination is stored in a table, such as $T[i, j, \Delta]$. The goal to find one value $\Delta'_{i,j}$ for each i,j such that $T[i, j, \Delta'_{i,j}] < t'$, where t is the average encryption time over all ciphertexts. Eventually, the values of $\Delta_{i,j}$ will become accurate guesses for the true values $\Delta = k_i^{10} \oplus k_j^{10}$, which should be the only values which cause significantly low encryption times.

The enhanced last round attack uses not only the access of the same indexes but all index that provoke a cache hit (collision). This feature depends on the cache line size. With this information, the attack can work with more possibilities of Δ.

3.1.3.2 Bogdanov's Attack

This attack is known as differential collision cache attack. This name refers to the exploration of collisions between pairs of plaintexts. These collisions affect the encryption time of the second plaintext, which becomes lower. It is challenging to detect the variations in time caused by the collisions, so Bogdanov explored a particular condition. He called it as the wide-collision. This situation has the potential to provoke five S-Boxes collisions in the first three rounds of AES algorithm.

To create the wide-collision situation, the pair of plaintexts (P_1, P_2) have to follow a specific formation rule. Firstly, the attacker has to define a target diagonal. Then, he creates randomly both plaintexts, chosen pairwisely equal the elements out the target diagonal, and pairwisely different the ones inside the target diagonal. The example below shows the main diagonal as the target, where the elements from P_1 are represented as a_i and from P_2 as e_i, where $0 < i < 4$:

$$P_1 = \begin{bmatrix} a_0 & x_1 & x_2 & x_3 \\ x_4 & a_5 & x_6 & x_7 \\ x_8 & x_9 & a_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & a_{15} \end{bmatrix} \quad P_2 = \begin{bmatrix} e_0 & x_1 & x_2 & x_3 \\ x_4 & e_5 & x_6 & x_7 \\ x_8 & x_9 & e_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & e_{15} \end{bmatrix}$$

The encryption process of this pair (P_1, P_2) can provoke a wide-collision if at least one position of the S-Box collide for both plaintexts in the second round of AES. The position of the second round S-Box is a result of the first round computation (Addround, SubBytes, ShiftRow and MixColumn). For example, to compute the element a_0 that

results in S_0, it is used the following equation 3.4:

$$S_0 \leftarrow 02.Sbox(a_0 \oplus k_0) \oplus 03.Sbox(a_5 \oplus k_5) \oplus 01.Sbox(a_{10} \oplus k_{10}) \oplus 01.Sbox(a_{15} \oplus k_{15}) \tag{3.4}$$

After computing the first round for all elements (S_i), one obtain the new pair of plaintext (P_1^2, P_2^2) used to perform the second round, as follows:

$$P_1^2 = \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \quad P_2^2 = \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}$$

Note that the elements in grey represents the pairwisely different values. The non-marked elements are pairwisely equal, but this is inherent to the wide-collision situation. This example considers a collision in S_0, identified by the elements in green (also pairwisely equal). Since all elements in the main diagonal are pairwisely equal, applying again the equation 3.4 in the second round pairs (P_1^2, P_2^2), one obtains four more collisions in the third round. The new pair of plaintexts (P_1^3, P_2^3) will collide for all elements in the first column. The plaintexts pair of the third round is represented as follows:

$$P_1^3 = \begin{bmatrix} t_0 & t_4 & t_8 & t_{12} \\ t_1 & t_5 & t_9 & t_{13} \\ t_2 & t_6 & t_{10} & t_{14} \\ t_3 & t_7 & t_{11} & t_{15} \end{bmatrix} \quad P_2^3 = \begin{bmatrix} t_0 & t_4 & t_8 & t_{12} \\ t_1 & t_5 & t_9 & t_{13} \\ t_2 & t_6 & t_{10} & t_{14} \\ t_3 & t_7 & t_{11} & t_{15} \end{bmatrix}$$

Three main stages comprise the attack:

Online Stage: For a pair of chosen 16-byte plaintexts (P_1, P_2), the main goal of the *online stage* is to measure the encryption time of P_2. To produce a wide-collision situation, the pair of plaintext has to follow the rule described above. Since this rule only applies for one diagonal per time, this process repeats four times, one for each target diagonal. The total amount of encryptions required for each diagonal is $N * I * r$. The variable N represents the number of random values the same diagonal will explore. The variable I accounts for each iteration that the pairwisely equal values are changed to perform an

in-depth exploration of all possibilities inside the same diagonal. The variable r represents how many times one performs the same encryption. The objective is to reduce noise interferences. The time of all encryptions is stored and sent to the detection stage.

Detection Stage: The detection is made by analyzing all encryption times t. It is expected that t will be lower than average (the result of five look-up tables already in the cache memory) if a collision occurs. The result of this stage is a list of the pairs of plaintexts (P_1, P_2) that possibly caused a wide collision. They are also classified as candidates.

Key Recovery Stage: This stage is divided into two steps. The first step supposes we found 4+m candidates on the detection stage, $m \in \{0, 1, 2, ..\}$. We consider all $\binom{4+m}{4}$ possibilities with all 2^{32} subkey candidates. For each choice of four pairs of (A_i, E_i) we execute AddRoundKey, SubBytes, ShiftRows and MixColumns. If a collision occurs for at least one position in the second round; for each of the four pairs in the group, the subkey candidate joins the final candidates' list. The total complexity of this step is in the order of $2^{32} * \binom{4+m}{4}$.

The second phase of this stage completes the full key recovery. It concatenates the subkey final candidates and performs an exhaustive key search. The total complexity of this searching algorithm is $(2^{32} * \binom{4+m}{4})^4$.

3.1.3.3 Application: Mobile phone devices

Spritzer and Plos investigated the applicability of Bogdanov's attack in mobile phone devices. The evaluation used the same strategy of the original technique, exploiting the wide-collision behavior. However, the work showed that the wide-collision detection in mobile hardware platform was a big challenge. The main reason was the size of the cache line, which was 64 bytes for the tested devices. This line size made the attack unpractical because few cache misses occurred. As a consequence, the chance of success in the detection stage of wide collisions was about 10%. Besides, Spreitzer and Plos analyzed the key recovery phase and concluded that this lack of precision in the detection would increase the false-positives significantly. Therefore, to check all the possibilities would be in the order of 2^{52}, which is not feasible.

3.2 Networks-on-Chip Attacks

An ACA on NoC requires the control of at least one component in the MPSoC. There are techniques that an attacker can use to tamper the software and infect an IP (FIORIN; PALERMO; SILVANO, 2008). By using malicious software (malware) that performs read and write transactions in forbidden memory areas, an attacker may change the behavior of an IP (victim IP) and turn it into an infected IP. Moreover, buffer overflows, and other similar techniques that address software weaknesses can be exploited for such a purpose (FIORIN; PALERMO; SILVANO, 2008). An infected IP may try to extract/infer data, modify the system behavior (by infecting other IPs) or deny the MPSoC service employing malicious transactions.

The work of (MANCILLAS et al., 2014) shows that long packets characterize the sensitive information. Hence, during a burst communication of a sensitive packet in the NoC, a significant part of the bandwidth is consumed. Several authors (YAO; SUH, 2012) (WASSEL et al., 2014) (SEPULVEDA et al., 2016) have cited the possibility to use the throughput degradation caused by sensitive packets as a leakage source. In this case, an attacker observes the time to inject packets in the NoC and verifies when there is an increase in that time. The delay is a result of a long packet consuming the bandwidth of the NoC, a possible sensitive packet. This attack is known as NoC timing attack.

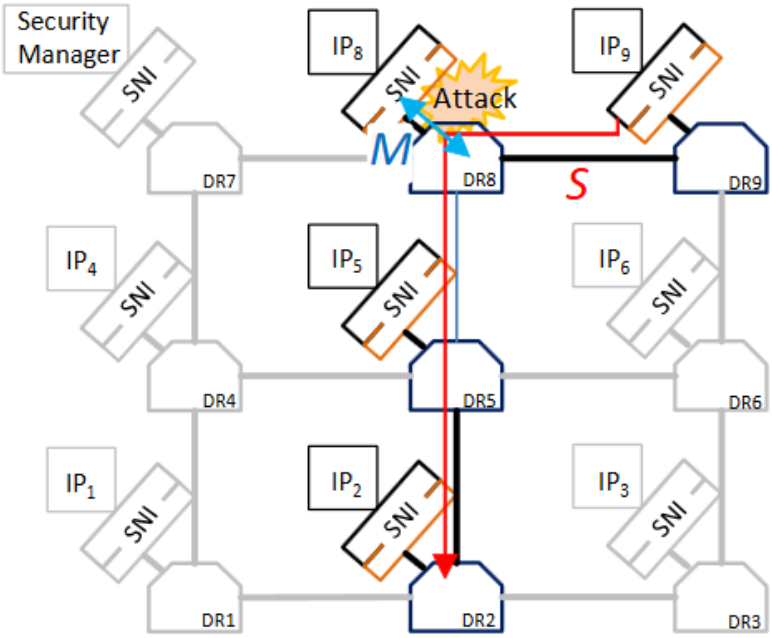
3.2.1 Timing-based NoC Attacks

NoC timing attacks elicit channel leakage by the evaluation of the attacker throughput. As routers are shared, the communication collisions between malicious and sensitive traffic may reveal the sensitive behavior (e.g. mapping, topology, routing, transmission pattern and volume of communication). The collisions are detected by the reduction of throughput of the attacker. Several authors explained the concept behind NoC timing attack (YAO; SUH, 2012) (WASSEL et al., 2014) (SEPULVEDA et al., 2016) (SEPULVEDA et al., 2015) and (STEFAN; GOOSSENS, 2011). However, no work has presented a detailed attack with its threat model, such as preconditions and configurations of the attacker.

According to the authors of (YAO; SUH, 2012) (SEPULVEDA et al., 2016), by injecting frequent transactions, the infected IP saturates a router output port. Due to that output port is shared by the malicious and the sensitive data, the throughput degradation of

the attacker can be used to infer the access pattern of the victim flow (e.g., communication flow generated by a crypto processor to memory). This behavior can be observed in figure 3.4, where the victim traffic is represented by the S letter and the attacker by letter A.

Figure 3.4: Malicious software M performing the NoC timing attack. The sensitive traffic S is the victim. *Source:*(SEPULVEDA et al., 2016)



Wang and Suh (YAO; SUH, 2012) studied this network interference showing the throughput observed by an attacker. Besides, they proposed a case, where an attacker could explore such leakage source. Their example used as the victim an RSA cipher IP. To perform the encryption, this cipher required to access a multiplication module. The RSA algorithm multiplies two large numbers (often 1024 or 2048 bits) that depend on each bit in a secret key (RIVEST; SHAMIR; ADLEMAN, 1978). When the bit of the key is one, then the multiplication occurs. Therefore, Wang and Suh explored the timing leakage of the NoC to understand the access of the IP and guess the secret key.

3.3 Security for Caches

Below it is presented the software-based countermeasures found in literature. Then, a summary comparing the main strategies is showed later through the work of Alawatugoda et al. (ALAWATUGODA; JAYASINGHE; RAGEL, 2011).

Regarding solutions implemented directly in the crypto-library, different strategies have been proposed. In (REBEIRO; MONDAL; MUKHOPADHYAY, 2010), the author

changed the typical T table from performance oriented AES to the S-box generation. It increases significantly the encryption time, since the values have to be calculated at run-time.

Other works proposed random permutations during AES encryptions to mask the actual cache access patterns (BRICKELL et al., 2006) (BLOMER; KRUMMEL, 2007). Hence, the performance of the encryption was highly degraded as well.

Stefan et al. (STEFAN et al., 2013) presented an instruction scheduler to be implemented in compilers. The objective was to rearrange the instructions in the threads to avoid timing leakages.

Crane et al. (CRANE et al., 2015) proposes a dynamic software diversification. The application diversification (modification of the code and data) can be applied in the compiler or in the software to modifies the behavior of an algorithm. However, Crane explains that an ACA can adjust itself at run-time and understand the new pattern. Therefore, they propose a dynamic diversification, where the software changes its flow during execution. Several control flows are inserted in the code, and random conditions set the behavior during run-time.

3.3.1 Countermeasures Comparison

Alawatugoda et al. presented a comparison between four countermeasures. They were implemented in software by the modification of the crypto library (ALAWATUGODA; JAYASINGHE; RAGEL, 2011). The techniques presented and analysed were: i) random loops; ii) specified loops; iii) pre-fetch of T tables; and iv) cache partitioning.

The random loop insert *for* statements inside AES code, using random numbers as the iteration limit. Hence, a random delay is inserted between T table accesses, creating a timing noise.

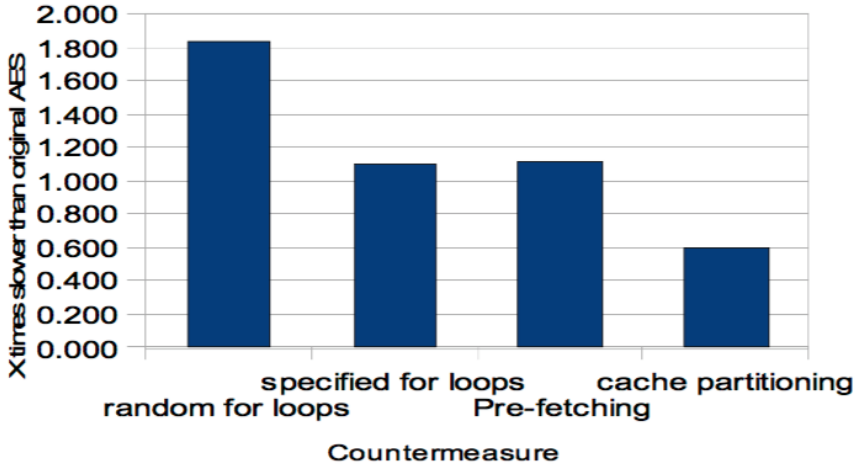
Another technique is the specified loops, which changes the random limit by a specific sequence. Using a known limit it is possible to avoid big numbers. High latencies degrade the encryption performance, then it is important to avoid unnecessary delays.

The third approach is the pre-fetch of T tables. Only a part of the T tables is read before each encryption. Hence, this will affect the behavior of cache misses and hits.

The final countermeasure is the cache partitioning. In this case, one isolates a memory space to execute only the AES. Hence, this may change the original pattern of the encryption.

These techniques use software implementation to protect the cache against timing attacks. The loop strategy from the first two degrade more the encryptiong performance. The last one, can avoid any performance penalty, since the memory isolation only benefit the execution of AES. However, a reserved memory space for cryptography can limit other features from the system. Results on figure 3.5 shows the latency overhead of each technique.

Figure 3.5: Latency overhead of software-based countermeasures against cache timing attacks. *Source:*(ALAWATUGODA; JAYASINGHE; RAGEL, 2011)



3.4 Security for NoCs

NoC design challenges regards to provide defence services and the required performance. Recent works presented protection mechanisms for NoCs, like authentication (SEPULVEDA et al., 2009), access control and integrity (SEPULVEDA et al., 2012). They used firewalls and error correction algorithms as a solution.

The works in (YAO; SUH, 2012) (WASSEL et al., 2014) (SEPULVEDA et al., 2016) (SEPULVEDA et al., 2015) and (STEFAN; GOOSSENS, 2011) address the protection against NoC timing attack. The works of Wang and Suh (YAO; SUH, 2012) and Wassel et al. (WASSEL et al., 2014) propose the integration of hard QoS (Quality-of-Service) mechanism to isolate the sensitive information. They include temporal network partitioning, based on high (YAO; SUH, 2012) and bounded (WASSEL et al., 2014) priorities arbitration schemes. Sepulveda et al. (SEPULVEDA et al., 2015) proposes random arbitration and adaptive routing as protection techniques. In another work, Sepulveda et al. (SEPULVEDA et al., 2016) present the secure enhanced router (SER) architecture that

dynamically configures the router memory space according to the communication and security properties of the traffic. Furthermore, the work of Stefan and Goossens (STEFAN; GOOSSENS, 2011) present the usage of multiple path communication for sensitive flows.

3.4.1 Wang and Suh - Priority Arbitration NoC

Wang and Suh proposed a NoC to deal with the multi-level security model (MLS) (YAO; SUH, 2012). The MLS can divide the traffic in several security domains. However, this work study only the application of two levels - high and low. Their MLS only has the restriction that the communication from high to low can not happen. To avoid a NoC timing attack in the high security flow, the work mentions that the typical approach is to employ static partitions in the network:

- Spatial Network Partitioning (SNP): Some routers are reserved only for high security flows.
- Temporal Network Partitioning (TNP): Routers with virtual channels are used, where the time-division multiplexing (TDM) is used. Then, a portion of the time slices is reserved only for high security flows.

Although both spatial and temporal partitioning schemes eliminate network interference between the security domains, they bring critical performance overhead. The main reason is that the resource allocation cannot be dynamically adjusted to match the actual demands from each security domain.

Therefore, Wang and Suh presented a novel NoC that uses priority arbitration for the crossbar that implements the TDM. The objective is to assign a high priority for the low security packets. Hence, the high security ones will not affect the traffic of the low security ones. But, such strategy creates a significant vulnerability for denial-of-service attacks. Low security traffic always are prioritized, and could inject data uninterruptedly. To avoid this issue, the authors implemented a traffic limitation. When the low security traffic creates long bursts and achieves a threshold of the bandwidth, the router stops that traffic for an awaiting time. This mechanism is also static, defined in design time. Results shows that these strategies allow to remove timing leakages from the sensitive traffic behavior being an effective secure-enhanced NoC.

3.4.2 Wassel et al. - Surf NoC

Wassel et al. brings the security concerns for critical applications, such as medical and aerospace (WASSEL et al., 2014). Considering MPSoC scenario, the isolation of the cores is the primary approach to enable security. Typically, this can be achieved through time or spatial partitioning, creating several domains. However, this strategy always penalize the performance. To overcome such drawback, they propose the Surf NoC. This NoC uses temporal partitioning. Each time slot of the temporal multiplexing is considered as a $wave$, due to the periodic behavior. Then, each domain's packets awaits for its $wave$ to proceed. To improve performance, the Surf router uses a specific scheduler that allows that different domain's packets forward in the same cycle. Since the timing slots are synchronized between routers, the packets after beginning the surfing do not wait again. It is important to mention that this approach is still a static partitioning. According to Wassel, this is important, because dynamic partitioning could provide new side channels.

Results showed that Surf NoC has good results when the MPSoC employs a great number of domains. The number of domains is related to how many levels of security the system will provide. The authors expect that in the same manner that occur in business networks, hundreds of levels will be implemented in such embedded systems.

3.4.3 Sepúlveda et al. - Random Arbitration and Adaptive Routing NoC

Two strategies are implemented in the NoC proposed by Sepúlveda et al. - random arbitration and adaptive routing (SEPULVEDA et al., 2015).

The random arbitration regards the router crossbar decision of which input port will be attended. If a typical round-robin is implemented, the deterministic characteristic can reveal patterns in the scheduling becoming a source of leakage. By implementing a random number generator inside the arbiter logic, the behavior of the scheduling becomes non deterministic.

However, another possible timing leakage in the NoC is the packet collisions that can be provoked by an attacker. It is possible to force interference in a sensitive traffic, because the deterministic routing algoritm allows the malicious element to infer the path of the packets. Then, it is possible to create a traffic that intersects the victim. To avoid this timing leakage, this work uses an adaptive routing algorithm. The used algorithm is the West First Routing Logic (WFRL). In this algorithm, each packet can take three

possible output ports (East, North and South). Then, the sensitive traffic pattern is splited in different paths, which poses a significant challenge regarding the NoC timing attack.

3.4.4 Sepúlveda et al. - SER

Sepúlveda et al. present the Secure Enhanced Router (SER) for NoC architectures (SEPULVEDA et al., 2016). SER protect the MPSoC against NoC timing attacks using dynamic virtual channel allocation. By reconfiguring the buffers allocated for each input port, the traffic throughput can be adjusted in run-time according the applications requirements and security concerns. As a result, it is possible to avoid the timing leakage from the collision of the packets, and avoid any performance penalty.

The SER was evaluated under different traffic scenarios. Besides, this work compared SER with the NoCs proposed by Wang and Suh (YAO; SUH, 2012) and Sepúlveda et al. (SEPULVEDA et al., 2015). The three NoC architectures were able to protect the NoC against timing attack, but SER obtained the better performance.

3.4.5 Stefan and Goossens NoC

Stefan and Goossens improves the time-divison-multiplexing strategies of the NoC employing the multipath routing (STEFAN; GOOSSENS, 2011). Their objective is to force the messages to be routed on multiple disjoint paths, which brings a non-deterministic communication behavior. The choice of the path can be made according to a pre-defined schedule, or randomly at run-time. This work uses the Æthereal NoC (GOOSSENS; DIELISSEN; RADULESCU, 2005) as a study case, analyzing the additional hardware cost and performance results of the proposed mechanism.

Æthereal employs source routing of packets, which means that the route of each packet is completely described by the sending Network Interface. The route, stored in the packet header, is read from a table of routes inside the NI and contains the list of turns the packet must take at each hop in order to reach its proper destination. To ensure freedom of collisions, each connection is allowed to transmit only at specific moments in time given by a schedule also stored into the Network Interface. The modifications consist of an extension to the table of paths to accommodate the new extra paths for connections and a selection mechanism for those paths.

Results showed that with a static path selection the allocation overhead is very low (about 3%), while the dynamic mode the overhead can be as high as 584%. They conclude that dynamic allocation should be applied to the NoC only for some of the communication channels.

3.5 Considerations

Several techniques to attack complex hardware systems were presented in this chapter. The first part targeted on cache attacks, where some powerful attacks and optimizations were described in details. This information aimed to explain how the attack could run in any hardware system, and it is important to have a clear understanding on the developed attacks of chapter six.

The timing-based attack are a statistical approach, where a high number of encryptions are required. Besides, the technique demands complex calculations, such as big averages values and variances. In a low performance processor with limited memory, typical scenario from MPSoCs, this attack could be unpractical. Then, to apply such technique, some adaptations are required.

The access-based attack from Osvik is a very powerful attack, where few samples can easily reveal the whole key. The main condition to implement such attack is the close hardware access, which is a normal feature from MPSoCs. Current MPSoCs have simplified software layers, which allow applications to access the hardware in a very close way.

The collision-based attacks take advantages of a particular timing behavior forced by the attacker. The theory behind the technique is very effective, but in practice several execution issues complicates the attack. One important issue reported by Spreitzer and Plos (SPREITZER; PLOS, 2013) was the cache line size. If the cache line size is bigger than sixteen, the probability of success of the attack is below 10%. The main reason regards the fact that almost all T table values are already in the cache, then few cache misses occurs. Therefore, this type of attack must be carefully analysed considering the target hardware platform before any implementation.

The NoC timing attacks have been cited by some authors, but only the possibility of such leakage exploration was demonstrated. No attack details, like threat model or implementation methodology, has been presented. Consequently, there are questions around the feasibility of this kind of attack, and how it can break a cryptography execut-

ing in some system. The chapter five explores this open issue and presents a structured information of this attack, based on observation and experimentation.

The state-of-the-art of countermeasures against architectural channel attacks has been exploring secure-enhanced mechanisms implemented directly in hardware. The main victims of current MPSoCs, the cache and the NoC, are the typical targets to provide the system security. Although many authors presented efficient security solutions, no work has studied and explored such efficacy in a real attack inside an MPSoC environment. The present thesis investigates new forms of attacks and evaluate how efficient can be the security mechanisms in the system.

4 EXPLORING THE NOC TIMING ATTACK

As presented in chapter 3, there is no publication explaining in details the NoC timing attack. The state-of-the-art only showed that an attacker could infer sensitive traffic information through communication interference. Hence, the NoC timing attack lacks essential information, such as i) the threat model; ii) the attack methodology; iii) the evaluations regarding the aspects of its implementation; iv) a practical execution.

The next section presents a study around NoC timing leakage, to understand its behavior and what are the conditions to compose the threat model. Then, the following sections show the threat model and the workflow (methodology) of the attack. The last section explores the influence of some NoC parameters in the attack.

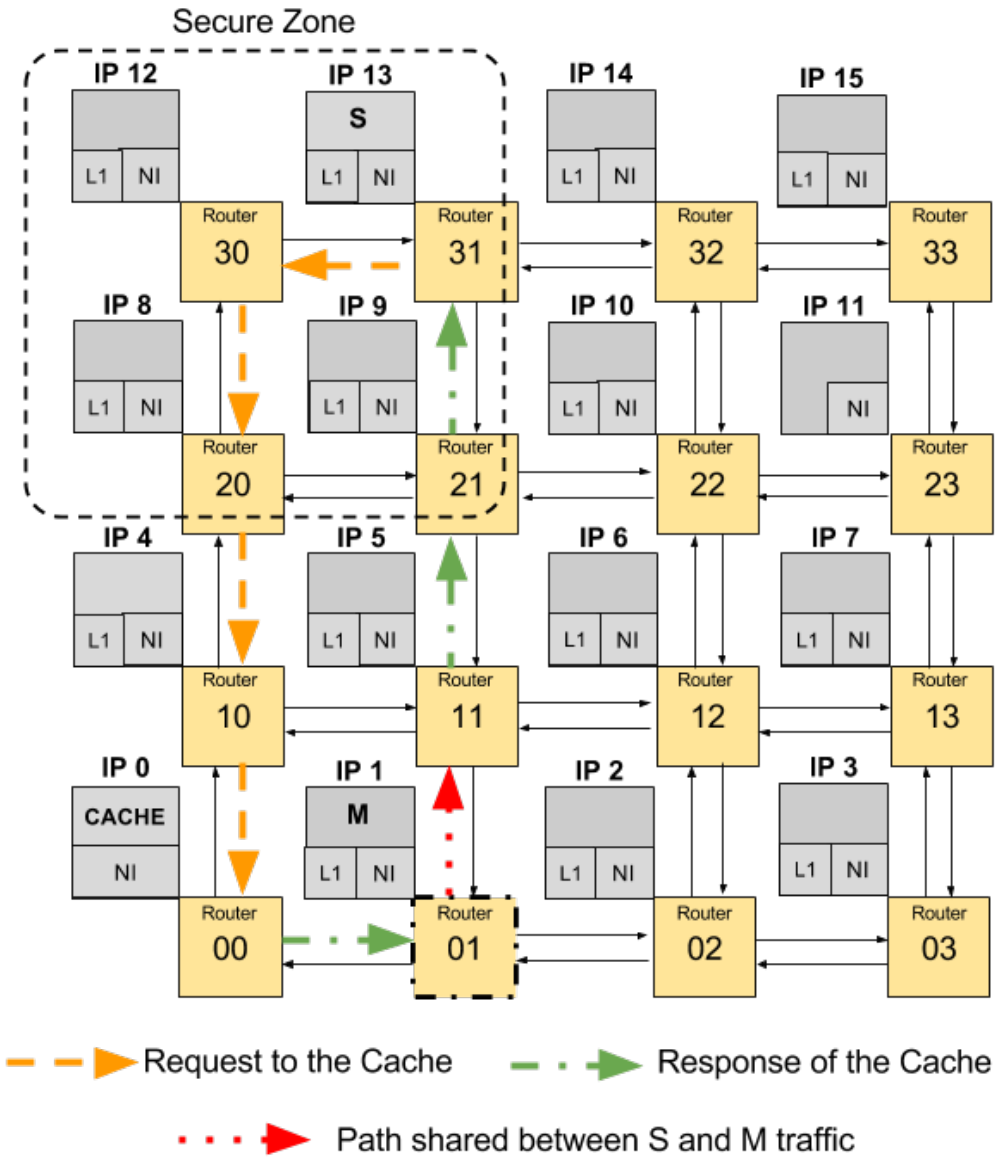
4.1 Understanding the NoC Leakage

The work of (MANCILLAS et al., 2014) showed that large packets could characterize the sensitive information. During a burst communication of a sensitive packet in the NoC, a significant part of the bandwidth is consumed. This value depends on the cipher block sizes and NoC link size. As a result, the sensitive packets degrade the throughput more abruptly than regular packets. Therefore, the throughput observation is a source of leakage in the system, which is the central concept of the NoC timing attack.

A scenario is demonstrated on figure 4.1. This scenario considers sensitive (S) and malicious (M) processes which are executed simultaneously at the MPSoC. S represents a performance-oriented AES cryptographic function, whose processor is placed at IP_{13}. When the data requested by the processor at IP_{13} is inside the local cache $L1_{13}$, a hit takes place, and the data is transmitted to the processor. Otherwise, a miss occurs, and an access to memory L2 located at IP_0 must be performed. As a result, a sensitive communication must be carried out through the NoC from IP_{13} to IP_0. The sensitive traffic must follow the deterministic path (sensitive path) determined by the well-known XY routing, which includes five routers ($Router_{31}$, $Router_{30}$, $Router_{20}$, $Router_{10}$ and $Router_{00}$). The response message, from IP_0 (cache L2) to IP_{13}, uses five routers ($Router_{00}$, $Router_{01}$, $Router_{11}$, $Router_{21}$ and $Router_{31}$).

Simultaneously, M is being executed in the infected IP_1, which has been carefully selected by the attacker for being located in the sensitive path ($Router_{01}$). The infected IP_1 may try to extract/infer data, modify the system behavior (by infecting other IPs) or

Figure 4.1: Example scenario of a NoC timing attack running in an MPSoC.

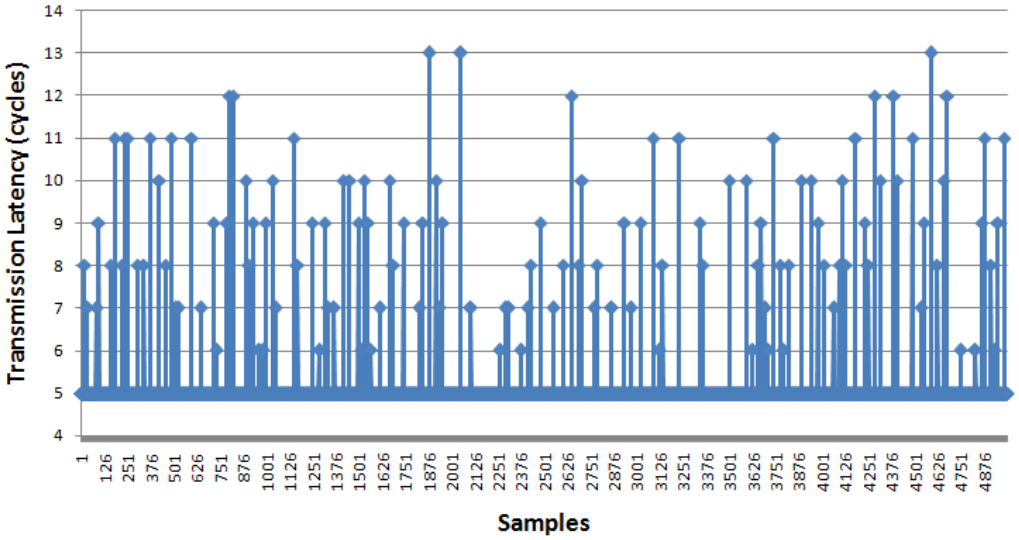


deny the MPSoC service employing malicious transactions. However, in our scenario, M injects frequent and useless transactions to saturate a particular output port of the $Router_{01}$, as observed in figure 4.1. Due to a collision in the North output port of this router by the malicious and the sensitive data, the throughput degradation of M (IP_1) can be used to infer the access pattern of the sensitive flow.

The malicious software M creates random packets and injects data continuously. The malicious packets are addressed to IP_5, because it creates an intersection traffic and the destination is outside any secure zone. Then, this useless traffic is not mitigated by some security mechanism in the system. Before and after sending any message, M samples the time of a cycle accurate timer. Then, it calculates the difference and stores the data for further analysis. Figure 4.2 shows the acquired times. One can observe that the

typical latency of the network interface is five cycles. The transmission latency above five cycles reveals the influence of the traffic passing through the target router. Different packet sizes, source and destinations can be present, where the sensitive messages also are in it. In order to maximize the correct guess of the sensitive traffic in such trace, the calibration step become strategic. Next subsections discuss the conditions and the methodology to implement this attack under different circumstances.

Figure 4.2: Samples captured by a malicious software running on an MPSoC.



4.2 Threat Model

Any attacker requires in advance environment information to execute the NoC timing attack. Besides, the victim system must allow some actions to enable the malicious software to perform this attack. These two issues are what defines the threat model of the NoC timing attack. The conditions to attack are:

1. System Susceptibility: An attacker can infect the system through a malicious software inside the MPSoC.
2. Attacker Reach: The attacker can communicate with the targets, even indirectly.
3. Sensitive Path Shared: The attacker observation path intersects the sensitive path.

System Susceptibility: This condition refers to the susceptibility of the MPSoC to be infected. There are techniques that an attacker can use to tamper the software and affect an IP (FIORIN; PALERMO; SILVANO, 2008). By using malicious software (malware)

that performs read and write transactions in forbidden memory areas, an attacker may change the behavior of an IP (victim IP) and turn it into an infected IP. Moreover, buffer overflows and other similar techniques address software weaknesses can be exploited for such a purpose (FIORIN; PALERMO; SILVANO, 2008).

Attacker Reach: This condition refers to the ability to communicate with the primary targets of the attack, for example, the victim processor and the shared cache. The first task after infection is to know the logical addresses of the elements. Typically, any MPSoC will provide at least system functions to ask jobs from other IPs. Therefore, three options are possible that makes this condition possible: i) the logical addresses are provided in MPSoC datasheet; ii) functions from an API performs the communication with the target IPs, or iii) functions from an API ask the service to a centralized manager, and it passes to the destination IP. The third option considers a high secure system, where the trusted IPs are entirely isolated. The objective here is to trigger the victim to perform the sensitive operation, like encryption.

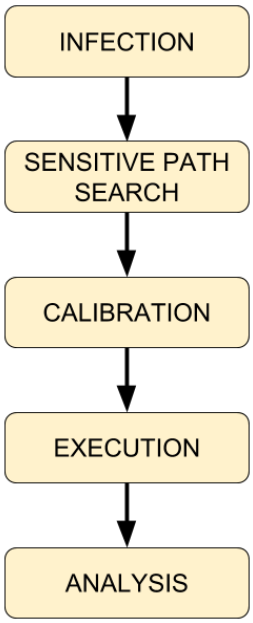
Sensitive Path Shared: To observe traffic interference caused by the victim, it is implicit that the malicious software shares some part of the sensitive path. The malware can ask encryptions to the crypto-processor and at the same time ask another task to other IP. The objective is to test which communication traffic intersects the sensitive one systematically. If the attacker does not find any interference, a different IP must be infected, and the search remains. Since AES provide a particular cache access pattern, it is practical to search the sensitive traffic. If the MPSoC provides more detailed information regarding its implementation, such as topology and routing algorithm, this process can be optimized.

4.3 Attack Methodology

A methodology is proposed to guarantee an attack with quality, having a high probability of success. The flowchart in figure 4.3 summarizes the five steps of the attack.

The first step is the infection, where the attacker downloads a malicious software into the MPSoC. The number of processors that can be infected depends on the target system. A high number of processors contaminated allows the attacker to find the best place to observe the sensitive traffic. It is expected that a portion of the MPSoC will be available to external applications since the interoperability between devices and external

Figure 4.3: Flowchart of the five steps to perform the NoC Timing Attack.



network are the major demand in the trends.

The second step is the sensitive path search. After the infection, the malware has to verify which place provide the best observational point, which means, are in the intersection of a sensitive traffic. This step can be made by trials systematically.

The third step is called calibration. This task tries to maximize the observation of the attacker. Since this technique uses throughput degradation to collect information, it is important to obtain a clear view of the target traffic. The calibration aims to find an injection rate that creates the minimum congestion to identify a sensitive message. If the attacker injects too fast, the leakage channel could be saturated, and any packet (small and big) would be identified. As a consequence, the false-positives would increase. Next subsection describes in details the process of calibration.

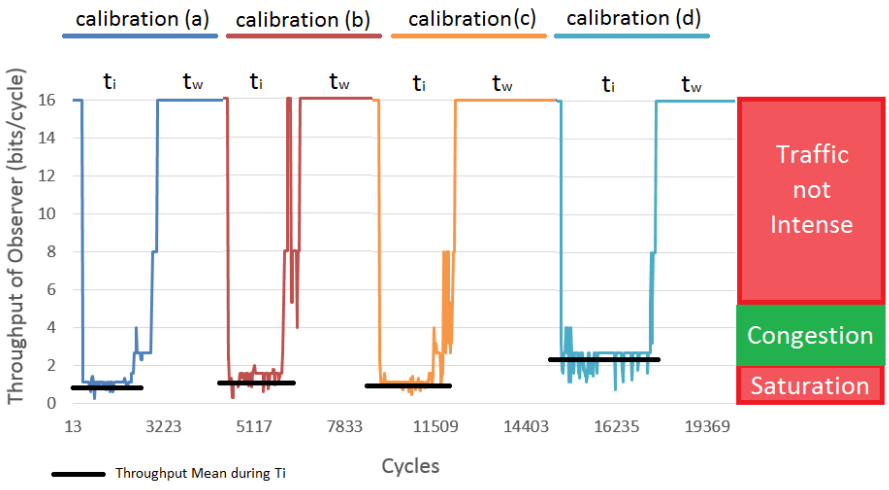
The fourth step performs the attack. After infects the correct processor, and identifies the proper injection rate, the malware does the execution. It injects data, and annotate the times to send each message. All samples are stored in the local memory and further sent to outside.

The last part of the NoC timing attack execution regards to the analysis. If implemented as a conventional timing attack, it will perform a mathematical algorithm to correlate the timing results collected by the monitor to infer an unknown key. Some techniques to correlate timing information are demonstrated at (JAYASINGHE et al., 2010), (KOCHER, 1996) and (TSUNOO et al., 2003).

4.3.1 Calibration

A calibration stage is required to increase the effectiveness of the NoC timing attack. The calibration aims to find the correct injection rate of the attacker for a target system. A good injection rate is when the attacker has the maximum detection of the sensitive packets without false-positives. Each MPSoC has different implementations and communication behaviors given by the environment. So, the calibration adjusts the attacker for each possible scenario before the attack to maximize its sensitivity. The calibration is performed by the infected IP (IP_1). First, it starts at a high injection rate (above 90%) and annotates the throughput in the usual behavior of the system. Then, it calculates the mean throughput. This process is repeated several times, each time reducing the injection rate. The calibration stops when the observed throughput is not affected by the attacker, due to internal buffers. Figure 4.4 presents a calibration process with four scenarios as examples. Each one was stimulated with a specific injection rate, performing a high number of experiments, in order to retrieve a mean value of the throughput sensed by the attacker. After these calibrations, Fig. 4.4, the (d) is the best configuration for the attack, because its mean value for the throughput result in a traffic congested but not saturated. This scenario allows the infected IP to have a clear detection, demonstrated by the spikes in the throughput. In the other scenarios, the traffic seems to be too saturated, so the peaks that represent noise and sensitive packets can not be easily distiguished, since they are at the same level. This mean value of the chosen scenario works also as the threshold to detection algorithm.

Figure 4.4: Throughput Trace sensed by attacker of four calibration scenarios: a) Injection rate of 70%; b) Injection rate of 50%; c) Injection rate of 40%; d) Injection rate of 30%.



4.4 Expanding the Attack to a Distributed Attack

One can infect multiple cores at the same time to improve the effectiveness of the attack. Compared to the single timing attack, the distributed timing attack (DTA) decreases the computation and storage requirements of the infected IPs. Besides, a distributed strategy can overcome countermeasures that exploit the routing of the packets, because different routes can be tracked. In DTA, malware is divided into two groups:

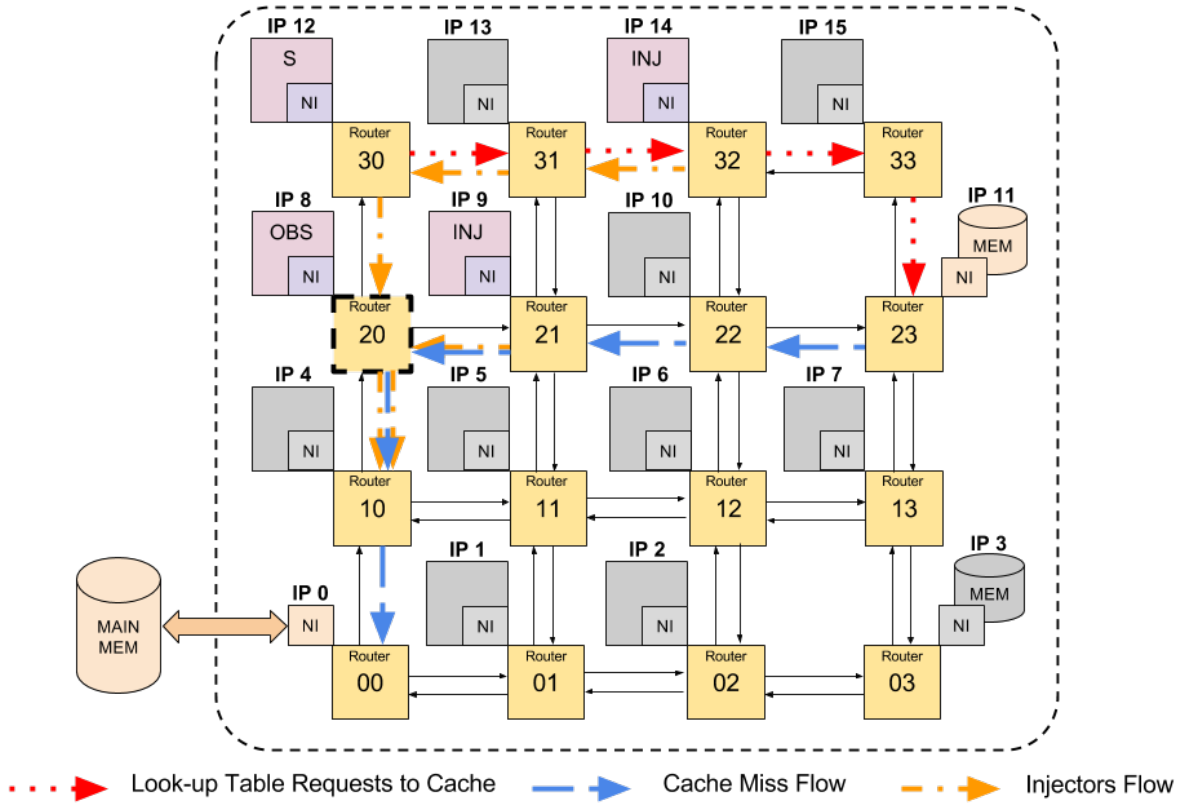
- *Injectors*, responsible for injecting data in the NoC with the objective to increase the congestion of the target path;
- *Observers*, responsible for injecting data at lower data rates and to collect the throughput traces of the target path.

Figure 4.5 presents a different attack scenario. This MPSoC comprises processors and IPs also interconnected by a mesh NoC. The memory organization used is the distributed one. In this case, the caches of the system (IP_{11} and IP_3) can be managed separately according to the application. Thus, sensitive tasks can run on an isolated cache memory (in this example IP_{11}), being a very secure strategy. However, the distributed caches require communicating with the main memory, located at IP_0. Then, this DTA example targets the communication between the sensitive cache and the main memory. In such scenario, the sensitive traffic is more difficult to be detected by a single NoC timing attack, since this kind of communication does not use a big bandwidth.

DTA follows the same methodology of the single approach. Firstly, the system is infected, and the attacker searches for the intersection paths. This attack requires more than one contaminated IP at the same time, so the condition of $susceptibility$ has to be higher (more than one processor can be infected). The attacker has to explore the number of *injectors* and the injection data rate of each one. This attack space exploration can be an exhaustive task. Once calibrated, the *injectors* will send packets while the *observer* will sample and store the throughput results.

The example in figure 4.5 shows a sensitive application S starting an encryption running on the IP_{12}. This crypto-processor accesses the secure cache memory (IP_{11}), to retrieve the look-up tables used in the cipher algorithm. When there is a cache miss at this memory, it creates another communication flow requesting data for the main memory. Three processors infected works on the attack. One *observer* at IP_8 (labeled as *OBS*) and two *injectors* at IP_9 and IP_{14} (labeled *INJ*). These malicious IPs send messages

Figure 4.5: MPSoC system running a sensitive application, after infection stage.



to IP_4, to create the desired congestion in the path used for cache misses requests. As observed in figure 4.5, the infected IPs work collaboratively to capture the sensitive traffic. Information regarding cache misses can be used for different attack approaches, such as access-based ones.

4.5 Evaluation

Regarding the NoC timing attack, some issues remain unclear when implementing in practice. How much other shared memory messages disturb the attack; which place is better in the NoC to observe the sensitive traffic and; what is the sensibility of the attack under different message sizes. These three issues were evaluated, and the results are presented in the following subsections.

4.5.1 Traffic Interference

The objective of this experiment was to understand how accurate is the detection of sensitive packets under traffic interference. Typically, the NoC timing attack aims to uncover the traffic between a victim IP and a memory element. The non-desired traffic beholds to the other IPs running applications that access the same memory.

To develop this experiment, the attacker aimed to reveal the traffic between a crypto-processor running an AES encryption accessing a shared cache. Then, six scenarios were designed, each one with a different accessing rate. This rate is the period of the non-desired traffic accessing the same shared cache. These interference packets can be interpreted as an IP that accesses periodically, or mutual IPs accessing the shared cache, creating a periodic interference. The scenarios used the following parameters:

- Scenario 1: Period of non-desired traffic at 500 microseconds.
- Scenario 2: Period of non-desired traffic at 250 microseconds.
- Scenario 3: Period of non-desired traffic at 100 microseconds.
- Scenario 4: Period of non-desired traffic at 50 microsecond.
- Scenario 5: Period of non-desired traffic at five microseconds.
- Scenario 6: Period of non-desired traffic at one microseconds.

For this analysis, two metrics were defined, the detection efficiency and the false-positives. The detection efficiency was given by the amount of the sensitive traffic detected. The false-positives referred to the wrong samples recognized.

Each scenario was experimented by the time to encrypt one plaintext, which varies from 300 microseconds to 450 microseconds. The attacker asked to the crypto-processor a random plaintext to be encrypted and performed the attack. During the experiment, another IP generated random requests to the shared cache according to the period defined for such scenario. This interference IP was placed in a location that the message intersects with the observation point of the attacker. Since it was a controlled environment to evaluate the NoC timing attack, our sensitive packets were monitored, to compare with the attacker's results. The detection efficiency and false-positives results are presented in table 4.1.

The attack detected about 60% of all sensitive accesses in the worst scenario. This 60% represents that the noise in the communication behavior of the target MPSoC affected the decision of the attacker to distinguish between noise and access to the cache

Table 4.1: Detection efficiency and false-positives of the NoC timing attack under six different scenarios. Each scenario used a different interference period. System running at 100MHz.

Scenario	Interference Period (us)	Detection Efficiency (%)	False-positives (%)
1	500	100	0
2	250	100	1.1
3	100	99.4	2.8
4	50	95.9	5.7
5	5	77.4	42.7
6	1	60.7	74.1

memory. Besides, 74% of the detected packets were false-positives, which creates a prohibitive situation for such attack. On the other hand, scenario 1 to 3 detected essentially all the sensitive packets correctly. The influence of the interference happened when the period of access was lower than the AES encryption time. Then, the access occurred during the encryption inserting noise in the detection. There is a direct relation between detection efficiency and false-positives because the noise traffic has the same behavior of the sensitive one. The same behavior of the messages is justified by the standard response of the shared cache. Therefore, the experiment concluded that the NoC timing attack could be compromised in the presence of non-desired traffic. Two conditions have to be met to interferes the attack:

- The communications that access one of the victim IPs intersects the attack observation path. This is defined as non-desired traffic.
- The period of the non-desired traffic lower that the attacker observation time.

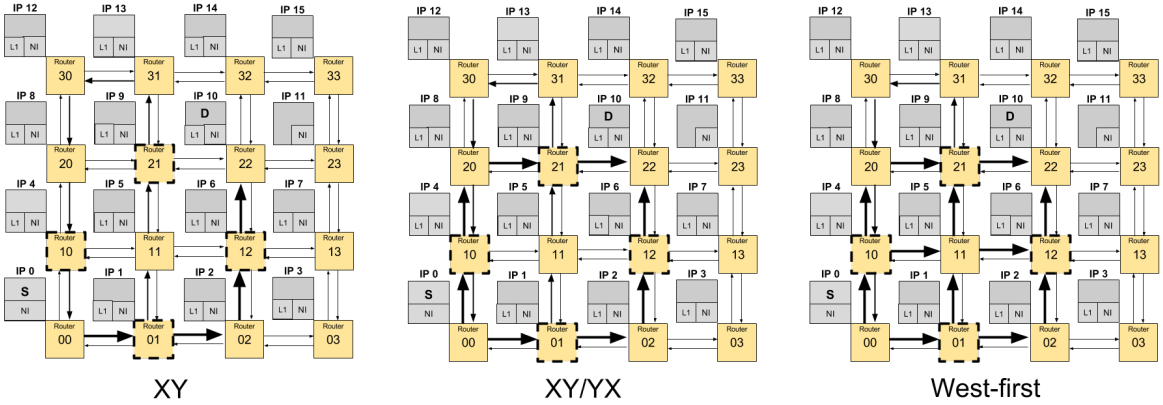
4.5.2 Placement in the Network

The second issue evaluated was the placement of the attacker in the NoC. For this experiment, it was considered that malware could infect any processor in the system. Besides, different routing strategies were used to enable a better understanding around the placement. Three scenarios were tested at the same MPSoC platform, each one with a different routing algorithm (figure 4.6):

- Scenario 1: Deterministic XY
- Scenario 2: Alternated XY and YX (random)
- Scenario 3: Adaptive West-first

Four routers were chosen to perform the tests, router at IP_1,IP_4,IP_6 and IP_9.

Figure 4.6: Placement experiment scenarios. The dashed routers are the experiment targets. S is the source and D the destination of the sensitive traffic. Highlighted arrows shows the allowed route at each scenario.



Two close to the source, and two close to the destination. The observation points were the output ports shared with the sensitive traffic. The source and destination of the sensitive traffic are identified by S and D respectively (figure 4.6). Experiment results are detailed in table 4.2.

Table 4.2: Placement experiments results under three routing algorithms.

Scenario		Attacker Observability	Victim Port
XY	Router 1	100%	East
	Router 4	0%	East
	Router 6	100%	North
	Router 9	0%	East
XYYX	Router 1	24.55%	East
	Router 4	75.45%	North
	Router 6	24.55%	North
	Router 9	75.45%	East
WEST FIRST	Router 1	57.73%	East
	Router 1	17.24%	North
	Router 4	16.68%	East
	Router 4	10.08%	North
	Router 6	74.39%	North
	Router 9	25.61%	East

For the deterministic XY, the output ports attacked that intersected the sensitive path were able detect all the packets. However, when XY and YX were used, the proportion of the observation at each port was related to the amount of traffic for each route. Since only two paths were possible, one observer was not enough to retrieve all sensitive traffic. The best results were the attacks on the east port of the router 9 and the north port of the router 4. However, such results are highly dependent on the routing arbitration. In

the last scenario, the West first algorithm has six route possibilities. Hence, the results of observability are very low, since the messages became spread in the NoC through different routes. However, the routers close to the destination were able to collect almost all the data, because all routes converge in the end. Considering all the proposed scenarios, one can conclude that there are more advantages in place the attackers near the destination.

The experiment can be extended to an analysis of the Distributed Timing Attack. The detection rates of the analyses resulted in the following table for the West-first routing algorithm 4.3. The west-first was chosen to this analysis because the single timing attack was less efficient in a scenario with several possible routes.

Table 4.3: Distributed Timing Attack under west-first routing algorithm.

Attackers	Detection
R1 N + R4 N	27,33%
R1 N + R4 E	33,93%
R1 E + R4 N	67,81%
R1 E + R4 E	74,41%
R1 E + R9 E	83,34%
R4 N + R6 N	84,47%
R6 N + R9 E	100,00%

Using two observers, only using the closest routers was possible to retrieve all the sensitive traffic. Other possibilities could detect only 84%. In the case of three observers, the attack would succeed without the requirement of the closest routers. However, the execution of an attack with more than two observers becomes very challenging, due to synchronization issues.

4.5.3 Size of the Packet

The detection of traffic collisions is a result of the sensitive and attacker traffics. The attacker traffic, given by the packet size and injection rate, is under control. But the sensitive traffic is not. Typically, the sensitive traffic uses big messages, because they are, usually, memory requests. However, if the sensitive traffic varies its size, the detection will be affected. This experiment evaluated the influence of different packet sizes in the attacker observation. In this scenario, one flit is a 32-bit word. Table 4.4 presents the results.

Big packets take more time to send a complete message, causing more traffic interference. Hence, 64 and 128 flits allow an attacker to observes all the sensitive traffic.

Table 4.4: Relation between packet size and detection rate.

Pkt Size	Packets Detected
128 flits	100,00%
64 flits	100,00%
32 flits	97,99%
16 flits	58,87%
8 flits	2,47%
4 flits	0,00%

Cache memories exchange information regarding the cache line size, which varies from 4 to 32 words usually (one word equals one flit). The communication with the main memory is performed in blocks of data, which ranges from 64 to 512 words often. Therefore, the trend to increase the size of the memories inside MPSoCs will provide more and more information to attacks. The size of the packets will not decrease because the performance penalty is too high to justify.

4.6 Considerations

The present chapter is very relevant because it elucidates the NoC timing attack. The experiments and analysis allowed one to define the threat model and attack methodology. Besides, the study proposes a significant step in the attack, the calibration. It is a kind of characterization of the attackers, to understand the target traffic and maximizes the attack. Using the calibration step make possible to perform an attack with multiple infected IPs, defined as Distributed Timing Attack (DTA). Another valuable contribution in this chapter are the evaluations of the influence of traffic interference, placement in the network and size of the packet in the attack.

Since the behavior of the message from one shared IP tends to be the same for any request, if different IPs access the victim IP, it could affect the efficiency of the attack. However, to compromise the attack, the interference packets have to intersect the observed path and communicate at a rate higher than the observation window. Even being difficult, it is possible that all these conditions could be satisfied in current MPSoCs since it depends on the application behavior. If the attack presents low efficiency, one can try a different placement in the network to overcome this issue.

The placement study revealed that if the routing algorithm is deterministic, any place that intersects the victim path works. But, if an adaptive routing or another different strategy is performed (a non-deterministic behavior), more than one observer would be

necessary. In this case, the best placement is near the target destination, because all the packets join in the end node.

The third analysis presented the relation between the observation capacity and message size. The NoC timing attack identifies the sensitive packets when a collision is sensed in the router. The sensitive message has to occupy the router at least the time to the next malware transmission to sense such collision. If the sensitive packet is too little the time in software to ask a new transmission became higher, and then, no contention happens. This metric has a relation between the data rate that the malware can inject packets and the buffer size. If future software layers from MPSoCs handle the communication, the increase of the delay between each transmission could make this attack unpractical. This upcoming feature is possible but not expected, because it would decrease the benefits of bandwidth and throughput provided by the NoC.

5 DEVELOPED ATTACKS

One of the objectives of this thesis is to investigate new forms to compromise MPSoCs. ACAs described in the state-of-the-art show different approaches to attack caches, one important shared resource. The NoC is also a shared component mentioned as a potential source of leakage. However, there is no publication giving details on how to attack the system through the NoC.

The leakage explored by the NoC timing attack can reveal relevant information of the system. For example, the attacker can annotate the moments that occurs the victim operations, being a timing channel. Another tactic is to observe the quantity or order of the transactions and use it as an access channel. These two strategies were combined with ideas from the literature to develop four attacks. They can be organized into three categories:

- Timing-based:
  - Hourglass Attack
- Access-based:
  - Firecracker Attack
  - Arrow Attack
- Collision-based:
  - Earthquake Attack

5.1 Hourglass Attack

This attack was based on the attack of Bernstein, which uses two main steps: statistical signature acquisition and correlation. The only difference is the leakage source. Bernstein used the total encryption time of the application, while Hourglass uses the time obtained through the NoC timing analysis. Using the NoC timing attack, it is possible to track the access of the encryption to the shared cache, and then calculate the time to perform the first round (the time of sixteen accesses). Hourglass can isolate the latency related to the first round, being more dangerous than Bernstein. Even mismatches in the NoC observation do not avoid the statistical strategy of Hourglass.

Hourglass threat model and methodology are extended versions of the NoC timing

attack (Chapter 5). The following subsections present them.

5.1.1 Threat Model

The following conditions are required to execute the Hourglass attack:

1. System Susceptibility
2. Attacker Reach
3. Sensitive Path Shared
4. Shared Cache Access
5. Plaintext Knowledge

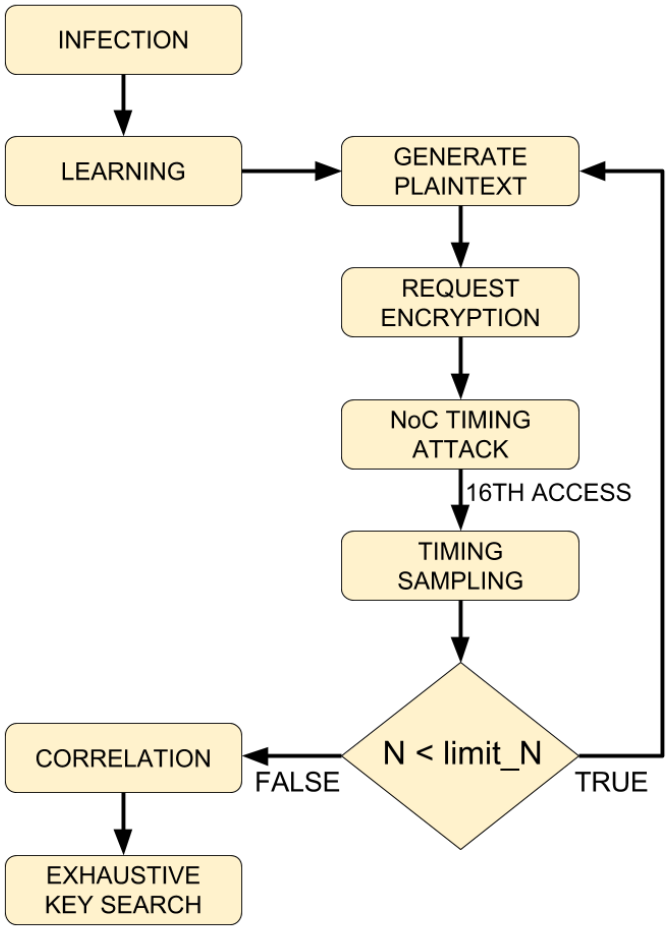
The first three conditions are presented in Chapter 5 belonging to the NoC timing attack threat model. The fourth item is the shared cache access, which is necessary to develop the learning phase. The statistical models generated in the learning step must be executed in the same environment as the victim to obtain an excellent efficiency in the attack. Therefore, to reproduce the behavior of the victim, the attacker needs access to the same shared cache. A different cache can be used if the configuration of both is similar, like the cache line and the mapping strategy (direct or set-associative). The last requirement from the threat model, plaintext knowledge, requires that the attack knows the plaintext used in the encryptions. This information is important to perform the reverse engineering to retrieve parts of the key. Usually, the attacker can generate the information that will be encrypted.

5.1.2 Attack Methodology

The flowchart explaining the execution of the Hourglass is presented in figure 5.1. The attack includes eight steps.

Infection: The purpose of the infection is to install the malicious application inside the MPSoC. In this stage, one or multiple IP cores of the MPSoC are infected. Then, the attacker gains access to the shared resources, such as the NoC and the shared cache.

Figure 5.1: Hourglass methodology flowchart.



Learning: The objective is to create a statistical model of the timing required to encrypt any possible byte with a particularly known key for the first round. This model is called signature. Hourglass performs the learning phase only for the first round of AES. Besides, it is made inside the victim environment, configuring the online learning. The main issue here is to perform the learning phase inside the target MPSoC, to achieve the correct statistical models of the target system. It does not require to run inside the victim IP to obtain a good model because the cache behavior will be the same independently of the IP.

Generate Plaintext: The objective of this step is to generate a random plaintext that the attacker knows it. The infected IP is now able to request encryptions to the AES core.

Request Encryption: The attacker sends the generated plaintext to the AES core, requesting a new encryption. Depending on the system, this task is made directly or using an API of the system.

NoC timing attack: The attacker performs the NoC timing attack to identify the accesses in the cache by the AES algorithm. At this step, Hourglass only requires triggering when the first round has finished, which happens after the 16th access.

Time Sampling: The sampling step for the Hourglass is only to annotate the time spend for the first round, according to NoC timing attack trigger. A precise timer is required for the best performance of this attack. Then, the attack backs to the *generate plaintext* step, entering in a loop. The process repeats for a significant number of samples, required to improve correlation results. The N iterator controls the repetition. The limit of the iteration ($limit_N$) is defined during the attack, observing the correlation results.

Correlation: All collected information, the encryption times and plaintext values, are sent to an external computer that performs the correlation. In the same manner, like Bernstein, equation 3.2 is used to calculate the correlation results. The high peaks results are selected for the final step.

Exhaustive Key Search Finally, the best candidates are used for a final brute-force task to verifies all possibilities and recover the secret key.

5.2 Firecracker Attack

Firecracker follows the concept presented by (OSVIK; SHAMIR; TROMER, 2006) with the Prime+Probe attack. The objective is to use the NoC timing attack to trigger the end of the first round and then probe the shared cache. This attack can be much more efficient than the Prime+Probe since the early probe action avoids several accesses to the cache to interferes in the attack. Since the NoC timing attack may lose some sensitive traffic in the process, the attack analyzes the non-accessed sets of the cache. This feature also guarantees the probe works even if the AES continue accessing the shared cache. The threat model and methodology are presented in the following subsections.

5.2.1 Threat Model

The conditions to perform the Firecracker attack are:

1. System Susceptibility
2. Attacker Reach
3. Sensitive Path Shared
4. Shared Cache Access
5. Plaintext Knowledge
6. Shared Cache Knowledge
7. AES Tables Location Knowledge

The first three conditions are the same presented in the NoC timing attack chapter. Then, the items four and five follows the same objective of the Hourglass attack. The novel conditions, six and seven, are required to know the positions in the shared cache that will be filled with the AES T tables. The relevant information regarding the cache is the total size, the type of mapping (direct, associative or set-associative), and the size of the cache line. Usually, this information is provided in the datasheet of the device.

Usually, the system will not provide the location of AES tables in the main memory. However, in literature, some works propose techniques to find such sites (IRAZOQUI; EISENBARTH; SUNAR, 2015) (GULLASCH; BANGERTER; KRENN, 2011).

5.2.2 Attack Methodology

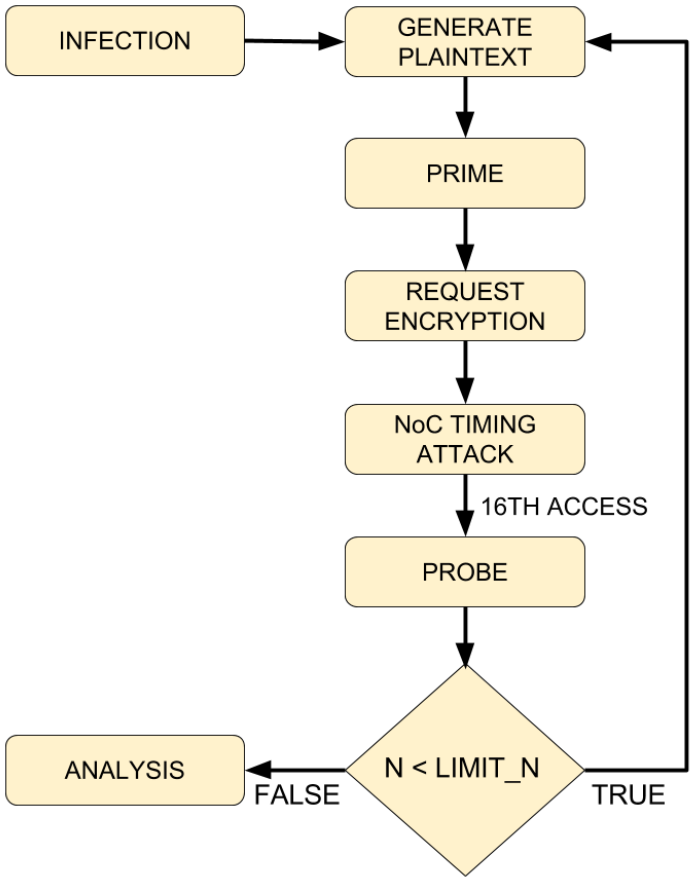
The Firecracker attack comprises eight steps. The flowchart of figure 5.2 presents the sequence of actions.

Infection: The *Infection* starts when the attacker stores a malware into the MPSoC.

Generate Plaintext: The attacker generates a known random plaintext to be encrypted by the crypto-processor. This plaintext will be used in the analysis step to calculate the key guesses.

Prime: The *Prime* consists in the preparation of the cache by the infected IP. The goal is to guarantee that there are no AES lookup tables in the cache before the attack. By spreading a random vector created by the attacker in the cache, the attacker overwrites all cache locations aligned with the T tables.

Figure 5.2: Firecracker methodology flowchart.



Request Encryption: With the cache prepared, the attacker asks an encryption with the known plaintext to the target cipher IP.

NoC Timing Attack: The infected processor start sending packets to the NoC and it monitors its throughput to detect the AES accesses to the shared cache. After the sixteenth access, the attacker starts the probe step.

Probe: The *Probe* aims to verify the accessed cache locations during the AES execution. After the trigger from the NoC Timing Attack fewer, the infected IP core fetches some parts of its random vector. It reads one data of each set, annotating if it was used or not. Longer fetching times reveal cache misses, thus used memory locations. Then, the plaintext and the information of the probing are sent to an external agent to compute the analysis. Then, the attack returns to the prime step, repeating the process until all possible bytes are excluded using the analysis step. The iteration uses N as iterator variable. The limit $LIMIT_N$ is given by the amount required to discover all the key. Initially, this limit is guessed and then adjusted after the analysis process.

Analysis: The *analysis* in this attack can be a post-processing or run-time step. In the flowchart, the post-processing flow is depicted. For each plaintext and cache usage information, the external host calculates the key possibilities. Firecracker is an attack that aims to check the non-used sets of the shared cache. This strategy is the same presented by Xinjie Zhao (XINJIE et al., 2008). The values generated after the first round follow the expression $x_i^0 = p_i \oplus k_i (i = 0, ..., 15)$. Therefore, by testing the data acquired in the *Probe* step, it is possible to identify which sets have not been used by the crypto-processor. Consequently, the indexes have not been used as well. By assuming that $x_i^0 \neq p_i \oplus k_i (i = 0, ..., 15)$ and knowing the plaintext byte p_i, is possible to prove that $k_i \neq p_i \oplus x_i^0 (i = 0, ..., 15)$. Hence, possible key candidates can be removed. As much samples the attack provides, more candidates can be unconsidered, being possible to reveal all the key.

5.3 Arrow Attack

This attack uses the same principle of the Firecracker. However, as its name suggests, this attack aims to break AES very quickly with few encryptions. Arrow focuses on the identification of the used indexes of the T tables with high precision. A high accuracy environment is required to obtain success in this attack.

Arrow attack uses the NoC timing attack to identify all the accesses of a target T table during the first round. At each four access, the same T table is reached. So, the objective of this attack is to identify the access with precision and probe fast the cache. The probe has to finish before the next access of the same T table, meaning a time of three access.

5.3.1 Threat Model

The conditions to perform the Firecracker attack are:

1. System Susceptibility
2. Attacker Reach
3. Sensitive Path Shared
4. Shared Cache Access

5. Plaintext Knowledge
6. Shared Cache Knowledge
7. AES Tables Location Knowledge

As observed in the list above, the same conditions of Firecracker are applied for Arrow.

5.3.2 Attack Methodology

Although the methodology of Firecracker can be employed for Arrow, few changes inside the steps are necessary, besides an *exhaustive search key* step in the end. Figure 5.3 presents the sequence of actions for the Arrow Attack.

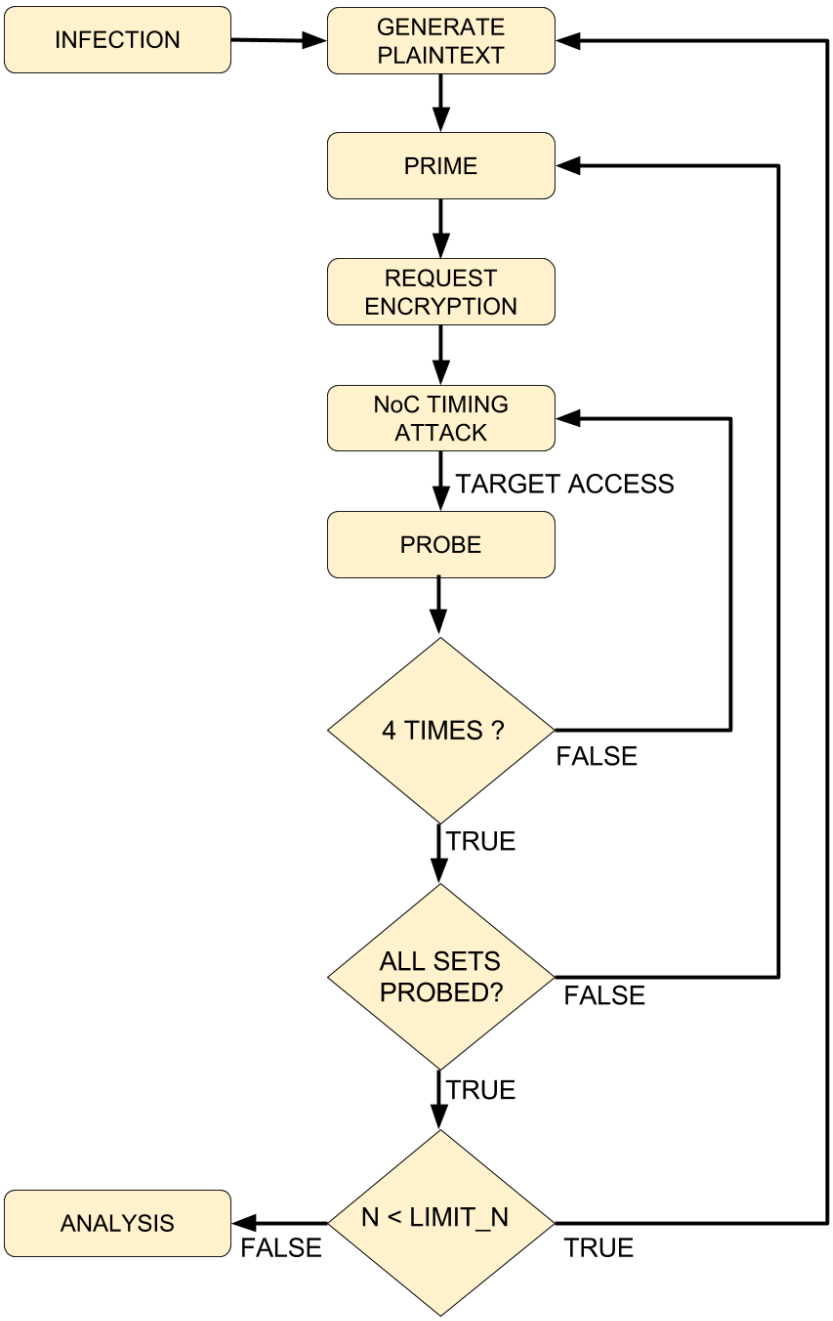
Only the modified and included steps are described below.

NoC Timing Attack: Arrow is an attack that aims to identify the used set of the shared cache at each access by the cryptographic algorithm. The NoC timing attack needs to identify all sensitive information transmitted without missing any traffic to work properly. At each four access detected, the attack triggers the probe step.

Probe: In AES algorithm, the pre-computed tables are accessed sequentially (T_0, T_1, T_2 and then T_3), which gives the possibility to probe the cache sequentially by the attacker. Then, to observes the next access, it awaits three accesses. As a consequence, this process must repeat four times (four access of the same T table in the first round). The objective of the probe step is to verify all possible sets for a given table (for example T_0) and find the set it was used. Depending on the number of sets, it may not be able to finish the probe step before the next access to the same T table. In that case, the process of the attack has to repeat, returning to the request encryption step. In this case, each new iteration will manage a different part of the cache. Since the plaintext does not change, the accessed index will remain the same. All the process is repeated N times. This iteration is defined by the analysis step, which identifies some samples required to reveal the key.

Analysis: The *analysis* step follows the original concept of Osvik (OSVIK; SHAMIR; TROMER, 2006), which relates the accessed set with the candidate byte of the key. We perform the first round analysis, where only the accesses during the first AES round are

Figure 5.3: Arrow methodology flowchart.



used for the reverse calculation. Using the expression of $x_i^0 = p_i \oplus k_i (i = 0, ..., 15)$, knowing the plaintext and the index, it is possible to calculate the key. However, each set accessed represents the group of addresses of the cache line. So, some elements in the cache line will generate the number of possible subkey candidates. Then, the attack needs to be performed with different plaintexts to eliminate these possibilities.

5.4 Earthquake Attack

This attack uses the differential collision strategy of Bogdanov (BOGDANOV et al., 2010) to break AES cryptography. The difference of Earthquake attack is that the NoC timing attack is used to trigger the gathering of the encryption time until the third round. The third round is the main part of the technique because it is when the five expected collisions happen in the presence of a wide-collision. The possibility to get the time of the third round execution create new potentials for this collision-based attack, which are discussed further in the experimental study chapter (Chapter 7). The threat model and attack methodology are presented in the following subsections.

5.4.1 Threat Model

The following conditions have to be met to perform this attack successfully:

1. System Susceptibility
2. Attacker Reach
3. Sensitive Path Shared
4. Shared Cache Access
5. Plaintext Control

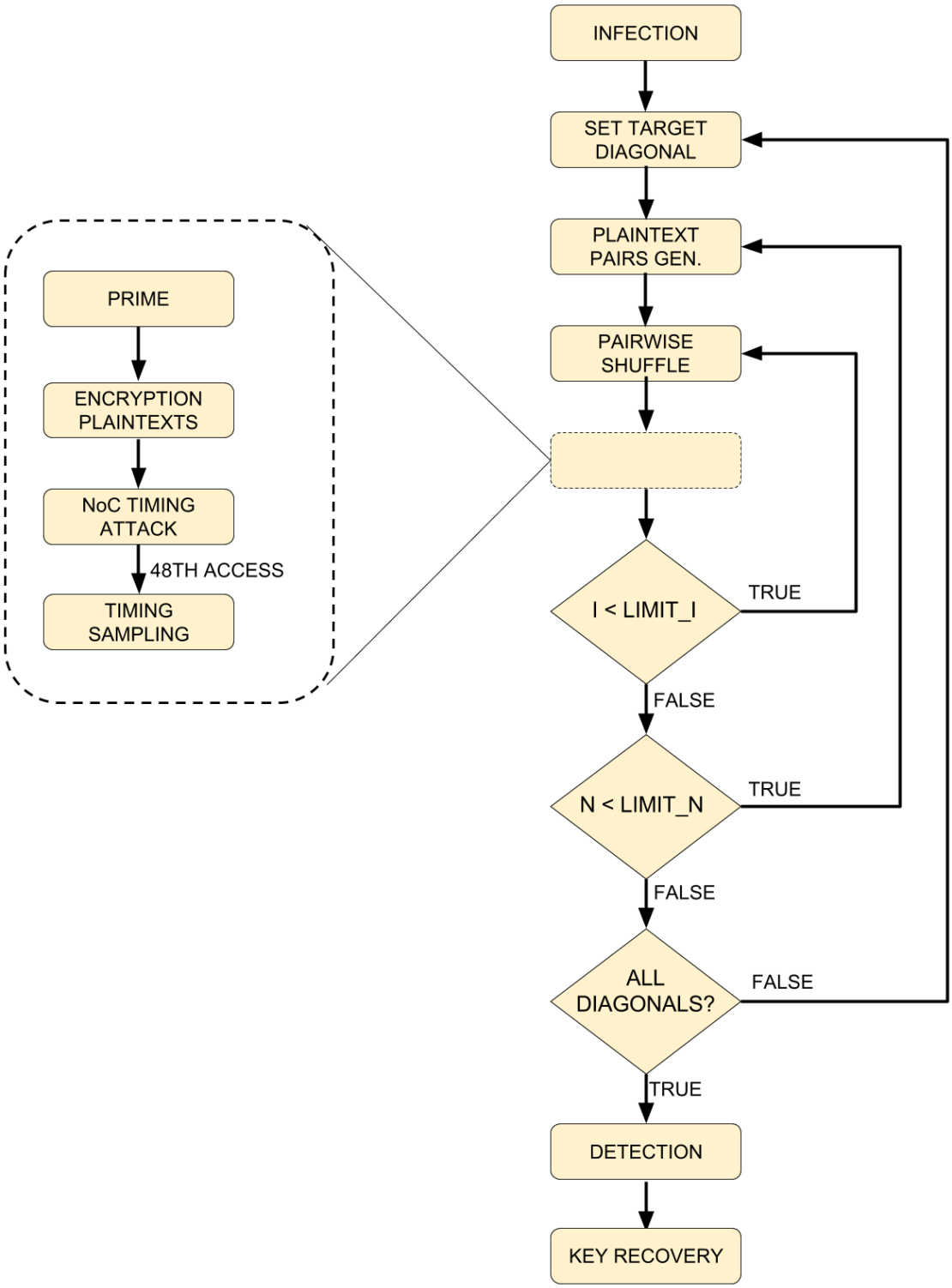
In this attack, the knowledge of the plaintext is not sufficient. The attacker has to generate pairs of plaintext according to the rule described in Bogdanov's work (BOGDANOV et al., 2010) (also presented in Chapter IV).

5.4.2 Attack Methodology

The methodology of Earthquake is composed of nine steps. Two loops in the flow iterates according to adjustable parameters. These parameters control how far will be the search for wide-collisions candidates. The flowchart in figure 5.4 shows the flowchart of the attack.

Infection: The first part of the attack is to infect an IP in the system that meets the conditions of the threat model.

Figure 5.4: Earthquake methodology flowchart.



Plaintext Pairs Generation: The malicious software generates pairs of plaintext according to the rule described in section IV. These plaintexts encrypted in sequence can provoke the wide-collision situation.

Pairwise Shuffle: This step changes the pairwisely equal elements of the plaintexts, to create different possibilities during the encryptions. This step is important to perform a deep search for a real wide-collision.

Prime: Before asking any encryption, the shared cache is filled with an attacker's data. The objective of that is to maximize the observation of cache hits and misses since all data in the cache will be initialized in the same state, without any T table's data.

Encryption of Plaintexts: The attacker sends the generated plaintexts to the AES core, requesting one encryption after another. The following step affects only the second encryption.

NoC timing attack: The attacker performs the NoC timing attack to identify the accesses in the cache by the AES algorithm. At this step, Earthquake only requires triggering when the third round has finished, which happens after the 48th access.

Time Sampling: Only the time of the second encryption is collected after the trigger of the NoC timing attack. Only the results below a given threshold are saved to optimize this step. Then, fewer data storage is required for this step. The collected data is organized as the following array $Collected[t, A_i, E_i]$. The t is the encryption time, the A_i is the array of the elements in the target diagonal from the first plaintext, and the E_i is the elements from the second plaintext. After this step, the process is repeated I times, modifying only the pairwisely equal values. Hence, it backs to Pairwise Shuffle step. Then, when I reaches its limit, the flow backs to the Plaintext Pairs Generating step, iterated by the variable N. This loop explores different diagonal combinations, to provide candidates for the detection stage. When the loop of N is make, the process must repeat for all diagonals, in the same manner as Bogdanov's attack. Then, with all data ready, the attack can advance to the next step.

Detection Stage: All the recorded data are then filtered with an even lower threshold. The key factor here is to select the minimum amount of candidates because the key recovery stage will explore all combinations in arranges of four against all subkey possibilities, which brings a complexity of $\binom{4+m}{4}^4 * 2^{32}$. However, few samples imply in fewer chances to find the correct key. This trade-off is the most challenging issue regarding this attack.

Key Recovery Stage: The input of this stage is given by $\binom{4+m}{4}$, which is the combination in arranges of four of the detected candidates. Each combination is checked against all 2^{32} subkeys. Using the equation 3.4, one searches over each subkey possibility which one generates a collision after the first round of all the four candidates in a group. As a result, each combination checked will output a subkey proposal. Then, all combinations of the proposals have to in an exhaustive search key step. This stage was performed offline in a program written in C.

5.5 Considerations

This chapter proposed four novel ACAs to break AES cryptography running on MPSoCs. All techniques explore the performance-oriented AES communication with a shared cache memory. Three approaches were explored, integrating the NoC timing attack: i) timing-based; ii) access-based; and iii) collisions-based.

The timing-based attack, called Hourglass, performs almost the same methodology from Bernstein, but it targets only the time to execute the first round. Hourglass is not only an optimization because if we analyze the computational requirements of the Bernstein, but the proposed attack is also more realistic to the target hardware platform. MPSoCs will provide limited processing elements, and limited memory storage, then, reduced operations will be crucial when implementing such attacks on mobile or IoT devices.

Firecracker and Arrow attacks are optimizations from the Prime+Probe attack by Osvik. They differ in the probe strategy, where Firecracker looks for remaining sets after the first round, and Arrow looks for the used set for each T table access. Therefore, Firecracker is more suitable for small shared caches. Small caches concentrate the tables T0, T1, T2 or T3 in the same sets. So, it is not possible to differentiate by the set which table was accessed. The Arrow attack focuses on bigger caches, where the T tables will be placed in different sets, allowing the attacker to analyze each table separately.

The last attack, called Earthquake, aims to optimize Bogdanov attack by profiling the wide collisions based on the first three rounds of AES. It is expected that Earthquake collects better samples, reducing the computational cost of the key search stage severely.

6 PROPOSED PROTECTION

This chapter presents a protection mechanism against NoC timing attacks, the Gossip NoC. The primary objective of the proposed countermeasure was to provide security with a small area and power overheads.

6.1 Gossip Network-on-Chip

Gossip NoC is a security enhanced architecture to deal with single and distributed timing attacks. It is composed by a traffic monitor and a counter-measure technique at each router, being a distributed security mechanism. The name *gossip* is used because the router monitors the traffic and generates alert messages to other routers when anomalies in traffic are detected, creating a *gossip message*. The gossip message is a dedicated signal included in the router port interface, where each cycle active represents a *gossip message*. Besides, to avoid false-positives the router uses a reinforcement parameter, called *gossip confidence*, to decide when to accept the *gossip message*. If an attack is detected, the router changes the routing algorithm to the packets that wants to go through the path under attack. Therefore, *Gossip NoC* combines two strategies to protect the MPSoC against SCA: i) Detection, which includes the bandwidth monitoring and the *gossip message* generation in the presence of abnormal behavior; and ii) protection, triggered when any *gossip message* is received. As a security mechanism, the router can change the route of the sensitive path, avoiding infected hops inside the NoC. Prior implementations explored the algorithms XY and YX as the possibilities. However, to avoid deadlock, such solution required virtual channels as presented by (BORHANI; MOVAGHAR; COLE, 2010) (TATAS; SAWA; KYRIACOU, 2014). Thus, such approach increased the area significantly. Then, an optimized version used the adaptive routing algorithm west first, also known as West First Routing Logic (WFRL). However, instead of changing the different route choice according to traffic congestion, the WFRL from Gossip understand a detected attack as a router blockage. Therefore, differently from the work of Sepúlveda, where WFRL uses a router contention to manage the routing possibilities, Gossip NoC monitors the traffic, and if the gossip messages detect an anomaly in some router, this router is treated as a blocked path. This blockage remains until the traffic intensity decreases over the observation time of the monitors.

6.1.1 Architecture

The *gossip router* microarchitecture is shown in Fig. 6.1. It is based on a conventional NoC router, which performs the switching of packets from an input to any output, according to the routing algorithm. Traditional routers integrate four main components: i) input buffers, which store the data at the input ports of the router; ii) routing algorithm, which selects the output port to redirect the incoming data. The route field of the packet header, used to define the output port based on West first routing algorithm; iii) arbitration logic, that grants the utilization of the crossbar switch to one of the input buffers; and iv) crossbar switch, which links the inputs to outputs of the router. Additional to these components, three main blocks are integrated to the *gossip router*:

- Gossip In Block (1): It controls the internal state of the *gossip router* according to the values of the input signals (outputGossip). When the number of *gossip messages* received from neighbor routers overcomes the *gossip confidence*, an attack is confirmed. As a result, the routing of the packets is modified.
- Gossip Logic (2): It commutes the state of the output port, defining as blocked.
- Gossip Generator (3): It monitors the traffic bandwidth. When it exceeds a protection bandwidth threshold, a signal indicating a possible attack is activated and transmitted (inputGossip) to the *Gossip In Block* of all the neighbor routers. This configures a *gossip message*.

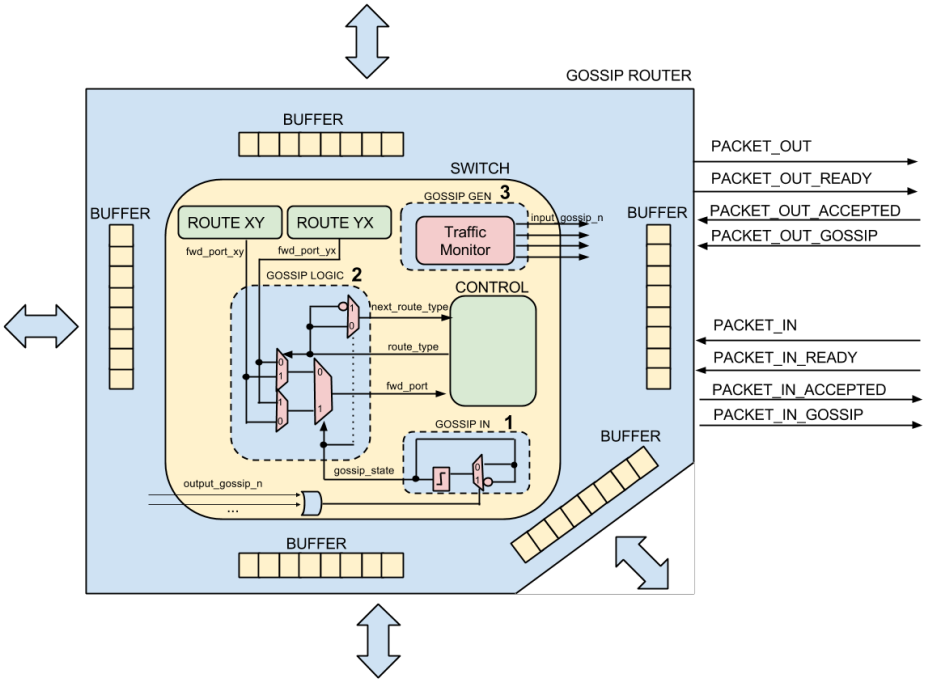
6.1.2 Functionality

Five stages compose the traditional router communication: i) Buffer write and route selection, to store incoming data and quantify the output port; ii) data allocation, to reserve the input buffer at the neighboring router linked to the output port; iii) arbitration, to schedule the transmission of the data; iv) crossbar switch, to commute the data by the crossbar; and v) link traversal, which includes the time required to reach the next input port.

Additionally, to this basic functionality, the *Gossip router* consists of two stages: i) alert messages generation and; ii) security activation.

In the first stage, all routers monitor the usage of the input channels. This event is concurrent to the buffer write and route selection stage of the router functionality. It looks

Figure 6.1: *Gossip router* microarchitecture: (1) Gossip In Block; (2) Gossip Logic; (3) Gossip Generator.

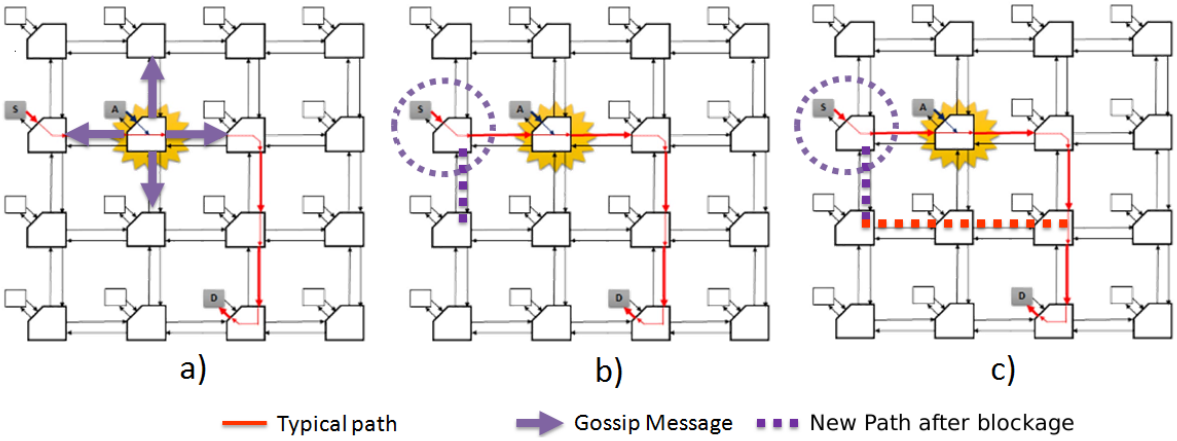


for periods of intense traffic. All routers have the assumption that excessive traffic means anomalies, and thus, possible attack. In that case, the router sends alert messages, called in this architecture as *gossip messages*, to the neighbor routers. Fig. 6.2(a) presents this behavior in the presence of a NoC timing attack (described in Section IV-C), where three actors can be observed: S (source), D (destiny) and A (attacker).

In the second stage, the routers that receive the *gossip messages* evaluate the possibility of activation of the security mechanism. A threshold, called *gossip confidence*, defines how many messages are required to activate the security scheme, that is, the change the routing algorithm. This behavior can be observed in Fig. 6.2(b), where the packets from S deviate from the original path. As a result, the infected IP that was collecting information in such a path is now unable to complete the attack. After the change in the routing algorithm, the sensitive information uses another route. Finally, in the next router, the security mechanism is not activated, which means that the packets return to pass through the router. Fig. 6.2(c) presents this final behavior.

However, attackers may notice the change of routing algorithm. To overcome this drawback, the MPSoC have to limit the number of processing elements that can execute external tasks, and thus the possible attackers. As *Gossip NoC* changes the route during run-time, the attacker needs to create, simultaneously, two zones of congestion (one for each possible path). Thus, requiring several infected IPs to accomplish this goal.

Figure 6.2: *Gossip NoC* functionality: (a) Gossip Messages; (b) Routing changing; (c) Back to normal behavior.



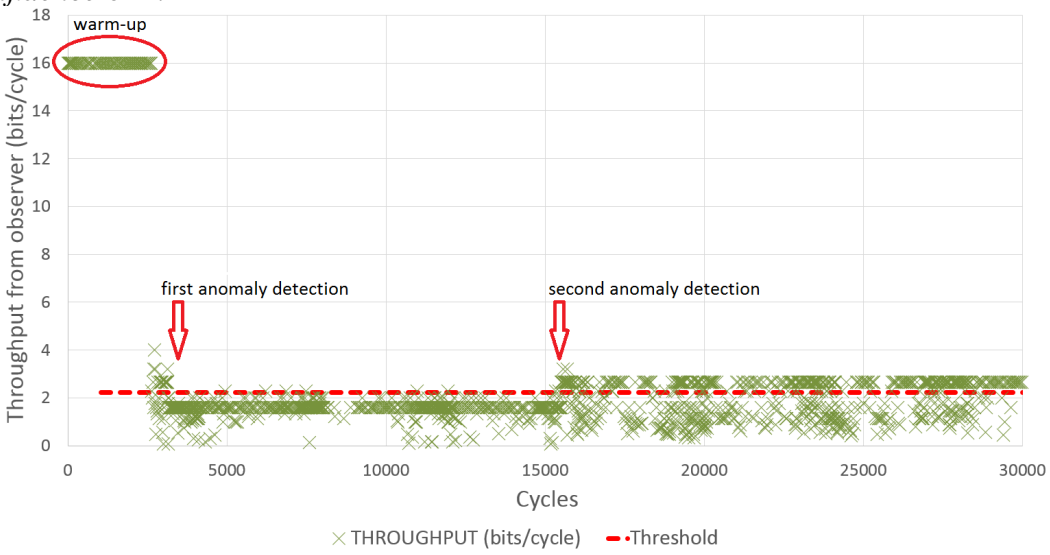
During the design phase, two parameters have to be adjusted, the throughput threshold and the *gossip confidence*. The throughput value triggers a *gossip message* to the neighbor routers. The *gossip confidence* defines the minimum number of *gossip messages* required to activate the countermeasure mechanism (routing algorithm modification).

6.2 Gossip NoC Evaluation

The Gossip NoC evaluation was accomplished through a simulation environment. The experimentation scenario was a distributed timing attack, which had two *injectors* at an injection rate of 30%, the *observer* at an injection rate of 15%, the crypto-processor at 10% and the other IPs in the system with random targets at 10% (communication noise). The simulation performed 50,000 traces, whose efficiency in the non-protected NoC was 100% (all sensitive packets were observed). Then, it was simulated such scenario with the Gossip NoC. The results are presented in Figure 6.3. There is a warm-up phase in the firsts 3000 cycles, during the filling of the buffers. After the warm-up, the throughput falls to 2.5 bps, being a similar behavior from DTA attack without protection. However, after cycle 4000 the response of the throughput changes again, bringing the mean throughput lower than the threshold, marked in Figure 6.3 as the first anomaly detection. Then, after the cycle 15,000, a new behavior occurs, where the mean throughput increases above the threshold, marked as the second anomaly detection. This anomaly indicates that *Gossip NoC* identified high bandwidth usage in the traffic and changed two times the route of

neighbor packets (at cycle 4,000 and cycle 15,000), altering all communications behavior.

Figure 6.3: Trace throughput of the distributed timing attack under *Gossip NoC*. *Gossip confidence* of 1.



The first changing in behavior (at cycle 4,000) created an increase of communication above expected from *observers* router. This difference happened because of the *gossip routers* detected not only the attacker's traffic as anomalies but the regular ones too. So, routers near the *observer* were triggered with *gossip messages* and changed the route. Then, because of this increase, at time 15,000, the *observer* route identified this excessive traffic as another threat, creating new *gossip messages*, and reducing the congestion (increasing the throughput) as a consequence.

In this experiment, what happened with the sensitive packets was that they were rerouted at cycle 6000. This information can be confirmed with the results presented in Table 6.1. It reinforces the fact that these packets changed their routes, not passing in observer router because 91% of the sensitive data was not identified, achieving an effectiveness of only 6.62%. The success rate of the attack decreased to 7% due to the traffic noise inserted by the *gossip router* mistakes, that change the route of conventional packets around, adding false data in observer measurements. So, except this 7

Table 6.1: Effectiveness (% of matches) of DTA using a threshold of 2.23 bps under *Gossip NoC*.

	Effectiveness	**Success Rate**	**False Positives**	**False Negatives**
DTA	6.62%	6.56%	93.43%	93.37%

The reason that ordinary packets were considered a threat by the *Gossip NoC* is that the *gossip confidence* parameter used was too low. It used the value one as the parameter. This value refers the quantity of *gossip messages* required to consider a threat.

So, even high used paths by regular packets could be considered anomalies by the system. To avoid these false alarms, the *gossip confidence* has to be adjusted, increased. Therefore, another experiment with Gossip NoC was developed, increasing this parameter, to understand this reinforcement strategy potential.

The result of the experiment with the *gossip confidence* set to ten (ten *gossip messages* to believe that is a threat) is presented in the Figure 6.4. The throughput trace depicted seems to be the same from a common DTA attack, but the results of effectiveness in Table 6.2 demonstrates that the protection mechanism was useful. The throughput trace did not show changing in behavior because the Gossip identified only the attacked path, and rerouted the sensitive path successfully. However, increase the *gossip confidence* from one to ten allows the attacker to double its effectiveness, from 6% to 13%. This behavior happened because the *Gossip NoC* needed more time to reinforce the fact that there was an attack, given more time to the attacker. Therefore, *gossip confidence* presents a trade-off between the impact on the common behavior of the system and the protection effectiveness.

Figure 6.4: Trace throughput of the DTA under *Gossip NoC*. *Gossip confidence* of 10.

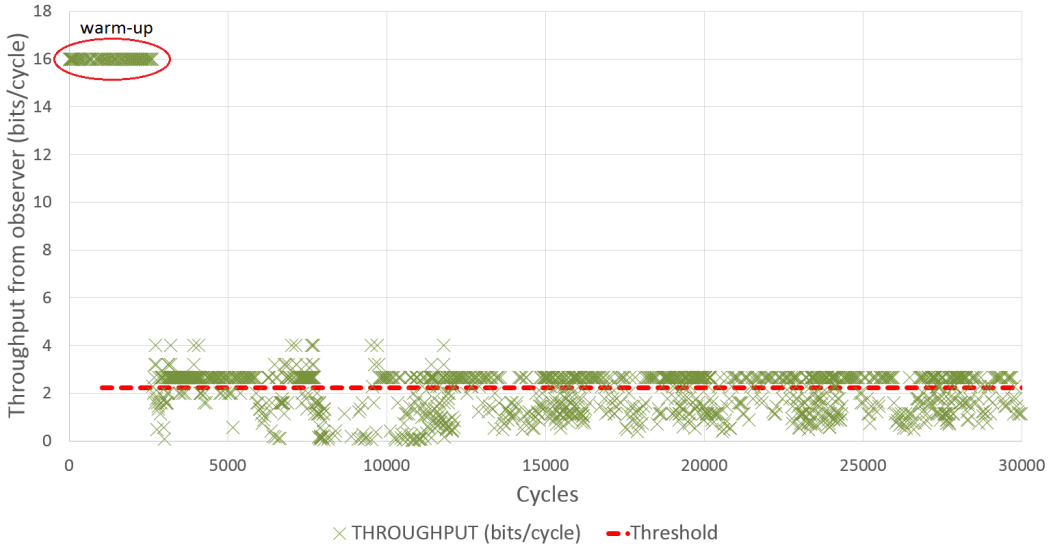


Table 6.2: Effectiveness (% of matches) of DTA using a threshold of 2.23 bps under *Gossip NoC*. *Gossip confidence* of 10.

	Effectiveness	**Success Rate**	**FP**	**FN**
DTA	13.55%	19.06%	80.93%	86.44%

In order to understand this trade-off, Figure 6.5 presents the results of DTA effectiveness when changing the *gossip confidence* parameter from 1 to 100. Simulations executed changing this setting with a step of 1 from 1 until 20, and after that, the step was 10. The last point represents the value where there is no protection because the DTA

achieves its general effectiveness of 59%. It can be observed that the increase of *gossip confidence* decreases the protection linearly. However, as presented in the Figure 6.4, a *gossip confidence* of 10 already has no much impact on communications behavior. Therefore, it should never be a value near 1, because it creates many false-positive detections, but neither a high one, because it decreases the protection level. The proper value has to be a lower value, from 10 to 40, to maintain the attack effectiveness below 30%, which means, to avoid that the attack acquires much data. It is important to emphasize that this result is valid for the experimented congestion threshold (the amount of traffic considered anomaly). A different environment and scenario can affect the decision of the gossip confidence.

Figure 6.5: Effectiveness of DTA for 50000 traces according different *gossip confidence* configurations.

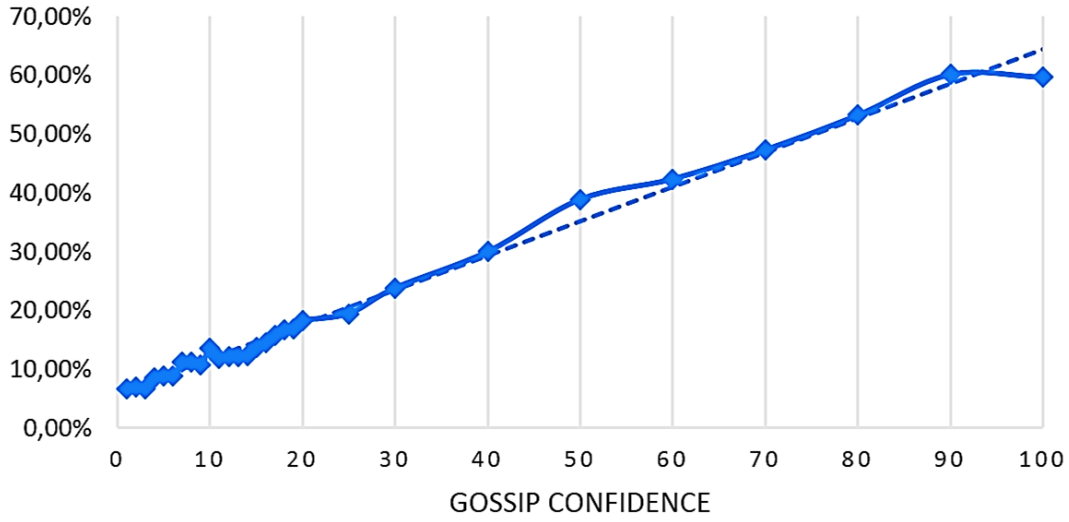


Table 6.3 presents synthesis results of a unprotected router and *Gossip router* under 65 nanometer technology and 1 gigahertz as reference clock frequency. Results show that each gossip router has 21% area and 16% power overhead when compared to the typical router. In spite of the high percentage, the interconnection network represents about 35% of the whole MPSoC system, being the Gossip NoC an increase of 7.3% in area and 5.6% in power regarding the whole chip design.

Table 6.3: Synthesis results for the Unprotected and Gossip routers for 65 nm ASIC technology.

	Unprotected NoC	Gossip NoC	% Overhead
Area (um2)	2632	3189	21.16%
Power (mW)	2.073	2.409	16.2%

6.3 Considerations

The present chapter has shown the Gossip NoC, an enhanced secure NoC. The primary objective of this proposal was to provide protection against NoC timing attacks with a small area overhead. The strategy of generating alert messages can disturb the system communication if any false-positive is identified as potential attacks. Then, the Gossip Confidence policy complemented the architecture. It has allowed the avoidance of such undesired situation. Experimented results on this metric had shown that this approach was capable of avoiding false-positives. However, a high Gossip Confidence could let the system vulnerable. Therefore, experiments concluded that setting this parameter to ten (ten messages to accept as a potential attack) are a good default value, balancing costs and effectiveness.

7 EXPERIMENTAL STUDY

This chapter presents the experiments to evaluate Architectural Channel Attacks and the protection mechanisms in an MPSoC environment. As a result, two main objectives of the thesis are obtained: i) the practical demonstration that MPSoCs are vulnerable through the shared resources, such as cache and NoC; and ii) the analysis of the NoC as a candidate to provide system security.

Experiments took into account attacks and countermeasures found in the state-of-the-art and the ones proposed in this thesis. The evaluation of the attacks was performed separately according to the attacking nature: i) timing-based; ii) access-based, and iii) collision-based. Then, for each one, four protection techniques were analyzed: i) random loops; ii) cache partition; iii) Gossip NoC; and iv) adaptive routing. Two countermeasures for caches and two for NoCs comprises the tests on protection.

All experiments considered a moderate noise scenario, which means a weak activity in communication and cache access from not associated IPs. The objective to execute the tests in such scenario was to observe the best results from each attack. In a high noise situation, the NoC timing attack would decrease the quality of results. It could work on the issues discussed in Chapter 5 do not compromise the technique.

A real MPSoC environment running in an FPGA were employed. The technology was from Altera (Intel FPGA), the Cyclone IV FPGA. The reference hardware platform was the MPSoC Glass (described in Chapter 2). MPSoC Glass organization is presented again in this chapter in Fig. 7.1. During the tests, the system was connected to a host computer through UART protocol (using USB adapter). The external interface did not affect any experiment because only the data after processing was sent to the host. The final section of this chapter presents synthesis results of the MPSoC Glass in the FPGA.

7.1 Experimental Setup

The setup of all experiments on the target hardware platform MPSoC Glass (Fig. 7.1) is detailed in the table 7.1. However, Arrow attack was an exception, and it required a different setup, which is presented in table 7.2. The main difference from Arrow setup was the cache configuration. The threat model from Arrow requires that no T table shares the same set in the cache. As a result, the shared cache in Arrow experiment was bigger than in other experiments.

Figure 7.1: Reference Architecture - MPSoC Glass. Four fast NIOS II core, ten economy NIOS II core, one UART interface, and one shared cache memory.

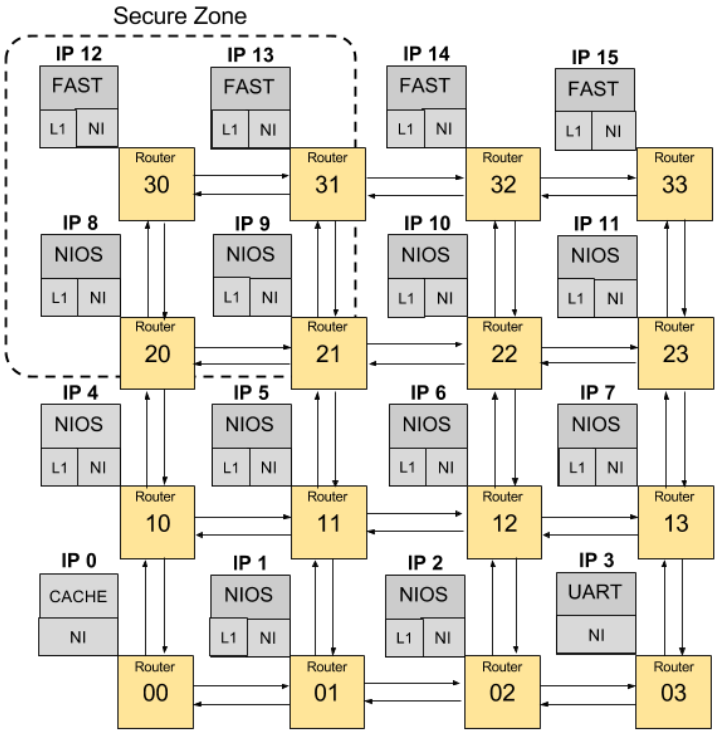


Table 7.1: Setup of the experiments on the following attacks: Bernstein, Hourglass, Prime+Probe, Firecracker, Bogdanov and Earthquake.

	Description
Source of Sensitive Traffic	Shared Cache - IP 0
Destination of Sensitive Traffic	Crypto-Processor - IP 13
Infected IP	Common Processor - IP 1
L1 Configuration	32kB Direct Mapped Line Size of 4 Bytes
L2 Configuartion	64kB 16-way Set Associative Line Size of 16 Bytes
NoC Configuration	4x4 Mesh 5-port router XY Routing Algorithm Flit Size of 4 Bytes

A direct-mapped cache was used as L1, to reduce the noise during tests. Different cache configurations affect the noise of the experiments, but in the same manner for all observed attacks.

7.2 Timing-based Attack Experiments

This section presents the measures of two timing-based attacks, the state-of-the-art Bernstein attack, and the proposed Hourglass. Firstly, these attacks were evaluated in a

Table 7.2: Setup of the experiments on Arrow attack.

	Description
Source of Sensitive Traffic	Shared Cache - IP 0
Destination of Sensitive Traffic	Crypto-Processor - IP 13
Infected IP	Common Processor - IP 1
L1 Configuration	32kB Direct Mapped Line Size of 4 Bytes
L2 Configuartion	256kB 16-way Set Associative Line Size of 16 Bytes
NoC Configuration	4x4 Mesh 5-port router XY Routing Algorithm Flit Size of 4 Bytes

non-protected environment. Then, four scenarios with different countermeasure strategies were analyzed under the attacks.

7.2.1 Bernstein Adaptation

Adaptations were necessary for Bernstein technique to attack an MPSoC. Originally, the Bernstein attack was developed to be operated remotely. However, in our proposed MPSoC scenario, the attacker runs inside the target chip. Then, the attacker may access the crypto-functions more directly. Another difference regards the communication delay, which inside the MPSoC became irrelevant when compared to a cache miss latency.

Another issue refers to the learning step. This stage in Bernstein is made in another machine, similar to the target one. However, in our scenario, the attacker can develop the learning phase inside the target system, and sometimes use the same shared cache.

The third issue regards to the computational effort required for the attack. According to Bernstein, after sampling the time of encryption, some calculations must be performed. One refers to the average, and another to the variance of the samples. All collected data was pre-processed and stored in arrays to avoid an excessive computational effort inside the infected IP. During an attack, from time to time, the malware transfers to an external agent the data to be computed in a further process. The adapted C code of the attack, showing the pre-processing and the array storage, is presented in the code 7.1.

```
1    generate_plaintext(plaintext);
2    time1=alt_timestamp();
3    send_plaintext(plaintext);
4    wait_ciphertext();
```

```
5     time2 = alt_timestamp();
6     delta = (time2 - time1);
7     time_enc[plaintext[i]] += delta;
8     var_enc[plaintext[i]] += (delta * delta);
9     time_enc_total += delta;
10    qty_enc[plaintext[i]]++;
11    qty_enc_total++;
```

Code 7.1 – Adapted C code of the Timing Attack.

Four arrays stores all needed information to build in the post-process the inputs for the correlation step. The $time\_enc$ accumulates the encryption times that together with qty_enc (counter) can calculate the averages for each byte case of the plaintext. Each byte case of the plaintext is given by the variable i. The var_enc accumulates the square of the encryption time aiming to calculate the variance in a further process. These calculations occur in the learning step, and in during the attack execution. The only difference of this code 7.1 from the Hourglass attack, is that instead await the end of the encryption (line 4), the NoC timing attack is performed.

Therefore, applying the attack inside the MPSoC required modifications of the original approach of Bernstein. Some parts became simpler, such as the communication between processors, but the generation of the statistical information had to be outsourced.

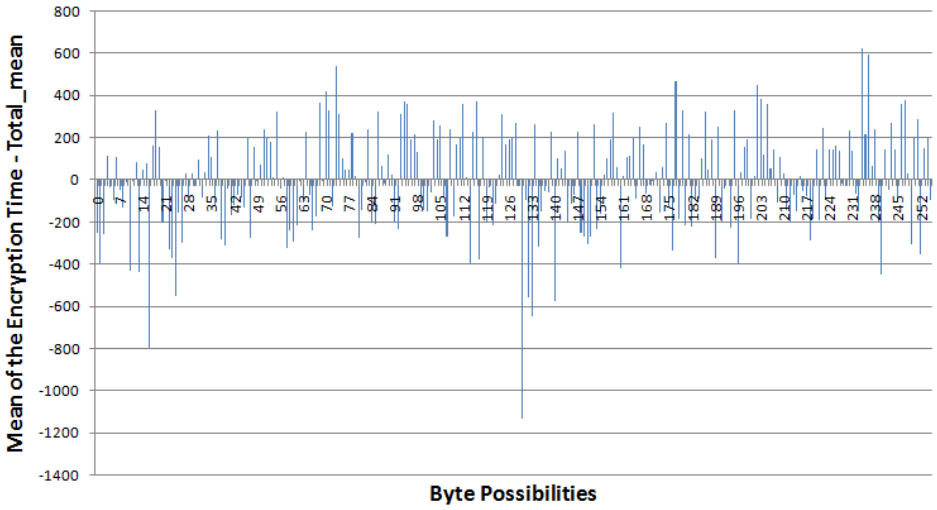
7.2.2 Attacks Evaluation

The learning phase used 10000 encryptions to create the signatures of each plaintext byte. Fig. 7.2 presents the signature obtained for the first byte with a known key. The signature is obtained calculating the variance of the mean time to encrypt the bytes with a fixed known key. The average of the variance from all possibilities is used as a reference to normalize the results, explaining why some results are negative.

After generating the signature for each possible byte, the attack was performed. Several different experiments were executed, from 10,000 samples until 10,000,000 samples. Experimental work shows that less than 10,000 samples do not provide the statistic properties required for the correlation.

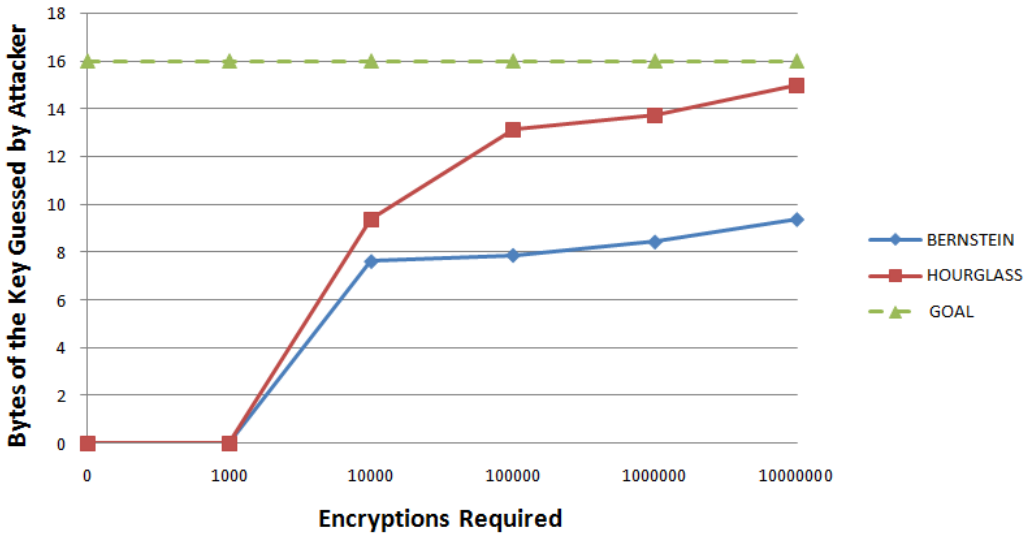
The Hourglass was executed in the same manner as the Bernstein's attack. The main difference was that the timing collected in Hourglass referred only to the first round. Since the timing behavior of the whole encryption and the first round is completely dif-

Figure 7.2: Learning phase - Byte 0 signature.



ferent, a new training phase for Hourglass was required. It was used 10,000 samples with a known key to obtain the signature of all possible bytes of the plaintext. Then, the attack was performed in the same range of samples used in Bernstein's experiments. Results of Bernstein and Hourglass are presented in Fig. 7.3, showing the number of bytes revealed per encryptions.

Figure 7.3: Bernstein's vs Hourglass attacks.



The Hourglass was able to discover more bytes faster than Bernstein. This better efficiency is explained by the fact that annotating the time earlier fewer cache misses (generated by the other AES rounds) disturb the analysis.

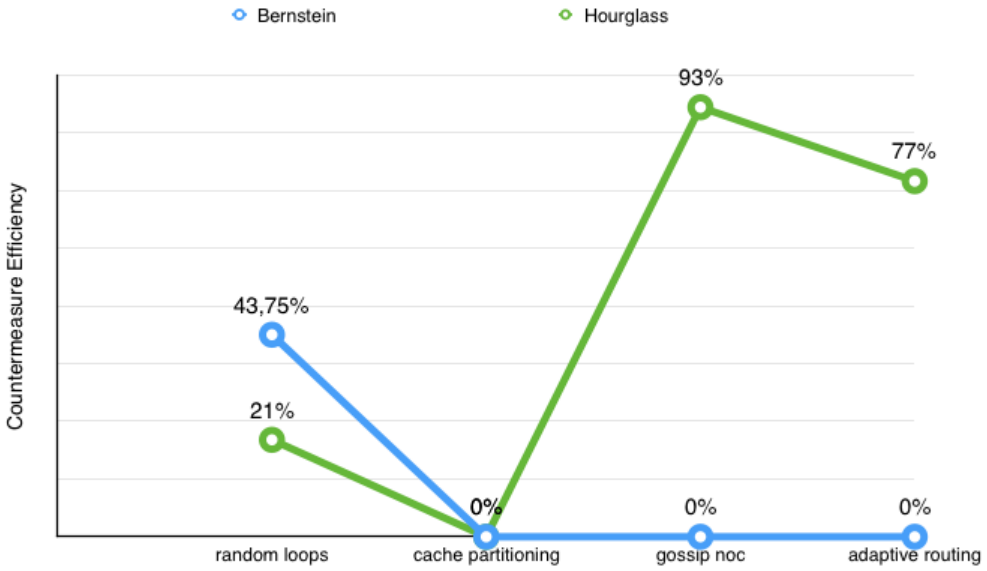
7.2.3 Countermeasures Evaluation

Four techniques were tested under Bernstein and Hourglass attack: i) random loops; ii) cache partitioning; iii) Gossip NoC; and iv) Adaptive Routing. To compare the results, a metric was proposed, defined as *countermeasure efficiency*. This efficiency metric is given by the following equation 7.1:

$$1 - \frac{reveiled\_key\_bytes_{protected-system}}{reveiled\_key\_bytes_{unprotected-system}} \tag{7.1}$$

The experiments, each one with a different countermeasure, tested both attacks, totalizing eight scenarios. The methodology followed the same one applied in the attacks evaluations. Then, the results of the eight experiments were converted to the efficiency value. Figure 7.4 shows the countermeasure efficiency of each case.

Figure 7.4: Countermeasures Evaluation under timing-based attacks. Two cache- and two NoC- protections techniques were evaluated.



Results show that the random loops are very useful against Bernstein because it alters the total timing behavior. The effect of the cache partitioning was not so efficient as expected because inside the MPSoC environment; the shared cache interferences were not so intense. A dedicated memory space for AES allowed executing the encryption with an explicit timing behavior. Inside the MPSoC environment, the learning step was performed with small communication noise, so the behavior of a cache partitioning does not affect the attack.

The NoC protection mechanisms were not sufficient against Bernstein. The main

reason was that any possible delay inserted by them would not be in the same magnitude of a cache miss latency. Then, they did not affect the total attack time. However, if a NoC countermeasure explores the application time behavior instead communication time behavior, it could avoid such kind of attack. For example, a NoC that tries to normalize the time of response of some IPs, like inserting random delays for caches that respond to hit behavior soon). Consequently, this strategy could avoid timing attacks like Bernstein, but decreasing overall performance, like random loops.

On the other hand, the NoC countermeasures had high-efficiency results against Hourglass. Since the NoC countermeasures deviate several packets from the path attacked, the sampling information became very erroneous. It was presented in the last section, that it was required 10,000 clean samples to achieve some result. Therefore, to achieve the same result of the non-protected, about 1,000,000 samples were needed. The difference of efficiency between Gossip NoC and adaptive routing is in the fact that Gossip NoC re-routed all the packets after the anomaly identification, while the adaptive routing only changed the route when congested.

The technique of random loops also created some difficulties for Hourglass. In this case, the software included timing noise in the first round behavior. However, the amount of delay inserted during the first round was not enough to avoid correlation efficiency. The cache partitioning strategy did not work well with Hourglass for the same reason that it did not work with Bernstein.

7.3 Access-based Attack Experiments

The attacks that explore the access in the cache by a cryptographic algorithm are known as access-based attacks. Three attacks were evaluated, the Prime+Probe by Osvik using the Xinjie's optimization (XINJIE et al., 2008), and the two proposed in this thesis, Firecracker, and Arrow. After the analysis of these attacks in the MPSoC, four protection techniques were tested for each attack. Next subsections present the experimental setup and the experiment details.

7.3.1 Attacks Evaluation

During tests, the three access-based attacks (Prime+Probe, Firecracker, and Arrow) generated 1000 random plaintexts. While Prime+Probe attack probed the cache after each encryption, Firecracker and Arrow probed the cache according to a trigger generated by the NoC timing attack.

The collected access behavior from probe step was stored in a variable of the malware. The size of this variable was the number of sets in the cache used to store the T tables. Based on the setup of these experiments, the size was 64, because each cache line could save four words. So, a 64-bit variable was used ($longint$). After each execution and probe had performed, the attacker sent the plaintext used and this variable to the external attacker, a host computer. Another possibility would be to send all the data after the 1000 encryptions. However, it would require storing the data in an array of size 1000 instead. The host of Prime+Probe and Firecracker computed the same algorithm to receive and recover all possible bits of the key. It was implemented in Python, and the reception and analysis of the first byte are presented in the code 7.2.

```
1   import math
2   import serial
3   print ("COLLECTING RESULTS")
4   ser = serial.Serial('COM5',115200)
5   ser.flushInput()
6   ser.flushOutput()
7   while (j < ATTACK_SAMPLES):
8       while ser.inWaiting() > 0:
9           data_raw = ser.read(1)
10          if (i < 16):
11              plaintext += (ord(data_raw) << (i*8))
12          else:
13              channel += (ord(data_raw) << ((i-16)*8))
14  print ("STARTING ANALYSIS")
15  # SEPARATING plaintext VALUES IN BYTES OF P
16  for x in range(0, 16):
17      P.append((plaintext >> (x*8)) & 0xFF)
18  # ANALYSIS OF THE FIRST BYTE OF THE KEY
19  for x in range(0, 64):
```

```
20          if (((channel >> x) & 1) == 0):
21              not_K = (x*4) ^ P[3]
22              if not_K in V_0:
23                  V_0.remove(not_K)
24              not_K = ((x*4)+1) ^ P[3]
25              if not_K in V_0:
26                  V_0.remove(not_K)
27              not_K = ((x*4)+2) ^ P[3]
28              if not_K in V_0:
29                  V_0.remove(not_K)
30              not_K = ((x*4)+3) ^ P[3]
31              if not_K in V_0:
32                  V_0.remove(not_K)
```

Code 7.2 – Data reception and first key byte analysis of Prime+Probe/Firecracker code in Python.

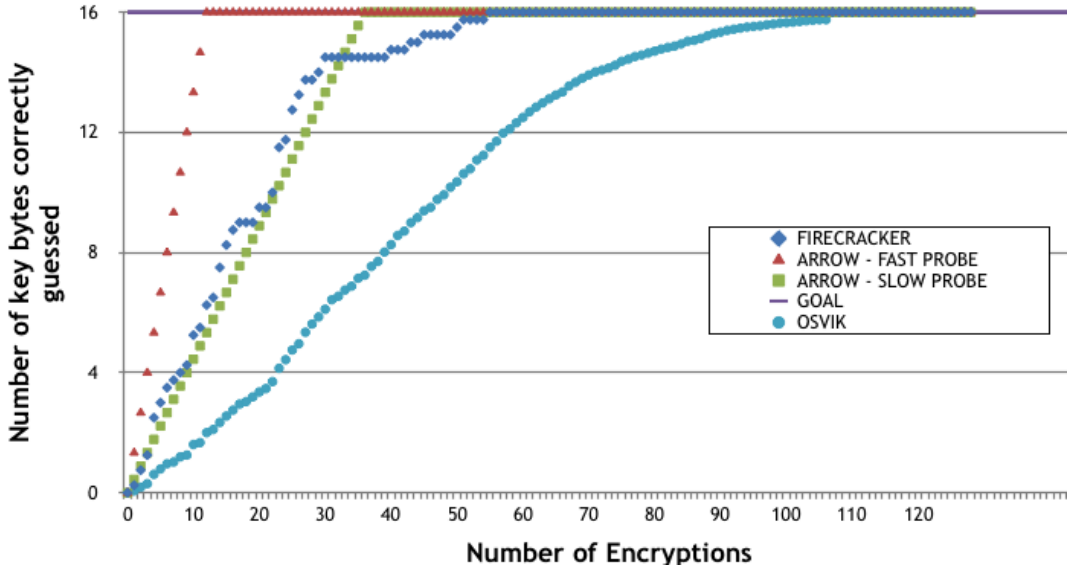
The first part of the code presents the acquisitions of the data, where the first bytes refers to the plaintext used, and the last ones the information about the sets (64 bit variable $channel$). Then, the analysis tested each access information, where the non-used sets were converted in the key byte candidates (using the Xor operation). Then, the attacker removed the result from the key possibilities vector (V_0).

Arrow attack presents a different methodology during probe and analysis step. For each encryption, the probe of Arrow attack generated four variables instead one, each one depicting the used sets for each access of a target T table. Then, these four variables were sent to the host. The host analyzed the accessed sets for each operation in the first round, revealing key byte candidates. After this, the attack was performed again with other target T table. In total, this attack executed four times before doing the exhaustive key search.

Firecracker improved up to 60% the attack efficiency compared to Prime+Probe, as shown in fig. 7.5. The NoC timing attack identified the 16th access, representing the end of the first round of AES algorithm. Probing the cache earlier allowed the attacker to analyze only the accesses from the first round instead all rounds as Prime+Probe. Therefore, it was possible to recover the bytes of the key with fewer encryptions. Fig. 7.5 presents the discovering of the bytes among the encryptions requested for the attacks, Osvik, Firecracker, and Arrow fast and slow.

The Arrow achieved the best result from all tested access-based attacks. In a scenario where high precision NoC timing attack was possible, the attack could perform accurate probes. For Arrow attack, it was evaluated two different scenarios. One sce-

Figure 7.5: Osvik (Prime+Probe), Firecracker, and Arrow attack results. Arrow was performed in a fast and a slow scenario.



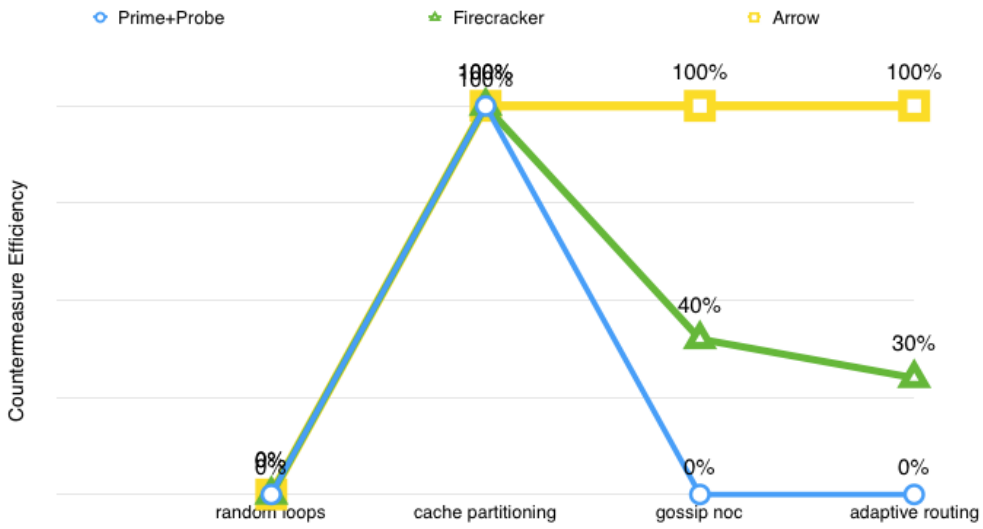
nario, defined as fast Arrow in fig. 7.5, the attack could probe all the 64 sets before the next T table access. The second scenario, defined as slow Arrow, used sixteen cache hits as the time to happen the next access from AES to the target T table. Hence, in the slow scenario, the attack required to remake the encryption with the same plaintext four times in the worst case ($4 * 16 = 64$ sets). In fact, the worst case rarely happened, because when any cache miss was identified, the attack finished the probe step. The average of encryptions repeated was about three times. If the probe step takes more time, like four times more, the technique becomes similar to Firecracker. There is a possibility to get even worst if AES cryptography has some optimization and performs faster operations. The Firecracker obtained good results as well, but required more encryptions than Arrow, since it analyses all access of the first round. Besides, it is possible that some access of the second round inserted some noise in the detection since high accurate NoC was not a condition in this case. The traditional Prime+Probe attack could obtain success in the attack, but with twice encryptions of Firecracker and four times of fast Arrow.

The final step did the exhaustive key search with the candidates. Since each cache line could store four words (2^2 possibilities), the exhaustive key search step required $2^{(16*2)}$ operations for all access-based attacks.

7.3.2 Countermeasures Evaluation

The access-based attacks were tested under the same four protection mechanisms: i) random loops; ii) cache partitioning; iii) Gossip NoC; iv) adaptive routing. The same efficiency metric from timing-based countermeasure experiments was used. This parameter, called protection efficiency, uses the portion of revealed key bytes in the protected system over the part in the unprotected system. Results of these experiments are presented in figure 7.6.

Figure 7.6: Countermeasures Evaluation under access-based attacks. Two cache- and two NoC- protections techniques were evaluated.



The first observation of the countermeasures is that random loops did not take any effect in the access-based attacks. The reason is that a different timing behavior does not affect the attack, that looks only for the access to the cache. On the other hand, the cache partitioning was able to avoid all the attacks. Since cache partitioning prevents any other process to access the same cache location, it was not possible to observe the accessed sets by the malware process.

The NoC countermeasures did not take any effect against Prime+Probe since the attack awaits the end of the encryptions to probe the shared cache. Hence, only the attacks that use the NoC timing attack obtained difficulties with these countermeasures. Arrow attack was not able to accomplish the attack since the high accuracy required for the attack was lost. Any error in the NoC timing attack makes Arrow unpractical. Firecracker was penalized by Gossip NoC and adaptive routing behaviors, with an efficiency of 40% and 30% respectively. Firecracker uses the NoC timing attack to probe the cache without other

rounds accesses. These countermeasures forced the NoC timing attack of Firecracker to trigger the probe after several rounds. As a result, the analysis obtained more non-desired accesses, but it did not make the attack unpractical.

7.4 Collision-based Attack Experiments

The last ACA type evaluated in this chapter is the collision-based. The attack from Bogdanov (BOGDANOV et al., 2010) and the proposed Earthquake were analyzed, and then, tested under four protection mechanisms. Before the evaluations, an analysis on an important aspect of the differential collision cache attack is presented.

7.4.1 Analysis on the Differential Collision Cache Attack

The differential collision timing attack aims to explore a particular condition called wide collision. This condition allows the attacker to exploit a difference of five cache hits when at least one collision occurs in the second round for the target pair of plaintexts. However, one issue can affect the practicability of the attack: the size of the cache line.

The size of the cache line affects the detection stage of the attack. A bigger line can store more than one word increasing the probability of cache hits during the encryption. As a result, the objective of the attack is compromised, since more cache hits allow false-positives candidates to present a lower latency, which is called in the experiments as bad positive. Spreitzer and Plos show an equation that calculates the probability of the presence of the following data in the cache (SPREITZER; PLOS, 2013). The same equation is presented in equation 7.2. It focuses on the accesses in the same T table, during all nine rounds of AES (the tenth round uses a different T-table), each round accessing four times. Then, if we consider the encryption of the first plaintext, it is possible to analyze the probability to face cache misses at the beginning of the second plaintext encryption.

$$P(misses) \leftarrow \left(1 - \frac{words\_per\_line}{256}\right)^{(accesses*rounds)} \tag{7.2}$$

Cache line sizes of four, eight, and sixteen words have the chance of new cache misses of 56.7%, 31.88%, and 9.79% respectively. Therefore, depending on the target hardware platform, this condition could avoid the feasibility of the attack. For example, sixteen words line has only 10% of chance that data is not already in the cache. As a con-

sequence, the cache hits are higher in the second plaintext encryption, making unpractical to distinguish the cache hits provoked by the wide collision. Even if the attacker accepts a wide range of samples as candidates, the key search exploration would be unfeasible also, because the attack searches over all possible candidate combinations. For the following experiments, the four-word size was chosen to enable a precise analysis of the attacks. However, this issue is important when applying these techniques.

7.4.2 Attacks Evaluation

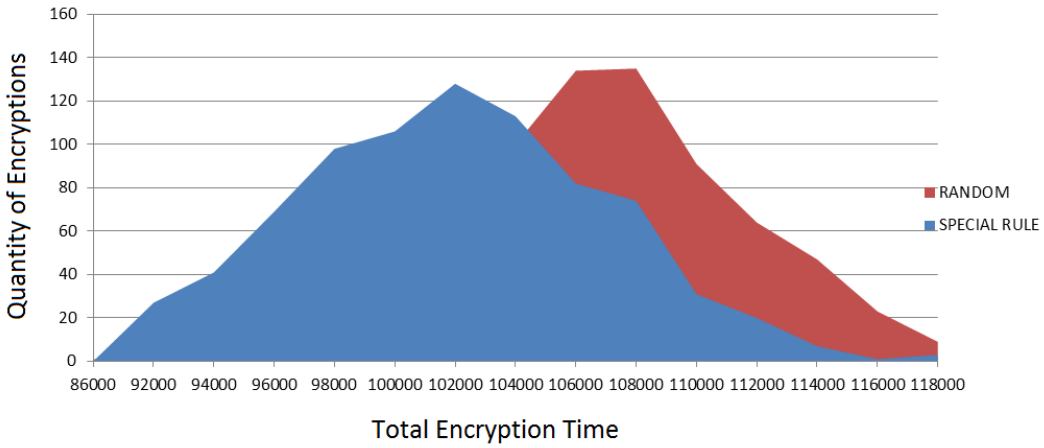
The differential collision cache attack of Bogdanov and the proposed Earthquake attack were evaluated in the target MPSoC Glass platform. Before performing any attack, it was developed the algorithm to generate the plaintext pairs. The generation has to follow the rule described by Bogdanov, also presented in chapter 3, that is the key to provoking the wide-collision situation. This rule defines that both plaintexts must have different values in the same position of the target diagonal, and equal values in the same position outside the target diagonal. Below, it is presented a representation of both plaintexts generated according to this rule, where the elements in gray represent different values.

$$P_1 = \begin{bmatrix} a_0 & x_1 & x_2 & x_3 \\ x_4 & a_5 & x_6 & x_7 \\ x_8 & x_9 & a_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & a_{15} \end{bmatrix} \qquad P_2 = \begin{bmatrix} e_0 & x_1 & x_2 & x_3 \\ x_4 & e_5 & x_6 & x_7 \\ x_8 & x_9 & e_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & e_{15} \end{bmatrix}$$

Then, a particular experiment tested the algorithm producing the pairs of plaintext. The first case encrypted 10,000 random pairs of plaintext. The process asked two encryptions in a row and annotated only the time to encrypt the second one. All timing responses were saved, creating a histogram of the frequency of each execution time. Then, the same was performed but using the plaintext pair generator algorithm. In the same manner, the second case saved all timing information of the second one. A further process elaborated its histogram. If the generator algorithm was correct, the average encryption time should be lower, since this generation rule forces more cache collision than usual. The result of this experiment is presented in the figure 7.7, where the expected behavior is confirmed.

Therefore, after confirming the correctness of the generator algorithm, both attacks could be evaluated.

Figure 7.7: Histogram of the encryption times of pairs of plaintexts generated by a random and Bogdanov's rule strategy.



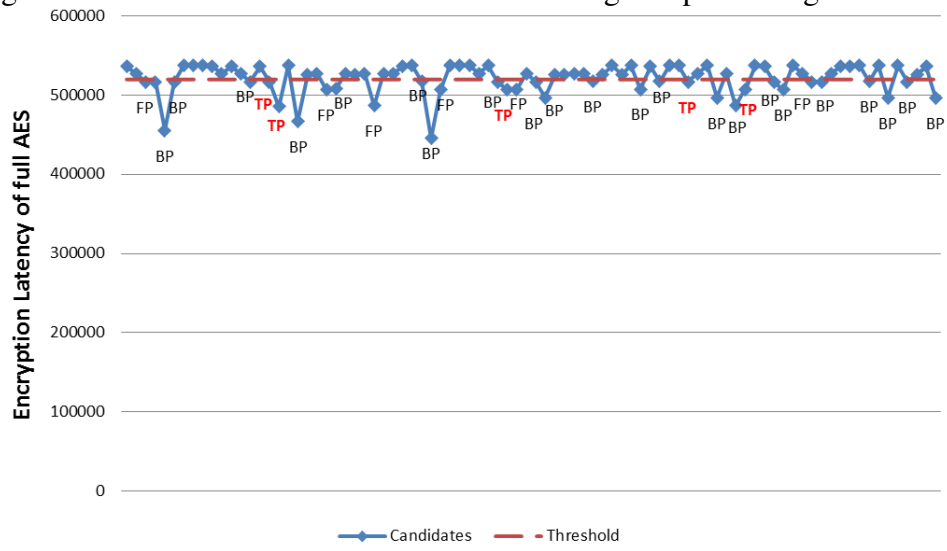
7.4.3 Bogdanov Evaluation

Before starting the attack, it was defined two parameters responsible for improving the searching of wide-collisions. It was used $n = 1000$, which is the number of different values in the target diagonal; and $I = 800$, which is the number of changes in the pair-wise equal elements. It was chosen these values because, in Bogdanov's work, these two presented the best results (BOGDANOV et al., 2010).

The time to process the online stage in the MPSoC running on an FPGA was about four hours. During the storage of the first samples, a relaxed threshold was applied. It resulted in eighty-six samples (from 1000) to be analyzed by the detection stage. Figure 7.8 shows the online stage output. For analyses purposes, marks in the figure present the status of each sample. The possibilities were true positive (TP), bad positive (BP), and false positive (FP). The true positives were the wide collisions that led to the correct subkey. The bad positives were the candidates that obtained cache hits due to the access of the same cache line. The false positives are the candidates without wide collision, but in some way got a low encryption time to be captured. The Bogdanov attack requires that at least four true positives (TP) samples are obtained in the online phase. To accomplish this condition, the minimum threshold accepted at detection stage was 510000.

After the detection stage, the attack resulted in thirty-four candidates to be analyzed, where five were true positives. Thirty-four candidates represents $\binom{4+34}{4} \rightarrow 46376$ groups to be checked. The key recovery stage looks for each group a subkey candidate that leads to a collision in some position, meaning a search of $46376 * 2^{32}$. The final complexity of this experiment to find the key was $2^{47.5}$. The code 7.3 presents the code

Figure 7.8: Detection candidates of the online stage output of Bogdanov attack.



with a summary of the key search stage.

```
    final_num_pos = combinations(possiblex, possibley, possiblez
       , possiblew, limit);
    printf("%u CANDIDATES TO BE ANALYSED\n",final_num_pos)
    i=0;
    while (i <= 4294967295) //0x00000000 -> 0xFFFFFFFF
    {
        k0  = (i >> 24);
        k5  = (i >> 16) & 255;
        k10 = (i >> 8) & 255;
        k15 = i & 255;

        permut = final_num_pos;
        while (permut > 0) {
            a = Te0[(diag_pl1_a0[possiblex[permut]] ^ k0)] ^ Te1
               [(diag_pl1_a1[possiblex[permut]] ^ k5)] ^ Te2[(
               diag_pl1_a2[possiblex[permut]] ^ k10)] ^ Te3[(
               diag_pl1_a3[possiblex[permut]] ^ k15)];
            e = Te0[(diag_pl2_e0[possiblex[permut]] ^ k0)] ^ Te1
               [(diag_pl2_e1[possiblex[permut]] ^ k5)] ^ Te2[(
               diag_pl2_e2[possiblex[permut]] ^ k10)] ^ Te3[(
               diag_pl2_e3[possiblex[permut]] ^ k15)];

            a0 = (a >> 24);
```

```
17              a1 = (a >> 16) & 255;
18              a2 = (a >> 8) & 255;
19              a3 = a & 255;
20              e0 = (e >> 24);
21              e1 = (e >> 16) & 255;
22              e2 = (e >> 8) & 255;
23              e3 = e & 255;
24
25              if ( (a0 == e0) || (a1 == e1) || (a2 == e2) || (a3
                    == e3) )
26              {
27                  #IDENTIFIED A COLLISION
28                  #TEST OTHER SAMPLES -> Y,Z,W
29              }
30          }
```

Code 7.3 – Key search stage code in C.

At the beginning of the code, there is a call to the function $combinations$, responsible for generating all possible combinations (46376 in this case). It organizes the indexes of four arrays, representing the combinations in groups of four. Then, the code starts a loop to check all possible subkey candidate, which represents 2^{32} tests, for each possible combination. At each iteration, the key candidates of the target diagonal are derived from the iterator variable, in this example generating the k_0, k_5, k_{10} and k_{15} subkey candidates. Then, the part of the first round related to the target diagonal is executed, also using the T tables algorithm. If any collision is identified between both plaintext, it is tested the other samples of the current combination. When four collisions are found (at least one per sample in the group), the subkey is included in a list. After this step, all the process repeats for the others three diagonals. The final step test all combinations between the candidates in the final list, retrieving the whole key.

7.4.4 Earthquake Evaluation

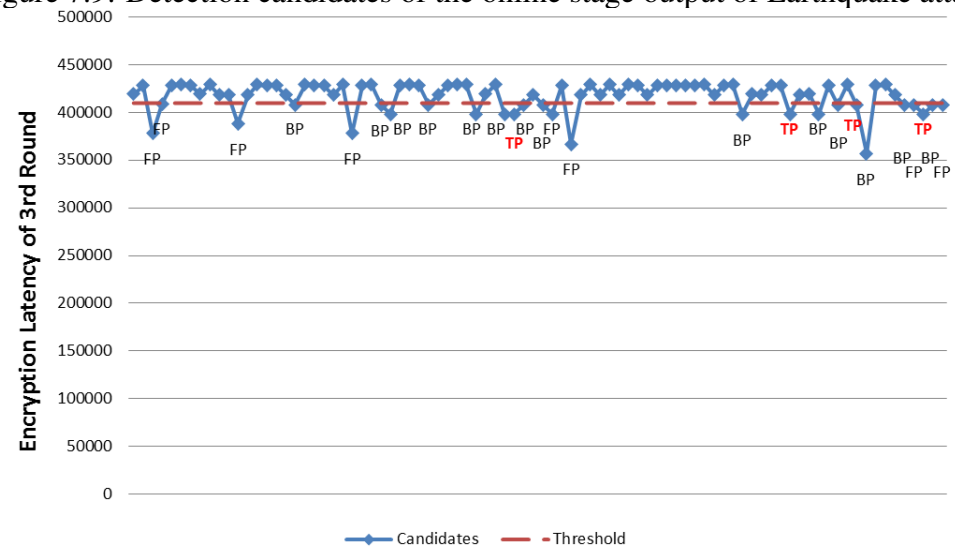
In the same manner as Bogdanov experiments, Earthquake used $n = 1000$, and $I = 800$. In the same manner as applied in Bogdanov experiment, a relaxed threshold allowed to collect fewer samples. The execution of the attack resulted in eighty-six sam-

ples to be analyzed by the detection stage. Figure 7.9 shows the execution output, which comprehends the steps:

- Plaintext Pair Generation
- Pairwise Elements Shuffle
- Prime the cache
- Ask encryptions
- NoC timing attack
- Timing sampling (filtered by threshold)

The resulted samples in figure 7.9 include the marks of true positive (TP), bad positive (BP) and false positives(FP).

Figure 7.9: Detection candidates of the online stage output of Earthquake attack.

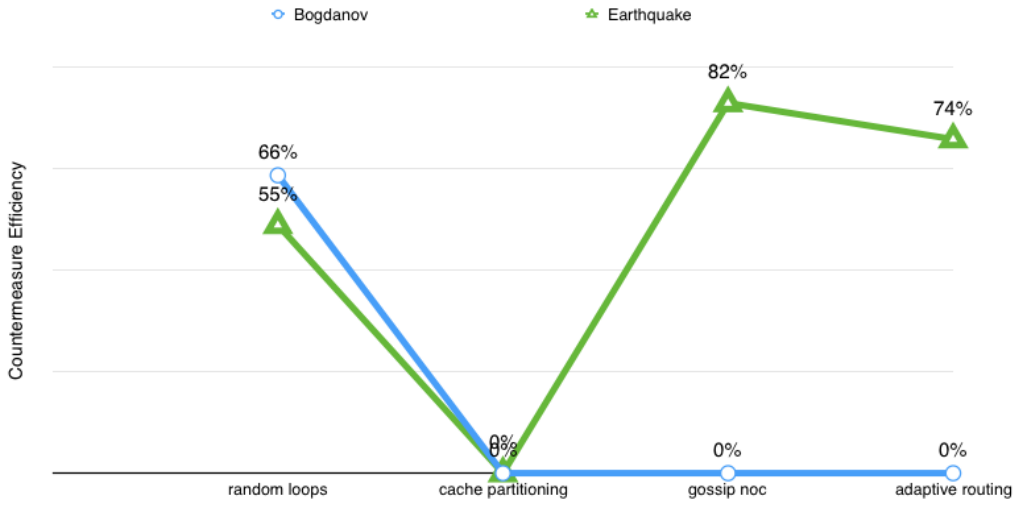


Earthquake used a threshold of 410,000 to accomplish the condition to collect at least four true positives (TP). This metric resulted in twenty-six candidates to be analyzed, where four were true positives. Twenty-six candidates represents $\binom{4+26}{4} \rightarrow 14950$ groups to be checked. Hence, the attacker had to search $14950 * 2^{32}$ possibilities. The final complexity of Earthquake was $2^{(45.86)}$. The same algorithm for the key search stage, presented in code 7.3, was used for the earthquake. Comparing to Bogdanov attack, Earthquake was able to perform the attack four times faster, due to fewer samples after detection stage.

7.4.5 Countermeasures Evaluation

Four countermeasures implemented in the target MPSoC were evaluated against Bogdanov and Earthquake attack. The same metric from the previous countermeasures experiments was used. Results showing the protection efficiency are presented in figure 7.10.

Figure 7.10: Countermeasures Evaluation under collision-based attacks. Two cache- and two NoC- protections techniques were evaluated.



The random loops countermeasure increased the challenge to attack the MPSoC for both attacks significantly. The main reason was that this attack expect few cache miss latencies in the process, and the behavior of the loops inserted a high amount of it. Then, the wide collision situations were masked most of the time. As a result, it was necessary to relax all the thresholds and work with more samples. The other countermeasures did not take any effect against Bogdanov attack.

The Earthquake was also immune against cache partitioning, but the NoC countermeasures impaired it severely. This attack is very sensitive to changes in the latency collected, then, any error in the NoC timing attack makes the attack to get the behavior of different rounds.

Considering the detection stage, both Bogdanov and Earthquake only lost efficiency in the attack, meaning more samples to be analyzed. However, the key search stage from both attacks can regularly become unpractical in such condition.

7.5 Considerations

Three different attack strategies were evaluated and analyzed under a real MPSoC scenario. Attacks from the state-of-the-art and the proposed in this thesis were compared and also tested against four countermeasures.

One important consideration about the timing-based attacks experiments was that the time to perform the attacks was considerable, taking several hours. In some cases of Bernstein, it took days. Thus, the hardware required for the processing elements in an MPSoC to execute such attack is much more complicated than the hardware required for access-based attacks for example. Since a great portion of the timing measurements and calculations have to be stored and manipulated inside the attacker IP, the processing element needs to support at least 64 bits variables organized in several arrays. As a consequence, it results in more constraints of the computational power and local memory sizes to runs this type of ACA, even executing part of the computation externally (outside the MPSoC platform). Therefore, if the target application demands a hardware with low area and power, it results in limited performance and storage, and the feasibility of the attack becomes questionable. However, evaluations had shown that Hourglass requires much less computational effort and storage capacity, being more suitable for a wide range of applications and devices, such as IoT.

The experiments on access-based attacks showed that the proposed techniques were able to improve the Prime+Probe technique. Arrow was capable of discovering the key with only ten encryptions. However, results show that the Arrow can be very fast only if a series of conditions are met. For instance, high accuracy in the NoC timing attack, and quick probing to the cache. This scenario is not unrealistic since the accuracy can be achieved in some conditions as observed in chapter five. Besides, if the crypto-processor manages different tasks, or implements a software protection that inserts timing noise, the time between each access would be much higher. Firecracker used about fifty encryptions but was still half the time executed by Prime+Probe. Analyzing the countermeasures, the strategy to segment the cache in isolated partitions was able to kill all these attacks, since the principal part of the attack, the probe step was not possible. The NoC countermeasures only disturbed the Arrow attack, since it needed accurate measurements. The insertion of random loops did not take any effect on the attacks.

The last part of the experiments evaluated the collision-based attacks. The attack from Bogdanov and the proposed Earthquake were implemented in the target MPSoC.

Results show that they were able to discover the key, but also revealed a complicated issue in the practicability of the attack. These attack tries to identify a variation of five cache hits in the average execution time from pairs of plaintexts. During an AES encryption, several cache hits between the elements can occur making difficult to detect the time from the five cache hits situation (wide collision). Hence, a vast number of samples were elected to the key search space. For Bogdanov attack, it was executed a search of a complexity of $2^{47.5}$, meaning a few days of attack execution. Earthquake was able to reduce such complexity to $2^{45.8}$, decreasing four times the complexity. Considering the computational effort and the time to perform Bogdanov attack, Earthquake presented a more practical attack, also being more suitable for a broad range of applications.

About the execution time of the attacks, table 7.3 presents the average computational time at FPGA (online computation) and at the host computer (offline computation).

Table 7.3: Approximate execution time of the attacks, during the execution of the attack (online time), and after in the post-processing step (offline time).

	Online Computation Time (FPGA)	Offline Compuation Time (Host PC)
Bernstein	2 days	4 hours
Hourglass	1 day	4 hours
Osvik	10 minutes	
Firecracker	5 minutes	
Arrow	2 minutes	
Bogdanov	3 hours	2 days
Earthquake	2 hours	1 day

Evaluating the proposed attacks, the integration of the works on caches and the NoC timing attack provided more than an optimization for the classical cache ACAs; it made possible the execution of such attacks in MPSoCs targeted for applications with limited hardware.

8 CONCLUSION

Multi-Processors Systems-on-Chips became the established hardware platform for a wide variety of applications and devices. High parallelism allied with energy efficiency allowed MPSoCs to accomplish the requirements of the new era on computation, in the first moment defined as Internet-of-Things, and further Internet-of-Everything. Consequently, these solutions will integrate any devices that will be interconnected by the Internet. This high integration through Internet brings several security concerns because all sensitive information stored on these devices will be reachable by external agents. Therefore, the present thesis investigated the presence of vulnerabilities in MPSoCs architectures and evaluated the Network-on-Chip as an essential element to providing system protection.

Works on the state-of-the-art have shown different threats for the System-on-Chips (SoCs), as well as countermeasures against them. However, few works were directed to multi-processor systems. This lack of information motivated this author to review in the bibliography the main attack and defense proposals for SoCs. The objective was to identify the main vulnerabilities that would be present also inside the hardware concept adopted for future devices - the MPSoCs. During this study, the present thesis proposed a new sub-category for the Side-Channel Attacks (SCA). Any SCA attack that explores architecture behavior should be classified as Architectural Channel Attack (ACA).

The first analysis concluded that the Architectural Channel Attacks have a high potential to harm MPSoCs. The increase of complexity in such architectures opens behavior leakages to be explored by attackers. The better efficiency of such attacks compared to conventional SCAs for MPSoCs were justified by:

- ACAs do not demand high specialized instrumentation;
- ACAs has no need to access the target device directly;
- High parallelism running on MPSoCs adds a significant noise for the physical measurements, such as power, EM, etc.;
- More structural complexity represent more potential leakage sources.

A deep study on the Architectural Channel Attacks revealed two components of the MPSoC as the most vulnerable points - the shared cache and the interconnection (a Network-on-Chip when a high number of elements are integrated). The reason regards in the fact that these units are shared among all parts of the system, which enables an

attacker to access direct or indirect information of any part of the MPSoC. Besides, both shared memory and NoC are critical, being part of any typical MPSoC architecture. Consequently, the research took two directions: i) attacks and defenses for caches; ii) attacks and defenses for NoC.

Cache attacks always were the primary threat, since it enabled remote attacks. Kocher (KOCHER, 1996) and Kelsey et al. (KELSEY et al., 1998) introduced such cache timing attack strategy opening a whole new approach to attack electronic devices. Then, Bernstein (BERNSTEIN, 2005) developed the timing attack to break AES, whose technique was further optimized by Neve et al. (NEVE; SEIFERT; WANG, 2006). Different variations of cache timing attacks were proposed by Osvik (OSVIK; SHAMIR; TROMER, 2006) with the access-based attacks; and Bonneau and Mironov (BONNEAU; MIRONOV, 2006) with the collision attacks. These attacks were adapted and implemented in different environments since embedded and mobile devices to cloud servers and virtual machines. The present thesis also changed three techniques to realize in an MPSoC environment: i) a timing attack; ii) an access attack, and iii) a collision attack.

About the countermeasures against cache attacks, the state-of-the-art presents several strategies by software and hardware. In software, there are works that change the cryptography algorithm (REBEIRO; MONDAL; MUKHOPADHYAY, 2010) (ALAWATUGODA; JAYASINGHE; RAGEL, 2011) (BLOMER; KRUMMEL, 2007) and works that modifies the compiler (STEFAN et al., 2013). A high-level software solution also is presented by Crane et al. (CRANE et al., 2015) through dynamic software diversification. About hardware approaches, Alawatugoda et al. (ALAWATUGODA; JAYASINGHE; RAGEL, 2011) show the efficiency of cache partitioning. All these techniques could be applied to the MPSoC environment, but the research focused on the hardware solution only.

The bibliography in ACAs comprises several cache attacks in many scenarios and applications, but only few information without details about the NoC attacks. The authors that refers to the NoC timing attack are (YAO; SUH, 2012) (WASSEL et al., 2014) (SEPULVEDA et al., 2016) (SEPULVEDA et al., 2015) and (STEFAN; GOOSSENS, 2011). No information regarding the threat model, practicability or demonstrations had been presented by these authors. This work did a detailed study around this theme including some experiments, which are described in chapter four - Exploring the NoC Timing Attack. As a consequence, for the first time, the NoC timing attack had its threat model, and practicability analyzed and evaluated. Therefore, the present thesis contributed to

enabling the replication and development of this technique through the publication in (REINBRECHT et al., 2016a).

Moreover, the NoC already has been demonstrated as a promising alternative to enhance system security. The same authors that mentioned the NoC timing attack used the NoC to provide some protection against NoC timing attacks. One of the primary objectives of this thesis was to evaluate the NoC as a potential component to ensure the security of MPSoCs. Hence, it was developed a secure NoC, the Gossip NoC. Gossip NoC aimed to provide protection with minimal area overhead. The proposed countermeasure was capable of avoiding several attacks, but only the ones based on NoC timing attack. A complete security solution in NoC requires more research to design a mechanism that mitigates different types of attacks with minimal area and power overhead or performance degradation.

From attacks for caches - Bernstein timing-based attack, Osvik access-based attack, and Bogdanov collision-based attack - this research developed four new techniques focusing on MPSoCs. All the techniques were a combination of cache attacks with the NoC timing attack. They are the timing-based Hourglass, the access-based Firecracker, the access-based Arrow, and the collision-based Earthquake. All the attacks were designed as an optimization of the reference cache attacks. During the experimental analysis, it was observed that the inclusion of the NoC timing attack was crucial to enable these attacks in an MPSoC implemented for IoT or IoE applications.

During experiments, described in chapter seven, the thesis was able to demonstrate that MPSoCs can be attacked in different ways through the NoC or the shared cache. Secret keys were discovered during experiments using AES cryptography on a real MPSoC environment running in an FPGA. Three standard attacks (Bernstein, Osvik, and Bogdanov) and the four proposed ones (Hourglass, Firecracker, Arrow, and Earthquake) were evaluated under the protection of four different security mechanisms: i) random loops; ii) cache partition; iii) Gossip NoC; and iv) adaptive routing. Two countermeasures for caches and two for NoCs comprises the tests on protection. Analyzing all results, it was possible to conclude the following points:

- Bernstein timing-based attack requires a considerable time and computational effort, taking hours or even days to execute. The attack demands a hardware capable of handling 64 bits variables, multiplications, and an enormous amount of storage. Therefore, if the target application requires a hardware with limited area and power, the feasibility of the attack became questionable. On the other hand, Hourglass

timing-based attack requires much less computational effort and storage capacity, being a practical technique for an extensive range of applications and devices, even IoT and IoE that present resource limitations. Therefore, Hourglass is not only an optimization but a method that make possible to attack future hardware systems.

- Arrow access-based attack was capable of discovering the key with only ten encryptions. However, results show that the Arrow can be very fast only if there is a high accuracy in the NoC timing attack and a high-speed probing to the cache. Such scenario can be realistic if the moments of external interferences in the NoC are higher than the attack accesses. If the IP responsible for the cryptography implements a software countermeasure that inserts random delays in the computation, this ideal scenario for Arrow attack can be easily met.
- Firecracker access-based attack used about fifty encryptions to break AES. The result was five times higher than Arrow, but it was still half the time required by the Osvik attack (Prime+Probe). The benefits of Firecracker regards the flexibility in the attacked scenario. This attack works even if the NoC timing attack is not accurate, and also if the probe of the cache can not be performed in a fast manner.
- All access-based attacks faced difficulties with the cache partitioning countermeasure. This protection uses the strategy to segment the cache in isolated partitions, which avoids the probe step of these attacks.
- The differential collision attacks (Bogdanov and Earthquake) are capable of revealing the secret key of AES. However, both had demonstrated a significant issue that must be addressed before any execution. These attacks try to identify a variation in the timing, caused by five cache hits during AES execution. During an AES encryption, several cache hits between the elements can occur, which makes hard to detect the time from this five cache hits situation (also known as wide collision). Hence, a high number of samples must be selected for the key search space. During experiments, the Bogdanov attack executed a search complexity of $2^{47.5}$, meaning a few days of attack execution. The Earthquake was able to reduce such complexity to $2^{45.8}$, decreasing four times the performance effort. Given the restrictions in the hardware platforms for IoT or IoE, the Earthquake became more suitable as a more realistic threat.

Finally, the work described in the thesis showed the presence of vulnerabilities in a typical MPSoC architecture and the importance of the NoC as an agent to protect the sys-

tem. Consequently, the investigation of new leakage sources, as well as the development of the NoC as the absolute protection element, must be addressed by future research.

9 CONTRIBUTIONS OF THE THESIS

As a result of this research project, several actions and developments were achieved becoming the main contributions of the thesis. Moreover, each accomplishment of the research was published as papers for conferences or journals.

The first contribution regards to the classification of the Side-Channel Attacks that explores architectural leakage as a new sub-category, the Architectural Channel Attacks. This new definition is necessary for the field because the review of the bibliography has revealed that this kind of attack will increase for the next hardware generation. The hardware platforms became more complicated, contributing to the creation of more vulnerabilities in architectures. Moreover, attacks based on physical features became difficult to analyze. Then, the developed attacks for MPSoCs, which merge NoC timing attack with cache attacks, are another contribution. During the experiments, it was possible to conclude that these attack proposals allows attacking an extensive range of devices, including limited platforms found in IoT or IoE segment. Another contribution is the countermeasure against NoC timing attack implemented in the NoC, called Gossip NoC. The practical experimentation of attacks running in a real MPSoC was for the first time executed being another valuable contribution. As a summary, the main contributions are listed below:

1. Definition of the Architectural Channel Attacks, as a sub-category of Side-Channel Attacks;
2. Development of a timing-based attack for NoC-based MPSoCs, the Hourglass;
3. Development of an access-based attack for NoC-based MPSoCs, the Arrow;
4. Development of an access-based attack for NoC-based MPSoCs, the Firecracker;
5. Development of a collision-based attack for NoC-based MPSoCs, the Earthquake;
6. Development of a secure-enhanced NoC Gossip NoC, that protects the system against NoC timing attacks;
7. The first execution of a practical attack inside an MPSoC running in an FPGA breaking an AES cryptography.

Besides, the thesis has published one journal and three conference papers related to the thesis:

- MICPRO Journal: Timing Attack on NoC-based Systems: Prime+Probe Attack and NoC-based Protection (REINBRECHT et al., 2017).

- SBCCI 2016 Conference: Side Channel Attack on NoC-based MPSoCs are practical: NoC Prime+Probe Attack (REINBRECHT et al., 2016b).
- ISVLSI 2016 Conference: Gossip NoC - Avoiding Timing Side-Channel Attacks through Traffic Management (REINBRECHT et al., 2016a).
- Cryptarchi 2016 Workshop: Side Channel Attacks on Networks on Chip (REINBRECHT; BOSSUET; SEPULVEDA, 2016).

There is another publications result of research co-operations:

- ICECS 2016 Conference: DHyANA: A NoC-based neural network hardware architecture (HOLANDA et al., 2016).
- SBCCI 2015 Conference: PHiCIT-Improving hierarchical Networks-on-Chip through 3D silicon photonics integration (REINBRECHT; SEPúLVEDA; SUSIN, 2015).
- ISCAS 2014 Conference: Adaptive multiple switching strategy toward an ideal NoC (MATOS et al., 2014).

10 FUTURE WORKS AND RESEARCH OPPORTUNITIES

During the experimental phase of the thesis, the workflow was very inefficient. The software of each core of the MPSoC had to be developed in a different project of the development tool. Besides, the boot of each binary in the cores running on the FPGA could not be performed by the FPGA tools, since they supported only the boot of one processor. Then, the solution was to convert each binary in a memory initialization file and synthesize hardware project with initialized memories. However, any changes in the software required a new synthesis process. Consequently, the debug process was very costly in time. The development of a framework started at the end of the thesis, where the hardware system was integrated with an IDE (integrated development environment). The IDE and hardware modifications were already developed but still needs final adjustments to be stable and useful for any applications. This IDE allows one to develop the programs in the same project, and compile and download the software of each core during run-time. Therefore, this framework will enable different research projects to take benefit of a flexible MPSoC platform running in an FPGA. Examples of fields that can use such platform are video/image processing, security, machine learning, parallel applications, etc. The Figure 11.2 shows the developed IDE, called MPSoC Glass. Further information can be found in Appendix A.

Regarding the field of security in MPSoCs, there is a need for continuously research novel vulnerabilities and countermeasures. The shared cache memory and the NoC still have potential leakage sources that depend on the hardware platform and application segment of the device. A combination of physical and logical attacks can be explored to enable traditional Side Channel Attacks in a high complex scenario of an MPSoC. Such strategy could compromise even very secure systems. In the aspects of countermeasures, the Gossip NoC has shown that a component of security has to provide multiple protection strategies at the same time to guarantee a minimum level of reliability. Future works must propose new NoC architectures capable of integrating different structures to enhance security being transparent to the applications running o the platform, and without degrading the system performance.

REFERENCES

ADEE, S. The hunt for the kill switch. **IEEE Spectrum**, Piscataway, NJ, USA, v. 45, n. 5, p. 34–39, May 2008.

ALAWATUGODA, J.; JAYASINGHE, D.; RAGEL, R. Countermeasures against bernstein's remote cache timing attack. In: **2011 6th International Conference on Industrial and Information Systems**. Kandy, Sri Lanka: IEEE, 2011. p. 43–48.

ALTERA, I. F. **Nios II Classic Processor Reference Guide**. Available at: <https://www.altera.com/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf>. Accessed at 2017-10-07.

ALY, H.; ELGAYYAR, M. Attacking aes using bernstein's attack on modern processors. In: **Progress in Cryptology – AFRICACRYPT 2013**. Cairo, Egypt: Springer Berlin Heidelberg, 2013. p. 127–139.

ANDRYSCO, M. et al. On subnormal floating point and abnormal timing. In: **Proceedings of the 2015 IEEE Symposium on Security and Privacy**. Washington, DC, USA: IEEE Computer Society, 2015. (SP '15), p. 623–639.

APECECHEA, G. I. et al. **Fine grain Cross-VM Attacks on Xen and VMware are possible!** Available at: <http://eprint.iacr.org/>. Accessed at 2017-10-07.

ARM. **White Paper - big.LITTLE Technology: The future of mobile**. Available at: <https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf>. Accessed at 2017-10-07.

BAYON, P. et al. Contactless electromagnetic active attack on ring oscillator based true random number generator. In: **Constructive Side-Channel Analysis and Secure Design: Third International Workshop, COSADE 2012, Proceedings**. Darmstadt, Germany: Springer, 2012. (Lecture Notes in Computer Science, v. 7275), p. 151–166.

BENGER, N. et al. Ooh aah... just a little bit: A small amount of side channel can go a long way. In: **Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2014**. New York, NY, USA: Springer-Verlag New York, Inc., 2014. p. 75–92.

BERNSTEIN, D. J. **Cache Timing Attacks on AES**. 2005. Available at: <https://cr.yp.to/antiforgery/cachetiming-20050414.pdf>. Accessed at 2016-12-31.

BLOMER, J.; KRUMMEL, V. Analysis of countermeasures against access driven cache attacks on aes. In: **Selected Areas in Cryptography: 14th International Workshop, SAC 2007, Revised Selected Papers**. Ottawa, Canada: Springer Berlin Heidelberg, 2007. p. 96–109.

BOGDANOV, A. et al. Differential cache-collision timing attacks on aes with applications to embedded cpus. In: **Topics in Cryptology - CT-RSA 2010: The Cryptographers' Track at the RSA Conference 2010**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 235–251.

BONNEAU, J.; MIRONOV, I. Cache-collision timing attacks against aes. In: **Cryptographic Hardware and Embedded Systems - CHES 2006**. Yokohama, Japan: Springer Berlin Heidelberg, 2006. p. 201–215.

BORHANI, A.; MOVAGHAR, A.; COLE, R. A new deterministic fault tolerant wormhole routing strategy for k-ary 2-cubes. In: **Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference on**. Coimbatore, India: IEEE, 2010. p. 1–7.

BOSSUET, L.; TORRES, L. **Foundations of Hardware IP Protection**. [S.l.]: Springer International Publishing, 2017. VII, 240 p.

BRICKELL, E. et al. **Software mitigations to hedge AES against cache-based software side channel vulnerabilities.** 2006. Available at: <https://eprint.iacr.org/2006/052.pdf>. Accessed at 2016-12-31.

CHANDHOK, R. The internet of everything. In: **2014 IEEE Hot Chips 26 Symposium (HCS)**. Cupertino, CA, USA: [s.n.], 2014. p. 1–29.

CHEN, P. et al. Multiprocessor system-on-chip profiling architecture: Design and implementation. In: **15th International Conference on Parallel and Distributed Systems (ICPADS)**. Shenzhen, China: [s.n.], 2009. p. 519–526.

CRANE, S. et al. Thwarting cache side-channel attacks through dynamic software diversity. In: **Annual Network and Distributed System Security Symposium, NDSS**. San Diego, California, USA: [s.n.], 2015. v. 15, p. 8–11.

DAEMEN, J.; RIJMEN, V. **The Design of Rijndael**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

FIORIN, L.; PALERMO, G.; SILVANO, C. A security monitoring service for nocs. In: **Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis**. New York, NY, USA: ACM, 2008. (CODES+ISSS 08), p. 197–202.

GANDOLFI, K.; MOURTEL, C.; OLIVIER, F. Electromagnetic analysis: Concrete results. In: **Cryptographic Hardware and Embedded Systems - CHES 2001**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. p. 251–261.

GEBOTYS, C. H.; GEBOTYS, R. J. Secure elliptic curve implementations: An analysis of resistance to power-attacks in a dsp processor. In: **Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems**. London, UK, UK: Springer-Verlag, 2003. (CHES 02), p. 114–128.

GIRAO, G.; BARCELOS, D.; WAGNER, F. Performance and energy evaluation of memory hierarchies in noc-based mpsocs under latency. In: **2009 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)**. Florianopolis, Brazil, Brazil: [s.n.], 2009. p. 127–132.

GOOSSENS, K.; DIELISSEN, J.; RADULESCU, A. Aethereal network on chip: concepts, architectures, and implementations. **IEEE Design Test of Computers**, v. 22, n. 5, p. 414–421, Sept 2005.

GUIN, U.; DIMASE, D.; TEHRANIPOOR, M. Counterfeit integrated circuits: Detection, avoidance, and the challenges ahead. **J. Electron. Test.**, Kluwer Academic Publishers, Norwell, MA, USA, v. 30, n. 1, p. 9–23, feb 2014.

GULLASCH, D.; BANGERTER, E.; KRENN, S. Cache games - bringing access-based cache attacks on aes to practice. In: **2011 IEEE Symposium on Security and Privacy (SP)**. Washington, DC, USA: IEEE Computer Society, 2011. p. 490–505.

HOLANDA, P. et al. Dhyana: A noc-based neural network hardware architecture. In: **2016 IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. Cairo, Egypt: [s.n.], 2016. p. 177–180.

HUTTER, M.; SCHMIDT, J.-M. The temperature side channel and heating fault attacks. In: **Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013. Revised Selected Papers**. Berlin, Germany: Springer International Publishing, 2014. p. 219–235.

ICC, I. C. of C. **The Economic Impacts of Counterfeiting and Piracy - 2017 Frotier Report**. Available at: <http://www.inta.org/Communications/Documents/2017_Frontier_Report.pdf>. Accessed at 2017-10-07.

IRAZOQUI, G.; EISENBARTH, T.; SUNAR, B. S$a: A shared cache attack that works across cores and defies vm sandboxing – and its application to aes. In: **2015 IEEE Symposium on Security and Privacy (SP)**. San Jose, CA, USA: [s.n.], 2015. p. 591–604.

IRAZOQUI, G. et al. Wait a minute! a fast, cross-vm attack on aes. In: **Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014**. [S.l.: s.n.].

IRAZOQUI, G. et al. Fine grain cross-vm attacks on xen and vmware are possible! **IACR Cryptology ePrint Archive**, p. 248, 2014. Accessed at 2017-10-07.

ISO/IEC. **Information technology – Security techniques – Authenticated encryption**. 29 p. Available at: <http://www.iso.org/iso/catalogue_detail.htm?csnumber=46345>. Accessed at 2017-10-07.

ITRS, I. T. R. for S. **2015 ITRS 2.0 OFFICIAL PUBLICATION**. 2016. Available at: <https://www.dropbox.com/sh/3jfh5fq634b5yqu/AADYT8V2Nj5bX6C5q764kUg4a?dl=0>. Accessed at 2017-10-07.

JAYASINGHE, D. et al. Remote cache timing attack on advanced encryption standard and countermeasures. In: **Information and Automation for Sustainability (ICIAFs), 2010 5th International Conference on**. [S.l.: s.n.], 2010. p. 177–182.

JERRAYA, A.; WOLF, W. **Multiprocessor Systems-on-Chips**. [S.l.]: Morgan Kaufmann, 2004. 608 p.

KARRI, R. et al. Concurrent error detection of fault-based side-channel cryptanalysis of 128-bit symmetric block ciphers. In: **Design Automation Conference, 2001. Proceedings**. Las Vegas, NV, USA, USA: IEEE, 2001. p. 579–584.

KELSEY, J. et al. Side channel cryptanalysis of product ciphers. In: **Computer Security — ESORICS 98: 5th European Symposium on Research in Computer Security**. Louvain-la-Neuve, Belgium: Springer Berlin Heidelberg, 1998. p. 97–110.

KOCHER, P. C. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: **Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology**. London, UK, UK: Springer-Verlag, 1996. (CRYPTO 96), p. 104–113.

KOCHER, P. C.; JAFFE, J.; JUN, B. Differential power analysis. In: **Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology**. London, UK, UK: Springer-Verlag, 1999. (CRYPTO '99), p. 388–397.

KOMAR, M.; EDELEV, S.; KOUCHERYAVY, Y. Handheld wireless authentication key and secure documents storage for the internet of everything. In: **2016 18th Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)**. St. Petersburg, Russia: IEEE, 2016. p. 120–130.

LIBGCRYPT, G. **The libgcrypt reference manual.** Available at: <http://www.gnupg.org/documentation/manuals/gcrypt/>. Accessed at 2017-10-07.

LIU, F. et al. Last-level cache side-channel attacks are practical. In: **Security and Privacy (SP), 2015 IEEE Symposium on**. San Jose, CA, USA: IEEE, 2015. p. 605–622.

MAIMUT, D.; REYHANITABAR, R. Authenticated encryption: Toward next-generation algorithms. **IEEE Security Privacy**, v. 12, n. 2, p. 70–72, Mar 2014.

MANCILLAS, C. et al. Extending multicore architectures with cryptoptocessors and parallel cryptography. In: **Colloque national du GDR SOC-SIP**. Paris, France: [s.n.], 2014.

MASOOMI, M.; MASOUMI, M.; AHMADIAN, M. A practical differential power analysis attack against an fpga implementation of aes cryptosystem. In: **Information Society (i-Society), 2010 International Conference on**. London, UK, UK: IEEE, 2010. p. 308–312.

MATOS, D. et al. Adaptive multiple switching strategy toward an ideal noc. In: **2014 IEEE International Symposium on Circuits and Systems (ISCAS)**. Melbourne VIC, Australia: IEEE, 2014. p. 1014–1017.

MEYR, H. On core and more: a design perspective for system-on-chip. In: **IEEE Workshop on Signal Processing Systems. SIPS 97 - Design and Implementation**. Leicester, UK, UK: IEEE, 1997. p. 60–63.

MORADI, A.; MISCHKE, O.; PAAR, C. One attack to rule them all: Collision timing attack versus 42 aes asic cores. **IEEE Transactions on Computers**, v. 62, n. 9, p. 1786–1798, Sept 2013.

NEVE, M.; SEIFERT, J.-P.; WANG, Z. A refined look at bernsteiń s aes side-channel analysis. In: **Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security**. New York, NY, USA: ACM, 2006. p. 369–369.

ORCUTT, J. S. et al. Nanophotonic integration in state-of-the-art cmos foundries. **Opt. Express**, OSA, v. 19, n. 3, p. 2335–2346, Jan 2011.

OREN, Y. et al. The spy in the sandbox: Practical cache attacks in javascript and their implications. In: **Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security**. Denver, Colorado, USA: ACM, 2015. (CCS 15), p. 1406–1418.

ORS, S. et al. Power-analysis attack on an asic aes implementation. In: **International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.** Las Vegas, NV, USA, USA: IEEE, 2004. v. 2, p. 546–552.

OSVIK, D. A.; SHAMIR, A.; TROMER, E. Cache attacks and countermeasures: The case of aes. In: . Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 1–20.

PERCIVAL, C. Cache missing for fun and profit. In: **Proceedings of BSDCan**. [S.l.: s.n.], 2005.

POLARSSL. **PolarSSL: Straightforward,secure communication.** Available at: <www.polarssl.org>. Accessed at 2017-10-07.

PROJECT, T. O. S. **OpenSSL: The open source toolkit for SSL/TLS.** Available at: <www.openssl.org>. Accessed at 2017-10-07.

REBEIRO, C.; MONDAL, M.; MUKHOPADHYAY, D. Pinpointing cache timing attacks on aes. In: **2010 23rd International Conference on VLSI Design**. Bangalore, India: IEEE, 2010. p. 306–311.

REINBRECHT, C.; BOSSUET, L.; SEPULVEDA, J. Side-channel attacks on networks-on-chips. In: **14th Cryptarchi 2016 – Cryptographic Architectures Embedded in Reconfigurable Devices.** La Grande Motte, France: [s.n.], 2016.

REINBRECHT, C.; SEPúLVEDA, J.; SUSIN, A. Phicit - improving hierarchical networks-on-chip through 3d silicon photonics integration. In: **2015 28th Symposium on Integrated Circuits and Systems Design (SBCCI)**. Salvador, Brazil: ACM, 2015. p. 1–7.

REINBRECHT, C. et al. Gossip noc - avoiding timing side-channel attacks through traffic management. In: **IEEE Computer Society Annual Symposium on VLSI (ISVLSI 16)**. Pittsburgh, USA: IEEE, 2016. p. 601–606.

REINBRECHT, C. et al. Side channel attack on noc-based mpsocs are practical: Noc prime+probe attack. In: **29th Symposium on Integrated Circuits and Systems Design (SBCCI)**. Belo Horizonte, Brazil: IEEE, 2016. p. 1–6.

REINBRECHT, C. et al. Timing attack on noc-based systems: Prime+probe attack and noc-based protection. **Microprocessors and Microsystems**, v. 51, Jan 2017.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. **Commun. ACM**, ACM, New York, NY, USA, v. 21, n. 2, p. 120–126, feb 1978.

ROSA, T. **Communication support in multi-core architectures through hardware mechanisms and standardized programming interfaces**. Thesis (Theses) — Universite Grenoble Alpes, Apr 2016.

SEPULVEDA, J. et al. Noc-based protection for soc time-driven attacks. **Embedded Systems Letters, IEEE**, v. 7, n. 1, p. 7–10, March 2015.

SEPULVEDA, J.; FLOREZ, D.; GOGNIAT, G. Efficient and flexible noc-based group communication for secure mpsocs. In: **2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)**. Mexico City, Mexico: IEEE, 2015. p. 1–6.

SEPULVEDA, J. et al. ddynamic noc buffer allocation for mpsoc timing side channel attack protection. In: **Network-on-Chip, timing, side channel attack (LASCAS)**. Florianópolis, Brazil: IEEE, 2016. p. 91–94.

SEPULVEDA, J. et al. Elastic security zones for noc-based 3d-mpsocs. In: **21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. Marseille, France: IEEE, 2014. p. 506–509.

SEPULVEDA, J. et al. Hierarchical noc-based security for mp-soc dynamic protection. In: **Circuits and Systems (LASCAS), 2012 IEEE Third Latin American Symposium on**. Playa del Carmen, Mexico: IEEE, 2012. p. 1–4.

SEPULVEDA, J. et al. Authentication and access control qoss (quality of security service) for noc-based systems. In: **21st IASTED – International Conference on Parallel and Distributed Computing and Systems.** Boston, USA: ACTA Press, 2009.

SPREITZER, R.; GéRARD, B. Towards more practical time-driven cache attacks. In: NACCACHE, D.; SAUVERON, D. (Ed.). **Information Security Theory and Practice. Securing the Internet of Things**. [S.l.]: Springer Berlin Heidelberg, 2014, (Lecture Notes in Computer Science, v. 8501). p. 24–39.

SPREITZER, R.; PLOS, T. On the applicability of time-driven cache attacks on mobile devices (extended version). In: **Network and System Security**. [S.l.]: Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science). p. 656–662.

STEFAN, D. et al. Eliminating cache-based timing attacks with instruction-based scheduling. In: **Computer Security - ESORICS 2013: 18th European Symposium on Research in Computer Security**. Egham, UK: Springer Berlin Heidelberg, 2013. p. 718–735.

STEFAN, R.; GOOSSENS, K. Enhancing the security of time-division-multiplexing networks-on-chip through the use of multipath routing. In: **Proceedings of the 4th International Workshop on Network on Chip Architectures**. Porto Alegre, Brazil: ACM, 2011. (NoCArc 11), p. 57–62.

SUH, G. E.; DEVADAS, S. Physical unclonable functions for device authentication and secret key generation. In: **2007 44th ACM/IEEE Design Automation Conference**. San Diego, CA, USA: IEEE, 2007. p. 9–14.

TATAS, K.; SAWA, S.; KYRIACOU, C. Low-cost fault-tolerant routing for regular topology nocs. In: **21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)**. Marseille, France: IEEE, 2014. p. 566–569.

TEHRANIPOOR, M.; WANG, C. (Ed.). [S.l.]: Springer New York, 2012.

TIRI, K. Side-channel attack pitfalls. In: **Proceedings of the 44th Annual Design Automation Conference**. San Diego, California, USA: ACM, 2007. (DAC '07), p. 15–20.

TSUNOO, Y. et al. Cryptanalysis of des implemented on computers with cache. In: **CHES 2003: 5th International Workshop on Cryptographic Hardware and Embedded Systems**. Cologne, Germany: Springer Berlin Heidelberg, 2003. v. 2779, p. 62–76.

WASSEL, H. et al. Networks on chip with provable security properties. **Micro, IEEE**, v. 34, n. 3, p. 57–68, May 2014.

WEISS, M.; HEINZ, B.; STUMPF, F. A cache timing attack on aes in virtualization environments. In: **Financial Cryptography and Data Security**. [S.l.]: Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7397). p. 314–328.

WEISS, M. et al. On cache timing attacks considering multi-core aspects in virtualized embedded systems. In: **The 6th International Conference on Trustworthy Systems (InTrust 2014)**. Beijing, China: Springer International Publishing, 2014. p. 151–167.

WOLF, W.; JERRAYA, A. A.; MARTIN, G. Multiprocessor system-on-chip (mpsoc) technology. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 27, n. 10, p. 1701–1713, Oct 2008.

XINJIE, Z. et al. Robust first two rounds access driven cache timing attack on aes. In: **2008 International Conference on Computer Science and Software Engineering**. Hubei, China: IEEE, 2008. v. 3, p. 785–788.

YAO, W.; SUH, E. Efficient timing channel protection for on-chip networks. In: **NOCS '12 Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip**. Lyngby, Denmark: IEEE, 2012. p. 142–151.

YAROM, Y.; FALKNER, K. Flush+reload: A high resolution, low noise, l3 cache side-channel attack. In: **Proceedings of the 23rd USENIX Conference on Security Symposium**. San Diego, CA: USENIX Association, 2014. (SEC 14), p. 719–732.

YOUNIS, Y. A. et al. A new prime and probe cache side-channel attack for cloud computing. In: **2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)**. Liverpool, UK: IEEE, 2015. p. 1718–1724.

ZHANG, Y. et al. Cross-tenant side-channel attacks in paas clouds. In: **Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security**. Scottsdale, Arizona, USA: ACM, 2014. p. 990–1003.

11 MPSOC GLASS IDE

In order to understand the MPSoC Glass IDE Design Flow, first it is presented the Altera Design Flow.

11.1 Altera Design Flow

Altera provides a software design environment for developing several applications in a MPSoC running on a FPGA. Therefore, it is necessary to be familiar with the Nios II Embedded System Design Flow, which consists of three stages of development: i) hardware design, ii) software design, and iii) system design.

First any project establishes the hardware platform, which consist typically of a processor, memories, peripherals, and an interconnection mechanism. Regarding MPSoCs, our system implements twelve processors with local memories, a shared cache memory L2 and an UART serial, all integrated through a NoC. The software design stage refers to the applications that are mapped to the target platform. The composition of both stages comprises the system stage. This design flow is better described below, and depicted in figure 11.1:

1. Create a new Quartus II project for your system. The Altera Quartus Design Software do the system hardware requirements analysis and implements it in an Altera FPGA chip. However at this point you can only focus in storing your project in a directory, assigning a name and choosing the FPGA device used on your board.
2. Inside Quartus II, open Qsys system integration tool, which is used to add components to the system and configure the selected components to meet the design requirements. Therefore, we have to specify Nios II as the embedded processor, the local memories, the peripherals, the network interface and the shared cache. After connecting the hardware components, the address map of each processor needs to be assigned. The physical addresses organizes the address reserved for the local memories, as well as the peripherals, which includes the system-on-chip shared memory and the network interface. Generate the hardware description through the corresponding button in the Qsys tool. Then, automatically it generates the interconnect logic to integrate the components in the hardware system and the files required for the synthesis.

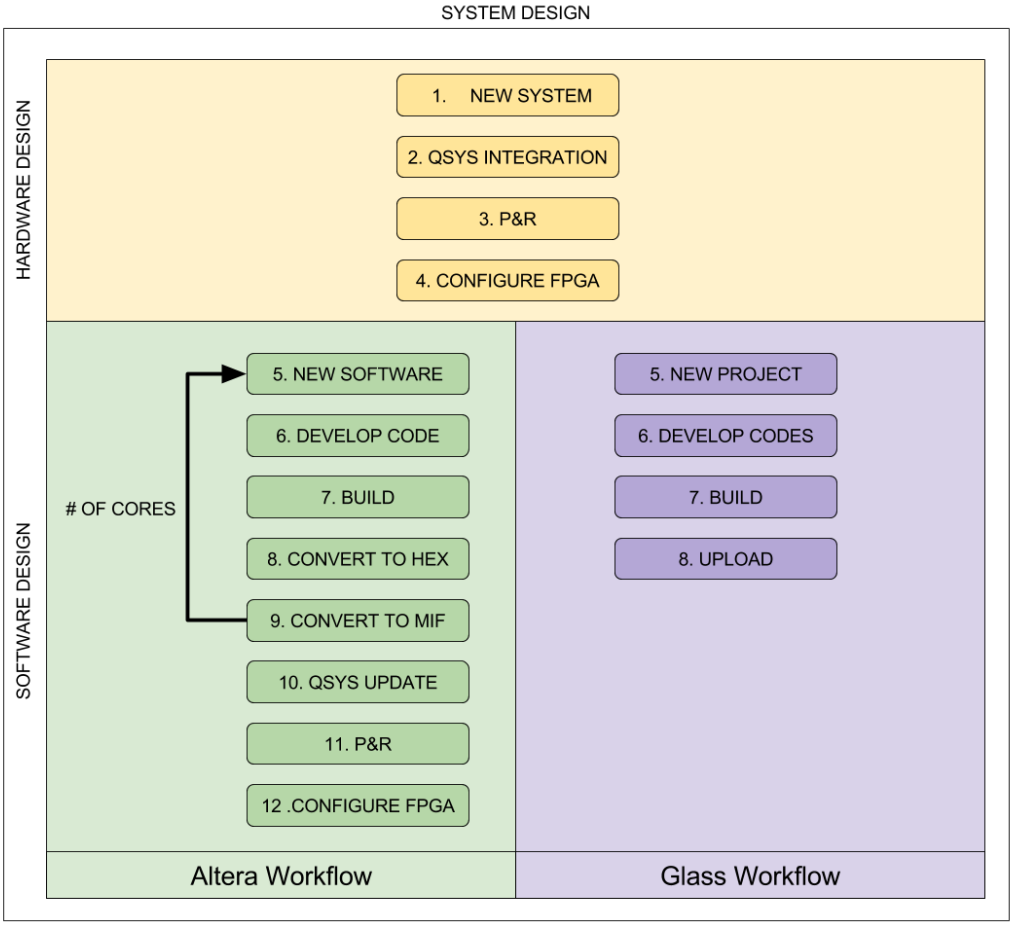
3. Instantiate the module generated by the Qsys tool into the Quartus II project and then start the analysis and synthesis stage, followed by the placing and routing of the design.
4. Now a bitstream generated by Quartus II will be ready to program the FPGA in the board.
5. Open the software build tools for Nios II and create a project for your developed hardware system. Choose the target processor in order to develop the software and create a baremetal project. The processor's boot code is automatically generated and you can write your own C application project.
6. Write the code of the application for the target processor.
7. Build the software system, which means compile the application together with the boot code. After compiling your application source code, the result is an Executable and Linking Format File (.elf).
8. Converts the ELF file to an hexadecimal file. Open a file format conversion tool called elf2hex which converts a .elf file to a hexadecimal one.
9. At this step you convert your .elf file to Quartus II Memory Initialization File (.mif) using the elf2mif file format conversion tool. The MIF will be used to initialize the local memories of the processors.
10. After all these procedures, you go back to the Qsys tool in order to locate the .mif file in the initialization setting of each processor local memory. Updates all hardware files, regenerating the hardware system.
11. Again you go to the analysis and synthesis stage, followed by the placing and routing of the design.
12. Now you are able to send a bitstream in an FPGA board and the process is completed.

11.2 Glass Flow

The proposed IDE allows a reduced design flow to develop, compile and upload applications to the target MPSoC architecture running in FPGA. The stages related to Glass Flow are described below (Figure 11.1):

1. Open the MPSoc Glass IDE.

Figure 11.1: Nios II Embedded System Design and MPSoC Glass Flowcharts.



2. The setup stage consists in configuring the compiler while the serial ports available are recognized.
3. Start developing your application by selecting an IP and writing the C code corresponding to it.
4. Select at least one IP to be compiled. Besides compiling the code relative to the selected IP, the compile button execute the software boatloader that generates the .elf file and also call an Altera tool which converts it file into hexadecimal ones.
5. The system boot is the last one to be called since it sends the compiled files thru the serial port to the FPGA.
6. Any results are shown at the Console.

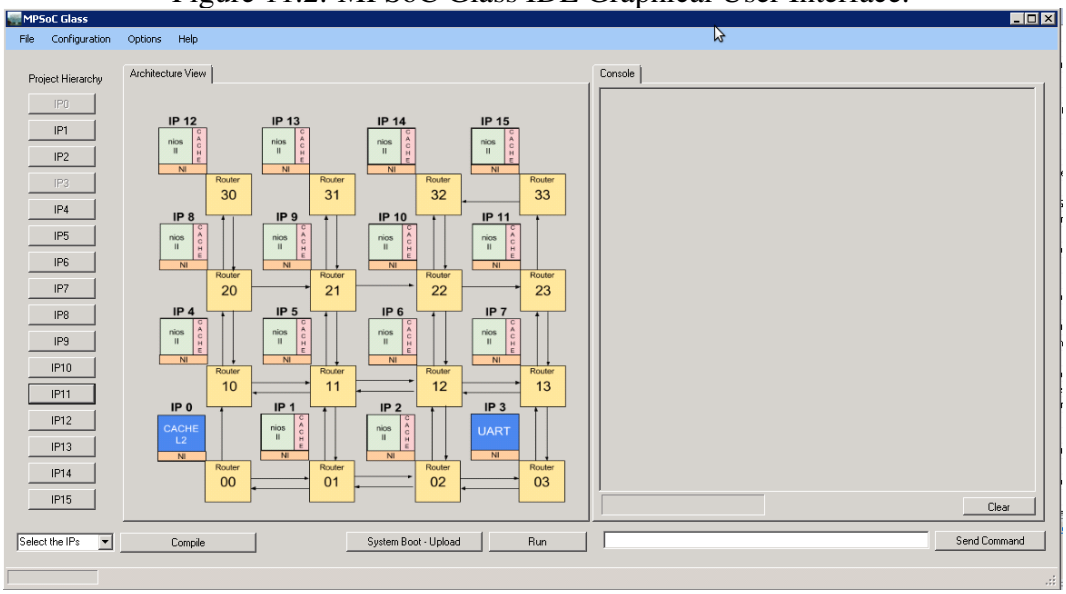
11.3 MPSoC Glass IDE

Idealized not only for evaluating but also software simulating of the multiple processes that simultaneously occurs in a system-on-a-chip, the MPSoC Glass IDE supports

C programming and compilation. Besides data access and I/O performance. According to these needs, the Visual C was the perfect development environment of this IDE, once it introduces the Microsoft C and its .NET Framework programming library allied to dynamic design possibilities such as project templates, property pages, code wizards, an object model and more.

The interface is shown in Figure 11.2. There are many applications for MPSoCs that the interface can support. However the focus is the development, compilation and execution of applications in a MPSoC running in a FPGA. It basically consists in a sequence of steps which leads to a software simulation of data flow between fifteen IPs. IP0 and IP3 are unavailable to any application of the interface because they are respectively the shared cache and the external interface. Each IP button contains a programming environment where you can write your own code and save it as a text file. These files will become executable ones when the compiler is configured and the process of cross-compilation is called.

Figure 11.2: MPSoC Glass IDE Graphical User Interface.



11.3.1 Cross-compilation Feature

When talking about embedded systems, the cross-compilation is mandatory. Debugging and testing requires more resources than are usually available so a compiler from a host platform can be less involved and less prone to errors than using the native compilation.

The configuration button at the main menu contains the compiler setup. By click-

ing on it, a window is open and it is possible to define the compiler address. Then, an IP must be selected and the compiler button is ready to perform the cross-compilation. The code to perform the cross-compilation activity is described down below.

```
1  using System;
2  using System.Diagnostics;
3
4  Process cmd = new Process();
5
6  cmd.StartInfo.FileName = compiler_address;
7  cmd.StartInfo.Arguments = toCompile;
8
9  cmd.StartInfo.RedirectStandardInput = true;
10 cmd.StartInfo.RedirectStandardOutput = true;
11 cmd.StartInfo.RedirectStandardError = true;
12 cmd.StartInfo.UseShellExecute = false;
13 cmd.Start();
14 cmd.WaitForExit();
15
16 String msg = cmd.StandardOutput.ReadToEnd();
```

Code 11.1 – Code to call external compiler to the Cross-compilation task.

A System.Diagnostics.Process component is a useful tool for starting, stopping, controlling, and monitoring external applications, in this case a compiler. You can use it to start a new process and once initialized, it can be used to obtain information about the running process.

The method Process.Start initializes a process resource by specifying its file name and command-line arguments. The file name compiler_address at line 6 does not need to represent an executable file. It can be of any file type for which the extension has been associated with an application installed on the system. The argument toCompile at line 7 refers to a string which contains the selected IP. The class ProcessStartInfo is used for a better controlling over the process started.

The properties RedirectStandardInput (line 9), RedirectStandardOutput (line 10) and RedirectStandardError (line 11) are a console application that reads and sorts text input, while the property UseShellExecute as false (line 12) enables the redirection of the input, output, and error streams.

The process itself starts at line 13 and the method WaitForExit (line 14) makes the

current thread wait until the associated process terminates. At last the string msg (line 16) reads the executable file on the correct output.

11.3.2 Binaries Upload Feature

Hardware designers goal is to make sure that the volatile memory will be mapped into the address space, so that the processor's boot code and the address to the boot code will be there for the processor to read when the power is turned on and reset is released. So the first code a processor runs is the boot loader, whose main task is to load the operating system and pass over the execution to it after setting up necessary environment for its setup. The boot loader is previously generated with the Altera workflow, and it is used the same for any application. Since this version of MPSoC Glass, the hardware elements can not be modified or parametrized, the boot remains the same. Altera boot program performs the essential initialization including programming the clocks, stacks, interrupt set up etc. Then, the boot points to the initial address that the applications are compiled. However, to execute the boot and application, all these data must be uploaded to the system.

What concerns to the MPSoC Glass IDE, there's a button called "System Boot - Upload" which perform the binaries (boot and application compiled) upload feature. By clicking on it, the serial port available is opened. The binaries, saved as an hexadecimal file, are read byte a byte, composing packages. These are boot packages that contain a header and the bytes of the binaries. The header is generated according to the protocol used by the NoC. The status of the conversion and upload is showed at the console while its evolution is informed at a progress bar. When all processors are ready the last boot release packet is sent.

12 MPSOC GLASS API

Another contribution of the thesis was a simple API responsible to provide the basic functions to communicate through the NoC and to ask cryptography services from the trusted IP.

12.1 NoC Communication Services

The code 12.1 shows the main functions from the NoC communication services.

```
1 void noc_send(unsigned int *data, unsigned char size, unsigned
     char dest);
2 void noc_recv(unsigned int *data, unsigned char size, unsigned
     char *source);
```

Code 12.1 – Functions provided by NoC API.

Below, the code 12.2 present the source code from each function.

```
1
2 #define HPS_NI_NOC_STATUS           0x00088000
3 #define HPS_NI_NOC_CONF             0x00088004
4 #define HPS_NI_NOC_DATA_RX          0x00088008
5 #define HPS_NI_NOC_DATA_TX          0x0008800C
6
7     unsigned int volatile * const noc_ni_status = (unsigned int
         *)  HPS_NI_NOC_STATUS;
8     unsigned int volatile * const noc_ni_config = (unsigned int
         *)  HPS_NI_NOC_CONF;
9     unsigned int volatile * const noc_ni_data_rx = (unsigned int
          *) HPS_NI_NOC_DATA_RX;
10    unsigned int volatile * const noc_ni_data_tx = (unsigned int
          *) HPS_NI_NOC_DATA_TX;
11
12 void noc_send(unsigned int *data, unsigned char size, unsigned
     char dest)
13 {
14     unsigned char i = 0;
```

```
15
16      while (i < size)
17      {
18          *noc_ni_data_tx = data[i];
19          i = i + 1;
20       }
21
22      //PKT MOUNT
23      data =  ((size << (1+2+10+1+1)) | (dest << 2) | 3);
24
25      *noc_ni_config = data;
26
27      status = *noc_ni_status;
28
29      //WAIT END OF MSG
30      while ( (status & 3) == 0)
31      {
32          status = *noc_ni_status;
33      }
34
35  }
36
37  void noc_recv(unsigned int *data, unsigned char *size, unsigned
        char *source)
38  {
39      unsigned char cont = 0;
40      unsigned int status;
41      //WAIT RECEPTION OF REQUEST
42      do
43      {
44          status = *noc_ni_status;
45      } while ( (status & 8) == 0);
46
47      *source = ((status >> (10+1+1+2)) & 255);
48      *size = ((status >> (1+1+2)) & 255);
49
```

```
50      while (cont < *size)
51      {
52          data[cont] = *noc_ni_data_rx;
53          cont = cont + 1;
54      }
55  }
```

Code 12.2 – Source code of NoC services.

12.2 Cryptography Services

The code 12.3 shows the main functions from the cryptography services.

```
1  void send_plaintext(unsigned char *plaintext);
2  void wait_ciphertext(unsigned char *ciphertext);
```

Code 12.3 – Functions provided by Crypto API..

Below, the code 12.4 present the source code from each function.

```
1  void send_plaintext(unsigned char *plaintext)
2  {
3      unsigned int pl[4];
4
5      pl[0] =   (plaintext[12]<<(8*3)) | (plaintext[13]<<(8*2)) |
           (plaintext[14]<<(8*1)) | (plaintext[15]<<(8*0));
6      pl[1] =   (plaintext[8]<<(8*3))  | (plaintext[9]<<(8*2))  |
           (plaintext[10]<<(8*1)) | (plaintext[11]<<(8*0));
7      pl[2] =   (plaintext[4]<<(8*3))  | (plaintext[5]<<(8*2))  |
           (plaintext[6]<<(8*1))  | (plaintext[7]<<(8*0));
8      pl[3] =   (plaintext[0]<<(8*3))  | (plaintext[1]<<(8*2))  |
           (plaintext[2]<<(8*1))  | (plaintext[3]<<(8*0));
9
10     noc_send(pl,4,13);
11
12 }
13
14 void wait_ciphertext(unsigned char *ciphertext)
15 {
```

```
16      unsigned int ci[4];
17      unsigned char size; //not used in this context
18      unsigned char source;//not used in this context
19
20      noc_recv(ci,&size,&source);
21
22      ciphertext[15]   = (unsigned char) (ci[0] >> (8*0)) & 255;
23      ciphertext[14]   = (unsigned char) (ci[0] >> (8*1)) & 255;
24      ciphertext[13]   = (unsigned char) (ci[0] >> (8*2)) & 255;
25      ciphertext[12]   = (unsigned char) (ci[0] >> (8*3)) & 255;
26      ciphertext[11]   = (unsigned char) (ci[1] >> (8*0)) & 255;
27      ciphertext[10]   = (unsigned char) (ci[1] >> (8*1)) & 255;
28      ciphertext[9]    = (unsigned char) (ci[1] >> (8*2)) & 255;
29      ciphertext[8]    = (unsigned char) (ci[1] >> (8*3)) & 255;
30      ciphertext[7]    = (unsigned char) (ci[2] >> (8*0)) & 255;
31      ciphertext[6]    = (unsigned char) (ci[2] >> (8*1)) & 255;
32      ciphertext[5]    = (unsigned char) (ci[2] >> (8*2)) & 255;
33      ciphertext[4]    = (unsigned char) (ci[2] >> (8*3)) & 255;
34      ciphertext[3]    = (unsigned char) (ci[3] >> (8*0)) & 255;
35      ciphertext[2]    = (unsigned char) (ci[3] >> (8*1)) & 255;
36      ciphertext[1]    = (unsigned char) (ci[3] >> (8*2)) & 255;
37      ciphertext[0]    = (unsigned char) (ci[3] >> (8*3)) & 255;
38  }
```

Code 12.4 – Source code of Crypto services.