DISS. ETH NO. 23929

Security Considerations for VLSI-Based Symmetric Encryption Devices

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

MICHAEL MÜHLBERGHUBER

Dipl.-Ing.,
Graz University of Technology

born on 23.10.1984
citizen of
Austria

accepted on the recommendation of

Prof. Dr. Hubert Kaeslin, examiner
Prof. Dr. Srdjan Capkun, co-examiner
Prof. Dr. Ir. Ingrid Verbauwhede, co-examiner

2016

Acknowledgments

Writing down a document like the present Ph.D. thesis is mainly a *one-man show*. However, the more important part—the actual content of it—is the result of a four year journey working on projects with many different people. I would like to take this opportunity to highlight several of them who deserve to be mentioned, as they significantly influenced this thesis in one way or the other.

First and foremost I thank Prof. Hubert Kaeslin from the Microelectronics Design Center for supervising my work and his support throughout the previous years. I am also grateful to my co-examiners, Prof. Srdjan Capkun and Prof. Ingrid Verbauwhede. This work was carried out as part of a collaboration between industry and academia and would not have been possible without the financial support from the Commission for Technology and Innovation (CTI). The regular and fruitful discussions with Dr. Richard De Moliner from Omnisec AG provided me an insight into the requirements for industrial products. Thanks a lot to Dr. Michael Hutter and Dr. Thomas Korak from the Institute of Applied Information Processing and Communication (IAIK) at Graz University of Technology for our collaborations.

Many colleagues at the Integrated Systems Laboratory (IIS) influenced my work significantly. I would like to express my deepest gratitude to Dr. Frank K. Gürkaynak and Dr. Norbert Felber. Without their help I would have most-likely never joined the IIS and would have missed a great opportunity. Furthermore, I appreciated working together with Dr. Patrick Maechler, Christoph Keller, Michael Gautschi, and Beat Muheim at the IIS. The output from many interesting discussions with the following people regularly inspired my daily work: Michael Schaffner, David Bellasi, Pirmin Vogel, Andreas

Traber, and Dr. Pierre Greisen. Administrative tasks and infrastructure support are often overlooked. Thanks Christoph Wicki for the smoothly working IT systems and Christine Haller for keeping away most of the office work, letting me focus on my research. Hansjörg Gisler constantly supported me with shipments and component procurements. I had the opportunity to supervise many brilliant students as part of master theses and semester projects. Many thanks to all of them for their great work and commitment. In particular I would like to emphasize the work done by Philipp Dunst, Cyril Arnould, Marco Eppenberger, and Stefan Mach.

Ultimately I would like to thank my family and friends, including some of my co-workers, who became more than colleagues during the time of my Ph.D. Mum and dad, you shaped my personality and attitude more than anybody else and made me the person I am today. Thank you so much for always guiding me into the right direction. The previous years were not the easiest ones for my girlfriend Magdalena. Often little spare time gave our relationship a hurdle to be overcome. We managed to do so and I am looking forward to a common future with her.

Abstract

The development and fabrication of secure, trustworthy, and efficient technical devices becomes an increasingly difficult task because of the high complexity of today's systems. Dedicated hardware solutions based on Application-Specific Integrated Circuits (ASICs) or Field-Programmable Gate Arrays (FPGAs) are generally preferred over their software counterparts to achieve ambitious throughput, power, or energy goals. Similar to their software equivalents, these hardware-oriented approaches suffer from several vulnerabilities that might be exploited by an attacker during the life cycle of a device.

As part of this thesis, we investigate risks and performance aspects, immanent in the development and fabrication of any VLSI-based symmetric encryption device. We consider ASICs and FPGAs as the lowest hierarchy level where an attacker may intrude the system. Therefore, we implant a hardware Trojan into an ASIC just prior to fabrication in a 180 nm CMOS technology by UMC. Both the genuine and the malicious design have actually been manufactured. Subsequently, we apply Trojan detection techniques based on side-channel fingerprinting. Despite the comparatively small size of the Trojan (0.5 % of the original design), we successfully distinguish malicious from genuine ASICs.

Since FPGAs are of particular interest for high-throughput designs of cryptographic algorithms, we also analyze Trojans on reconfigurable hardware. More specifically, we insert a malicious circuitry into a bitstream after placement and routing. Thereafter, we show how to use electromagnetic radiation as a side-channel to successfully detect the Trojan and experimentally demonstrate a method to actually localize it on the FPGA.

Symmetric encryption devices need to share a common cipherkey prior to their communication. This exchange is often accomplished with the use of small hardware tokens like USB sticks or smart cards. Because such items are widely accessible by the general public, they need to be secured against implementation attacks like Differential Power Analysis (DPA). For that reason, we have developed and manufactured ZORRO, an ASIC to assess DPA countermeasures on a real chip. Based on measurements acquired from the fabricated ASIC, we show that 100 000 traces or less are sufficient to successfully attack ZORRO with standard DPA. ZORRO does not merely constitute an evaluation platform for DPA countermeasures, but also represents the smallest, DPA-secured ASIC implementation of a KECCAK-based Authenticated Encryption (AE) scheme available to date.

Eventually, we analyze the efficiency in terms of throughput-per-area of emerging AE algorithms. We aim at ASIC architectures with throughputs of 100 Gbit/s and even beyond. To do so, we create a GCM-AES reference architecture, targeting a 65 nm CMOS technology by UMC. This design serves as a basis for a comparison between several candidates from the CAESAR competition. We show that all of the CAESAR algorithms investigated outperform GCM-AES when looking at the asymptotic use case. However, we point out that when more realistic scenarios are considered, for instance, communication protocols like Ethernet, this advantage diminishes substantially.

Zusammenfassung

Die Komplexität technischer Produkte ist in den vergangenen Jahren kontinuierlich gewachsen. Immer mehr Software- und Hardwarekomponenten werden zu grösseren Gesamtsystemen kombiniert um den Ansprüchen der Kunden gerecht zu werden. Aus diesem Grund wird die Entwicklung und Herstellung von hochperformanten sowie gleichzeitig als sicher und vertrauenswürdig geltenden Geräten stets schwieriger. Um den Anforderungen nach hohem Durchsatz und geringem Energieverbrauch nachkommen zu können, werden oftmals hardwarebasierte Lösungen ihren Software-Alternativen vorgezogen. Sowohl anwendungsspezifische integrierte Schaltungen (ASICs) als auch programmierbare Hardwarekomponenten (FPGAs) werden dafür häufig eingesetzt. Leider können im Zuge der Entwicklung und Fabrikation dieser Komponenten zahlreiche Schwachstellen dazu genützt werden, um die Sicherheit der Geräte zu unterwandern.

Die vorliegende Arbeit behandelt Risiken und Performanceaspekte, die während des Lebenszyklus eines VLSI-basierten symmetrischen Verschlüsselungsgerätes nicht vernachlässigt werden dürfen. ASICs und FPGAs werden dabei als unterste Hierarchieebene eines solchen Produktes angesehen. Aus diesem Grund untersuchen wir zunächst Hardware Trojaner. Dazu wurde ein bösartiger Schaltkreis in ein existierendes ASIC Design eingeschleust. Der unverfälschte sowie der leicht abgeänderte Chip wurden in einer 180 nm CMOS Technologie fabriziert und im Anschluss daran mittels Seitenkanalanalysen untersucht. Obgleich der implantierte Trojaner nur 0.5 % des ursprünglichen Designs ausmachte, konnten wir diesen zuverlässig detektieren.

Zur Umsetzung von kryptografischen Algorithmen in hochperformanten Anwendungen werden immer häufiger FPGAs verwendet. Daher vertieften wir unsere Trojaner-Untersuchungen in diese Richtung und implantierten einen bösartigen Schaltkreis in eine existierende FPGA Konfiguration. Unter Verwendung der elektromagnetischen Abstrahlung als Seitenkanal konnten wir zwischen bösartigen und unverfälschten Varianten eindeutig unterscheiden. Zusätzlich zur Detektion stellen wir eine Methode zur Lokalisation von Trojanern vor.

Für den Austausch von Schlüsselmaterial zwischen zwei Parteien werden oftmals USB Sticks oder Smartcards eingesetzt. Diese sind aufgrund ihres Einsatzbereiches meist relativ einfach einer breiteren Masse zugänglich und müssen daher gegen Implementierungsangriffe wie die differentielle Leistungsanalyse (DPA) geschützt werden. Hierzu präsentieren wir ZORRO, einen ASIC der ausschliesslich zur Beurteilung der Qualität von DPA Gegenmassnahmen entwickelt wurde. Basierend auf Messergebnissen zeigen wir, dass Techniken wie *Hiding* und *Masking* alleine keinen Angriffen auf unseren Chip mit bis zu 100 000 Stromverbrauchsprofilen vorbeugen können. Des Weiteren repräsentiert ZORRO die kleinste verfügbare, DPA-sichere ASIC-Realisierung eines KECCAK-basierten *Authenticated Encryption* Systems.

Abschliessend untersuchen wir die Effizienz hinsichtlich Durchsatz pro Fläche für aufstrebende kryptografische Algorithmen, welche sowohl Vertraulichkeit als auch Integrität gewährleisten. Dazu präsentieren wir zunächst eine GCM-AES Referenzarchitektur basierend auf einer 65 nm CMOS Technologie für Durchsätze von 100 Gbit/s und mehr. Im Anschluss daran werden hochperformante Designs einiger Kandidaten der zweiten Runde des CAESAR Wettbewerbs vorgestellt und mit der GCM-AES Architektur verglichen. Wir zeigen, dass alle untersuchten Mittbewerber deutlich effizienter in Hardware umzusetzen sind als der Referenzalgorithmus solange der asymptotische Durchsatz betrachtet wird. Werden die entwickelten Architekturen hingegen für Kommunikationsprotokolle wie Ethernet verwendet, so relativiert sich deren Vorsprung gegenüber GCM signifikant.

Contents

1 Introduction

The complexity of today's applications in the field of Information and Communications Technology (ICT) increases continuously. Throughout a system's development and fabrication life cycle, typically software as well as hardware components from multiple parties are combined to fulfill the desired application requirements. As a result due to the high complexity of these systems, assuring a certain level of security has become a costly and non-trivial task. A major decision, which must often be taken very early in the development phase of an application is, whether the targeted functionality should be achieved using a software- or hardware-centric approach. The main aspects to be considered when taking such a fundamental decision are illustrated in Figure 1.1 and can be summarized as follows:

Costs: Like for many other applications, minimizing costs is often of highest priority throughout the development of a secure application. Although this aspect should only be secondary for security systems, it is often neglected as a result of bowing to pressure of management and marketing demands. Since devel-

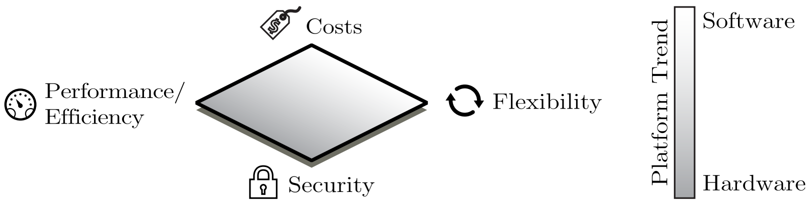


Figure 1.1: Aspects to be considered when deciding among hardware- or software-centric approaches for security applications and the trend for the corresponding implementation platform.

oping hardware implementations requires substantially more effort compared to their software counterparts, it is often not even discussed whether software or hardware approaches should be favored. However, as briefly discussed in Section 1.1, developing *actually secure* software can become way more expensive than initially thought.

Flexibility: While software in the testing or deployment phase can be updated at will, there is no (easy) way to apply changes to an Application-Specific Integrated Circuit (ASIC) after tape-out. Although today's Field-Programmable Gate Arrays (FPGAs) provide more flexibility by using electrical reconfiguration, they are inferior to software implementations in terms of flexibility. Hence, if frequent updates are necessary for a system, a software-centric approach is favored in most cases.

Performance/Efficiency: As for performance and efficiency, hardware solutions are typically employed for two fields of application. First, when high performance in terms of *throughput* is of utmost importance, ASIC or FPGA systems significantly stay ahead of software alternatives. Second, if *peak power* or *energy consumption* must be considered as limiting factors, dedicated hardware implementations are usually employed. The efficiency achievable with ASIC implementations can hardly ever be reached with their software counterparts hosted on Commercials Off-The-Shelf (COTS).

Security: The last aspect, which often gets neglected during development, is the actually targeted security. The origin of secure software is typically a so-called *trust anchor*, from which trust gets inherited over several hierarchy levels. Trust anchors are often based on small hardware components, such as *Trusted Platform Modules (TPMs)*. From them, the trust is usually derived all the way through drivers, firmwares, and the operating system to a certain user application. Consequently, if security is one of the major goals, hardware-near solutions are more and more often favored since there is no need to trust the complete software stack.

Taking into account the four aspects mentioned above, a rough guideline can be that the more important performance, efficiency, and security are, the more hardware-near the platform should be (cf. Figure 1.1). Still, costs and flexibility should not be neglected completely in such a decision.

1.1 The Need for Hardware-Based Security Systems

General-purpose processors have evolved significantly over the last years. Because of the technology scaling of CMOS processes used for fabrication, the number of transistors has grown continuously according to Moore's Law [77] in the past. Due to the ever increasing integration density and the rising importance of security aspects, chip manufacturers have started to incorporate dedicated instructions for encryption operations. Most notable is the AES New Instruction Set (AES-NI), proposed by Intel in 2008 [50] and introduced into their Westmere processors in 2010. As a result, software implementations of security applications using such hardware-accelerated platforms already reach a high level of efficiency compared to their dedicated hardware counterparts, and this trend is to continue. Hence, the question arises for what type of applications is it actually still justifiable to build a hardware-centric rather than a software-based system.

As outlined in the previous section, performance, efficiency, and security are the main driving forces behind hardware developments. For smart cards or Radio-Frequency Identification (RFID) systems, as well as applications in the field of resource-constrained environments in general, the circuit complexity and the peak power or the energy consumed are often metrics of paramount importance. Since for such applications very high volumes are usually required, ASIC implementations are chosen most of the time. On the other end of the efficiency range, high-performance applications targeting throughputs of 100 Gbit/s and beyond can be found. For these systems circuit complexity, power, and energy are often secondary and are mostly only considered after reaching a certain throughput goal. Since the volume for these types of applications is sometimes significantly smaller than for resource-constrained environments, FPGA platforms are often considered as an alternative to ASIC implementations. Due to their reconfigurability, FPGAs offer a good trade-off between the high efficiency provided by hardware components and the flexibility of software. However, they typically lag behind their ASIC counterparts in terms of efficiency by approximately an order of magnitude [36].[1] In general, despite the fact that more and more security instructions are included into general-purpose processors, the performance and the efficiency of hardware implementations are still two of their major selling points. Especially for non-standard applications where efficiency is of high importance, dedicated hardware solutions should be favored.

Security and trust, on the other hand, are of growing importance for system designers. While hardware engineers can influence the resulting CMOS circuitry down to the gate level or even to single transistors, software developers are forced to use general-purpose processors, Graphics Processing Units (GPUs), or other COTS to realize their security application. As a result, the greater flexibility of the platform must be paid with costs for verifying its security, which are often not considered during design decisions. Although difficult to quantify in numbers, an industry rule of thumb exists, saying that to reach a security level of EAL6 costs about $1000 per line of code [67]. It is

[1]Recall that an FPGA is a VLSI chip by itself and thus, potential comparisons always depend on the technology the FPGA was fabricated in.

important to note that hardware architectures are not free from implementation errors which occur during the development of the Hardware Description Language (HDL) code of the circuit. However, all the hierarchy levels (e.g., the operating system, drivers, or third-party software libraries) lying in between a software implementation and the actual hardware executing the respective software must not be considered as potential weaknesses. Nevertheless, there are still quite a number of weaknesses in the life cycle of hardware-based systems, which might be the target of a potential attacker. Those weaknesses, potential countermeasures to get around them, as well as highly efficient VLSI implementations thereof are the main topic of the present thesis.

1.2 Cryptography Basics

The requirements of a security system can usually be broken down to a subset of the following basic cryptographic goals:

Confidentiality: Only the parties designated to participate in a certain communication should be able to read the transferred data. Unauthorized parties must not be able to read them.

Integrity: As soon as the messages being sent within a cryptographic system get modified, the participating parties should be able to recognize the modification.

Authenticity: This goal is sometimes further split up into *entity* and *data authentication.* Entity authentication refers to the service that the parties communicating with each other convince the other participants of their identity. Data authentication means that when one party receives a message from another party, the receiver must be able to verify that the message indeed originates from the supposed sender.

Non-Repudiation: As soon as a party sends a message to another party, the sender is not able to repudiate that the sent message does not originate from him.

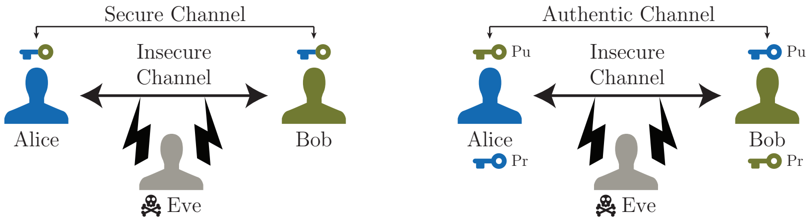


Figure 1.2: *Left:* Symmetric-key encryption scheme using a single shared secret cipherkey; *Right:* Public-key encryption scheme using pairs of private (Pr) and public (Pu) cipherkeys

Security by obscurity is often (mistakenly) applied in order to achieve one of the above mentioned goals or simply to increase the overall security of a system. However, as early as in the nineteenth century A. Kerckhoffs came up with a very fundamental principle, which should still be followed during the design of cryptographic applications these days:

Definition 1.1 (Kerckhoffs' Principle)**.** *A system's security should only rely on the secrecy of the key being used. Hence, even if an attacker has a copy of the system, it should still be secure as long as he does not have access to the utilized key.*

Throughout the last couple of centuries, two major types of security schemes have emerged to provide secure communication. First, *symmetric-key* approaches are based on the fact that the communicating parties share a common secret cipherkey, which is then used to assure services such as confidentiality and/or authenticity. *Public-key* systems, on the other hand, are based on a pair of keys for each participating party, one of which is kept secret (the private key) and the other one is made publicly available for everybody (the public key). Figure 1.2 illustrates both schemes with the two parties *Alice* and *Bob*, communicating over an insecure channel. This transmission might be intercepted by an attacker (*Eve*). Thanks to shorter keys, symmetric key approaches can achieve much higher efficiencies compared to their public-key counterparts. However, since for any communication pair a shared secret must exist, symmetric-key approaches suffer from the problem of a more complicated key management. Also the distribution of the keys via an authentic communication channel prior to the actual data transmission becomes a challenging task.

Despite the key management challenges, when it comes to data-intensive communication systems these days, the public-key approach is mainly used during the setup phase of the transmission (e.g., to exchange a symmetric key). For the subsequent data transfer, however, symmetric-key algorithms are typically employed.

1.3 Goals of the Thesis

Due to the ever increasing complexity of systems in the field of ICT, it becomes more and more difficult (if not impossible) to provide *secure* applications. Although this thesis solely deals with vulnerabilities stemming from the hardware components of a system, it turns out to be alarming how sophisticated and powerful state-of-the-art attack scenarios are. This work was accomplished as part of a collaboration between industry and academia. The industrial party involved in the project was *Omnisec AG*, a Swiss company that provides hardware-accelerated symmetric encryption solutions for governmental as well as defense markets. Therefore, this thesis deals with security issues to be considered throughout the development of VLSI-based encryption devices. Due to confidentiality issues, we could not work with the algorithms currently in operation at the industry partner. Hence, for each of the investigated attack scenarios we tried to find appropriate replacement candidates, which are expected to be of interest for both academia as well as industry. The major research questions of this thesis can be summarized as follows:

Question 1.1. *How dangerous are hardware Trojans for VLSI-based devices and how practical are they?*

ASICs and FPGAs are often used as core components of security-critical applications these days. Therefore, their integrity with regard to fabrication (ASICs) and configuration (FPGAs) must be assured thoroughly. The term *hardware Trojan* has shaped parts of the security research community during several years now. Nevertheless, practical examples thereof are still rare and thus, detection methods are mostly evaluated using simulations or on FPGA platforms. We aim at answering this question by simulating an attack at the ASIC development chain, thereby tackling the problem of untrustworthy foundries. With the use of non-destructive detection techniques, we analyze our ASICs and try to distinguish genuine from malicious samples. Moreover, we do not only focus on the detection of hardware Trojans, but also provide first suggestions on how to actually localize unintended malicious logic components.

Question 1.2. *Are state-of-the-art Differential Power Analysis (DPA) countermeasures of embedded devices ready to withstand sophisticated attack scenarios, and what is the price we have to pay for them?*

Symmetric encryption devices suffer from the problem of key distribution. Unless cipherkeys are distributed among the communicating parties electronically with the use of a public-key infrastructure, physical devices such as smart cards or similar hardware tokens are used. With the adoption of *keyed devices*[2], which are often more easily available to the general public, implementation attacks[3] become a major threat. DPA has emerged as one of the most powerful methods to reveal secret internals based on the leakage of side-channel information. With the use of an emerging Authenticated Encryption (AE) system, we aim at answering the question how well DPA countermeasures like *hiding* or *masking* work on an actually fabricated ASIC. Moreover, we want to compare those countermeasures with regard to their resource requirements needed to achieve the DPA protection.

Question 1.3. *Can future AE algorithms keep up with today's standards in terms of their hardware efficiency from a VLSI perspective?*

The Galois/Counter Mode of Operation (GCM) using the Advanced Encryption Standard (AES) as the underlying block cipher can be considered as one of the de-facto AE standards available in today's security applications. The *Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR)* aims at

[2]By *keyed devices* we refer to hardware components that process any kind of sensitive data such as secret keys or plaintexts.

[3]*Implementation attacks* are attacks that do not target an algorithm by itself, but the actual implementation of it on a certain device.

finding potential alternatives or even successors. Our goal is it to seek for the most promising candidates of the competition to be used in high-performance VLSI architectures. We do not only investigate the candidates regarding their asymptotic performance[4], but use state-of-the-art communication protocols to analyze them under real-world conditions. For that, packet size distributions of protocols from different OSI layers are considered, including Ethernet and TCP.

1.4 Contributions

The main goal of this thesis is to investigate potential attack scenarios for symmetric AE systems from a hardware developer's point of view. Additionally, its focus lies on designing high-speed VLSI architectures of potential future AE systems. Therefore, several algorithms from the CAESAR competition are studied with regard to their hardware performance, targeting throughputs of 100 Gbit/s and beyond. Summarizing, the major contributions of this work are as follows:

- **Hardware Trojan detection on an actual ASIC:** Detection of Trojans in the field of VLSI designs has previously either been conducted with FPGA platforms or based on simulations. The main reason for this is that a genuine and a malicious ASIC are not normally available for analyses purposes. Our taped-out chips, called *Chameleon* and *Chipit*, represent one of the very rare known such pairs of chips. Both contain an AES-based cryptographic primitive. While Chameleon solely hosts the genuine design, in Chipit a Denial-of-Service (DoS) hardware Trojan has been implanted in the interface of the chip. Insertion of the malicious circuitry was accomplished solely on the layout data of the design, assuming no special knowledge about the actual internals of the genuine architecture.

  Using several Side-Channel Analysis (SCA) techniques based on dissipated power, we demonstrate that despite the very small size of the Trojan, occupying less than 0.5 % of the area of the

[4]By *asymptotic performance* we refer to the use case where very long input messages are available to be processed.

genuine design, its detection is actually feasible with a comparatively simple measurement setup. The presented ASICs have already been acquired by other research institutions [38] to study hardware Trojan detection techniques other than SCA fingerprinting based on power.

- **Localization of FPGA Trojans using Electromagnetic Radiation (EM):** FPGAs are another potential target for the implantation of malicious circuitries. Related work from previous years [79, 80, 81] has shown that the bitstream encryption process of FPGA vendors cannot be considered to be perfectly secure. As a result, the configuration stream for an FPGA might be altered by an attacker unnoticed to insert additional, malicious circuitry in the FPGA.

  Based on a post-placement insertion flow, we propose to use the picked-up electromagnetic radiation not only for distinguishing malicious from non-malicious configurations, but also to actually localize the Trojan on the device. By inserting the same DoS Trojan at different locations on the FPGA, we demonstrate that detection is independent from the actual location of the Trojan within the configurable logic. Although we successfully identified the malicious designs using EM fingerprinting on a Xilinx FPGA, we fell short of pinpointing the exact location of the Trojan logic by stepping over the packaged Integrated Circuit (IC) with a near-field probe.

- **ASIC-based DPA countermeasures evaluation platform:** DPA can be considered to be one of the major analysis methods that has lead to numerous practical attack scenarios on secret-key-processing cryptographic hardware devices in the recent past.

  As a result, we have developed Zorro, an ASIC serving as an evaluation platform for DPA countermeasures. It contains three independent, permutation-based AE primitives using Keccak-*f* as the underlying permutation, which differ with regard to the utilized masking scheme. All designs provide both hiding as well as masking countermeasures that can be enabled or disabled at will. Our ASIC has been developed for applications in the field of resource-constrained environments. Hence, low circuit complexity was one of our main design goals. We demonstrate that neither hiding nor masking alone are sufficient as DPA countermeasures to withstand attacks on ZORRO with up to 100 000 traces.

- **High-performance hardware designs of AE algorithms:** GCM-AES represents a de-facto standard when it comes to AE algorithms with Associated Data (AD). The CAESAR competition aims at finding a potential portfolio of successors or at least alternatives to GCM-AES, since the latter has suffered from several security flaws in the past [69, 29, 113, 32, 96, 42] and also a need for more efficient algorithms emerges.

  We contribute to the competition by analyzing several of the second-round CAESAR candidates with regard to their suitability for high-throughput hardware designs. More specifically, we target applications with data rates of 100 Gbit/s and even beyond, which are expected to play a major role in the upcoming years. We compare those competitors against a GCM-AES reference design as well as against each other under two different use cases in terms of the circuit complexity required to reach the target throughput of at least 100 Gbit/s. The two scenarios in question are the so-called *data at rest* scenario and the *data in motion* use case, for which we use Ethernet as an example protocol. Based on a mature 65 nm ASIC technology, we show that although many of the competitors result in better hardware efficiency for the data at rest use case, most of their advantages significantly diminish when it comes to more practical data in motion scenarios like Ethernet.

1.5 Outline

This thesis can be subdivided into three main chapters, each of which gets identified by an icon. Each such icon refers to one phase in the life cycle of a VLSI-based encryption device, where the respective attack scenarios or performance metrics need to be considered.

Manufacturing. In Chapter 2, we investigate hardware Trojans for both ASICs and FPGAs. After some background information on hardware Trojans in general, we first present an ASIC Trojan, which we implanted into a genuine AES architecture on the mask level. With the use of side-channel fingerprinting techniques we then classify the actually fabricated genuine and malicious chips both with and without the use of a golden IC. Moreover, we propose to use EM not only for Trojan detection, but also for the localization of the hidden logic based on Trojans implanted into already placed-and-routed FPGA configurations.

Operation. Implementation attacks and their countermeasures, which must be considered for any keyed cryptographic primitive, are covered throughout Chapter 3. In order to analyze DPA countermeasures on an actual ASIC, we introduce our Zorro chip, providing hiding and masking techniques that can be enabled or disabled at will. Next, we present the hardware figures of this smallest, DPA-secured ASIC architecture of a keyed, Keccak-based AE scheme available to date. Using an appropriate measurement setup, we then compare the attack resistance of the different countermeasures.

Performance. Chapter 4 deals with current and potential future AE algorithms to be used in high-performance ASIC architectures. Since these days, GCM-AES represents one of the favored primitives in this domain, we first present a reference architecture reaching an asymptotic throughput of at least 100 Gbit/s based on a mature 65 nm CMOS technology. Next, we provide high-performance VLSI designs of several of the participants of the CAESAR competition and compare them against our GCM-AES reference design. We also analyze the resulting performance of the investigated candidates under real-world conditions using state-of-the-art communication protocols such as Ethernet or TCP.

Finally, in Chapter 5 we draw overall conclusions about the topics treated in this thesis and give suggestions for potential future work.

2 Real-World Hardware Trojans

Outline. We begin this chapter with a general introduction about hardware Trojans in Section 2.1. Throughout Section 2.2, we present one of the very rare, actually taped-out ASIC Trojans. We discuss how we inserted the Trojan on the mask layer, thereby tackling the so-called *untrustworthy manufacturer problem*. Moreover, we provide details about our detection process based on side-channel fingerprinting. In Section 2.3, we investigate the detection of FPGA Trojans using Electromagnetic Radiation (EM) as a side channel. For that we insert a Trojan into a placed-and-routed FPGA bitstream. We propose to use EM not only for distinguishing malicious and genuine FPGA configurations, but also to actually localize the Trojan circuitry within the FPGA. After all, we provide final remarks about our hardware Trojan analysis in Section 2.4.

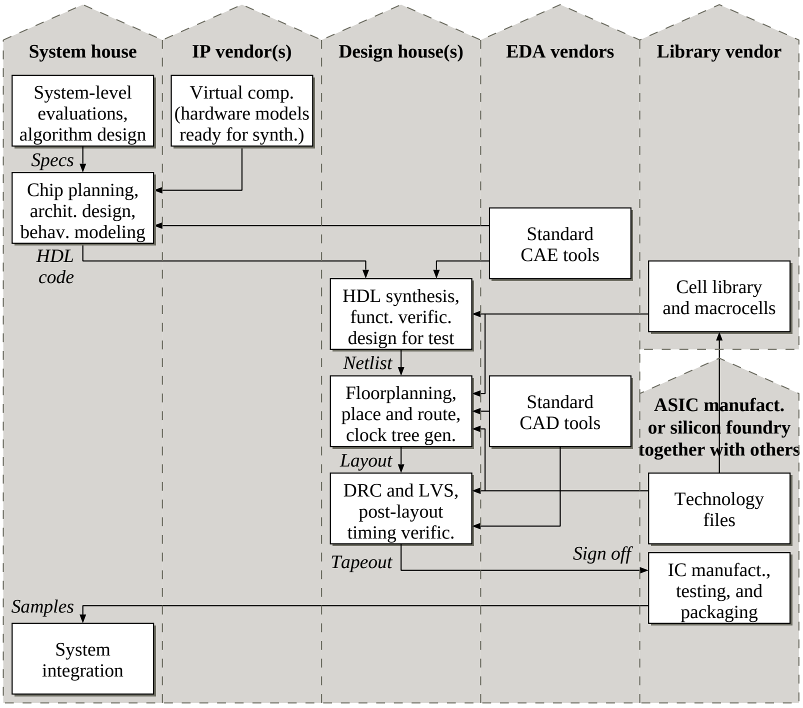

System house
IP vendor(s)
Design house(s)
EDA vendors
Library vendor
System-level evaluations, algorithm design
Virtual comp. (hardware models ready for synth.)
Specs
Chip planning, archit. design, behav. modeling
Standard CAE tools
HDL code
HDL synthesis, funct. verific. design for test
Cell library and macrocells
Netlist
Floorplanning, place and route, clock tree gen.
Standard CAD tools
ASIC manufact. or silicon foundry together with others
Layout
DRC and LVS, post-layout timing verific.
Technology files
Tapeout
Sign off
IC manufact., testing, and packaging
Samples
System integration

Figure 2.1: Parties involved and their responsibilities of a cell-based, full-custom ASIC development (illustration adapted from [63]).

2.1 Background

Throughout the last decade, trustworthiness in hardware components got more and more important as these components must provide a reliable basis for their software counterparts [40]. Development and fabrication of modern VLSI circuits for both ASIC and FPGA platforms rely on an ever increasing set of active participants to manage both costs and complexity. Figure 2.1 illustrates the parties involved and their responsibilities of a typical cell-based, full-custom ASIC development. More often than not, large parts of the design are outsourced to specialist teams all around the world, pre-designed Intellectual Property (IP) blocks from different vendors are added to the design, and complex Electronic Design Automation (EDA) software is used in the design flow for synthesis and analysis purposes. Finally, in the occurrence of ASIC development, a specialized semiconductor foundry is responsible for the actual manufacturing of the IC, including wafer processing, packaging, and testing.

ASIC and FPGA fabrication are based on the same CMOS manufacturing process. This process must be considered as a point of attack where a potential adversary may intrude the life cycle of a security-critical hardware device. The complex fabrication of ASICs and FPGAs provides a skilled adversary various possibilities for inserting a (relatively small) circuit that can remain undetected during the design and verification process. Such a malicious hardware circuitry—also known as a *hardware Trojan* [41]—would then be manufactured together with the actual circuit and can be used for different purposes, such as:

Altering the specification or reliability: Trojans may slightly change the intended functionality of a circuit or compromise the performance by altering the physical characteristics of a design (e.g., narrowing certain wires to accelerate aging).

Denial-of-Service (DoS) attacks: A malicious circuit causing a complete failure of a design is referred to as a *DoS Trojan*.

Leaking information: If a design is intended to process any kind of sensitive data such as secret keys, a Trojan may leak this information, for instance, over some sort of side channel.

For additional information about the effects of hardware Trojans, we refer the reader to [108]. Since the complexity of ASICs continues to increase driven by competitive market pressure, more and more opportunities for an attacker to insert a Trojan emerge. This is due to the following factors:

- While IT circuits continue to grow in complexity, the size of hardware Trojans does not necessarily increase proportionally. Therefore, detecting an unwanted circuit smaller than, say, one millionth of the original circuit becomes a challenging task.

- Basically, the high costs for developing an ASIC can be justified in two situations. On the one hand for systems where the absolute performance of speed, area, energy efficiency, or security is of paramount importance. On the other hand for very high-volume fabrication, where millions of ASICs are to be sold. Both cases are attractive targets for adversaries, as ASICs often constitute either the most important part of a system or are widely deployed and thereby compromise a large number of systems.

- As the expected harm caused by a compromised device is very high, adversaries can afford to invest significant sums into designing and inserting hardware Trojans.

At the same time, ASIC designers face considerable challenges to verify the correct functionality of their own (genuine) circuits properly. Even the most modern verification flows do not stand much chance to detect hardware Trojans inserted by determined adversaries. That is why hardware Trojan detection has attracted significant interest in recent years.

Trojan detection techniques: In general, Trojan detection approaches can be classified into destructive and non-destructive methods as depicted in Figure 2.2. Destructive techniques include depackaging, delayering, and various mechanical and/or chemical steps. Eventually, the exposed layers of an ASIC are visually compared against the original Graphic Database System II (GDSII) layout data to find any malicious alterations. However, these techniques are very time-consuming and rather expensive, especially for modern manufacturing processes [112]. Hence, for small- and mid-sized hardware companies, destructive techniques are hardly affordable.

Within the group of non-destructive techniques, one can differentiate between detection approaches carried out during runtime or as part of testing. Both runtime monitoring and logic testing, which belongs to the test-time approaches, are referred to as *invasive* methods, since they require additional test circuitry to detect a Trojan. Side-channel detection methods [6], on the other hand, are based on building a fingerprint using physical characteristics of a Circuit Un-

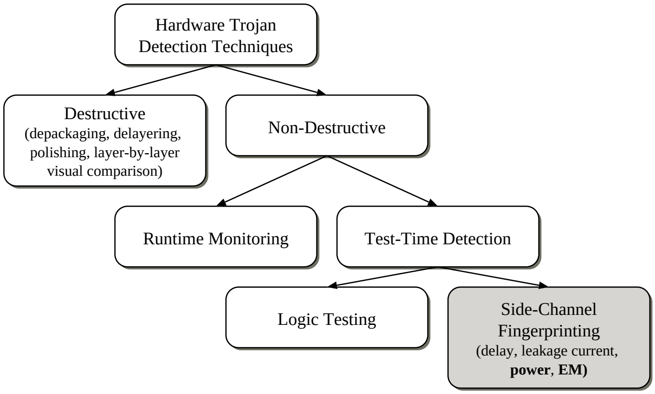

Hardware Trojan Detection Techniques
Destructive (depackaging, delayering, polishing, layer-by-layer visual comparison)
Non-Destructive
Runtime Monitoring
Test-Time Detection
Logic Testing
Side-Channel Fingerprinting (delay, leakage current, **power**, **EM)**

Figure 2.2: Taxonomy of hardware Trojan detection techniques [109].

der Test (CUT). Typical side channels used for this approach are delay [73, 61], leakage current [3], (dynamic) power, or Electromagnetic Radiation (EM) [6]. Moreover, Narasimhan et al. [89] proposed to combine different side channels in order to improve the detection ratio. In our experiments, we focus on power and EM as side channels to detect the ASIC and FPGA Trojans, respectively.

Design type notation: During the remainder of this chapter, we refer to the different designs of actually fabricated ASICs or FPGA configurations using the following notation.

Original design: The pristine design as intended by the system house is referred to as the *original* circuit.

Circuit Under Test (CUT): After the fabrication of an ASIC, the integrity of the circuit needs to be verified. The same applies following the configuration of an FPGA. At this stage, we denote the designs as *Circuits Under Test (CUTs).*

Genuine design: Once a CUT is proven to be Trojan-free, we refer to it as being *genuine.*

Malicious design: If a CUT contains a Trojan, it is considered to be *malicious.*

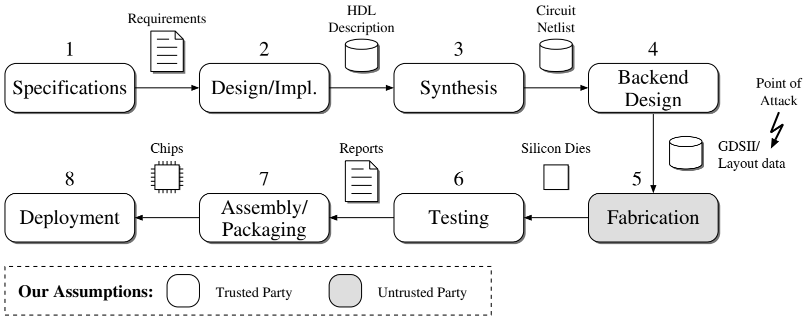


Figure 2.3: Simplified overview of the design phases of today's ASIC development chains and their respective outputs and our assumptions regarding their trustworthiness.

2.2 Red Team vs. Blue Team ASIC Trojan Analysis

Despite the huge amount of related work carried out on hardware Trojans and their detection, most work has been accomplished based on FPGA platforms or ASIC simulation results only. Therefore, we decided to set up an experiment similar to the untrustworthy manufacturer problem, putting ourselves into the role of a fabless customer and an adversarial foundry. Compared to previous experiments, our approach tries to answer the question if Trojan insertion during fabrication and subsequent detection can be carried out successfully for a real-world design under the restrictive time schedule of the ASIC manufacturing process.

2.2.1 ASIC Development Chain

Assume a simplified ASIC development chain as depicted in Figure 2.3. After initially determining the specifications of a design, it is usually captured using a Hardware Description Language (HDL). Next, the HDL description gets synthesized into a netlist of standard cells. The subsequent step is to generate the physical mask layout that will be used as a blueprint for manufacturing the IC. Finally, wafer processing takes place, eventually resulting in individual dies that are tested, packaged, and delivered to the customer.

An adversary may target any of the design stages outlined in Figure 2.3. From stage 2 to stage 4, the technical requirements continue to increase and smaller companies usually need to outsource portions of the design flow to more specialized enterprises. Larger firms outsource at least the actual IC manufacturing to one of the very few semiconductor foundries. The last step at which a hardware Trojan can be inserted into the ASIC is just prior to *Fabrication*. Assume an attacker, who is not part of the design team, intercepts the development of an ASIC at this stage. Usually, such an adversary knows very few technical details about the target design and will have only limited time to analyze the target circuit and to insert the Trojan. Otherwise, the disruption of the tight manufacturing schedule will raise suspicion. However, if inserted at this late stage, there will be virtually no opportunity for the system house (i.e., the party commissioning the ASIC) to detect the Trojan before measuring the fabricated samples.

For our investigations, we considered fabrication to be the only untrustworthy stage. We believe that it is one of the most vulnerable phases in an ASIC's life cycle, since many people from third parties are involved. Moreover, monitoring the whole fabrication phase, including mask generation and wafer processing, is too expensive for most fabless hardware firms.

2.2.2 Experimental Setting

To tackle the so-called *untrustworthy manufacturer problem*, we initiated an experiment. We tried to imitate an attack as realistically as possible and therefore, set up a *red team vs. blue team* approach based on the following two teams:

1. Blue team: In our experiment, the blue team acts as the *blue-eyed* system house, aiming at building an ASIC, expecting the following assumptions and responsibilities:

- As a first step, the blue team develops the original design all the way down to the detailed layout (in GDSII format), which they send to the foundry for fabrication.

- Since the blue team is a fabless system house, they have to trust several third parties to get their design manufactured.

- Once the blue team gets back their chips, they are responsible for devising an adequate measurement setup to analyze the ICs using a non-invasive, side-channel-based analysis approach.

- Using certain post-processing techniques, this team should build hypotheses regarding which ASICs represent genuine designs and which contain malicious circuitries, respectively.

2. Red team: The red team, on the other hand, assumes the role of the adversarial party located at the foundry. It is expected to fulfill the assumptions and responsibilities given below:

- The red team is assumed to only have access to the mask-layout (data typically in GDSII format) sent for fabrication by the blue team.

- We do not expect the red team to have any detailed knowledge about the target design, since such knowledge would assume *insider information* from the blue team.

- Although such information can be reverse engineered by a sophisticated attacker, it is usually a very time consuming task. Since the fabrication process of an ASIC normally already has a very tight schedule, any additional delays may raise undesirable suspicions.

- The red team is responsible for the development of the Trojan circuit. As target-independent hardware Trojans can be designed prior to an actual attack, their development does not cause any major delay.

Table 2.1: Key properties of Chameleon, the target ASIC for the Trojan insertion.

Property	
CMOS technology	180 nm
Supply voltage (core/pads)	1.8/3.3 V
Core area	38 000 GE
Maximum frequency (f_{max})	125 MHz
Latencies (incl. I/O interface)	
AES encryption[1]	945 cycles
AES decryption[1]	1558 cycles
Grøstl hashing[1]	3465 cycles

[1] Applies for both standalone and GrÆStl version.

- Finally, the red team actually inserts the Trojan into the target design. Due to the tight schedule of an ASIC fabrication, we limit the amount of time for the insertion to a handful of working days.

2.2.3 Chameleon - The Target Circuit

For our investigations, we could have chosen any existing ASIC design with a well-defined interface, regardless of its functionality. We had access to a number of designs that were due to be submitted for fabrication. Our victim circuit, called *Chameleon* [97], was designed independently prior to this work and includes the following three cryptographic cores[1] that share peripheral circuits and a common top-level controller:

1. The first core is a standalone, low-area implementation of the Advanced Encryption Standard (AES) [94], targeting applications in resource-constrained environments.

[1]Choosing a cryptographic design has made some of the choices in the project simpler. However, our approach is generic enough, and we are confident that we would have achieved similar results with most other designs as well.

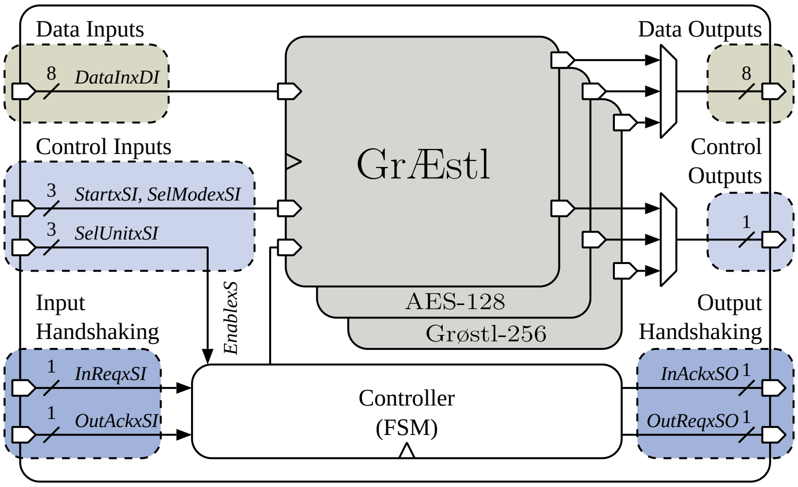


Figure 2.4: Chameleon top-level hierarchy, showing independent designs and interfaces; clock and reset signals are omitted

2. The second core implements an iterative version of the cryptographic hash function Grøstl [46], which was one of the finalists of the SHA-3 hash competition [93] organized by the National Institute of Standards and Technology (NIST).

3. Eventually, the third design on Chameleon is called *GrÆstl* and combines both AES and Grøstl in a single optimized datapath. Since the two cryptographic primitives share similar core components, the resulting GrÆstl architecture provides both a block cipher and a hash algorithm with only very little overhead in terms of silicon area compared to the standalone versions.

All three cores are completely independent of each other. The chip is configured to run only one of the cores at a time. Key data of the Chameleon implementation are given in Table 2.1.

Figure 2.4 shows the top-level architecture of Chameleon, which communicates with its environment based on an eight bit wide I/O interface. A four-phase handshaking protocol was implemented to control the data flow. The signals `InReqxSI`, `InAckxSO` and `OutReqxSO`,

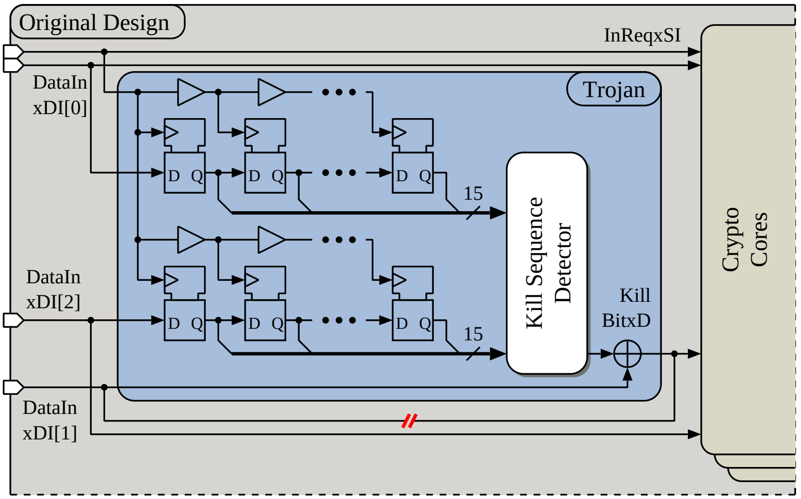


Figure 2.5: The structure of the DoS hardware Trojan and how it got implanted into the original design. The combinational logic for the kill sequence comparator consists of about 10 logic gates.

`OutAckxSI` control input and output handshaking, respectively. For Trojan insertion, the red team used no design-specific information other than the four-phase handshaking signals as will be discussed in the following subsection. Strictly speaking, this information is not available from layout data. Yet we believe that it is straightforward to extract this information, either through cursory examination of the circuit or through ancillary data such as pin names or from the supporting documentation.

2.2.4 Trojan Circuit

The goal of the red team was to design a Trojan difficult to detect. This called for an extremely small circuit. Hence, we decided to integrate the DoS Trojan illustrated in Figure 2.5. The Trojan observes certain inputs of the original circuit and waits for a specific *kill sequence*. Once detected, another input bit of the original circuit is flipped, which causes functional failure in the following clock cycles, resulting in a DoS attack. Unlike many published hardware Trojans, this design is continuously active and is placed into the peripheral logic of the ASIC. This resulted from the fact that we did not want to expect the adversary to have deep knowledge of the internals of the design, as such information could be difficult or even impossible to obtain.

Since the kill sequence is observed through a serial 8-bit interface, there is a trade-off between the size of the Trojan circuit and the length of the kill sequence. For this design a 30-bit sequence[2] was used, i.e., the same number of Flip-Flops (FFs) were needed, which made up the majority of the area of the hardware Trojan. The Trojan was designed to tap into two input bits in parallel (`InxDI[0]` and `InxDI[2]`), resulting in the architecture shown in Figure 2.5. The wires required to connect the Trojan to the target design were placed close to the original connections and added very little additional load to the circuit. The input sequence, obtained during multiple read cycles, is continuously compared to the kill sequence—which got assigned the hexadecimal value `0x0DDDDDDD`[3]—using a simple combinational circuit. As soon as the sequence is detected, data input `InxDI[1]` is inverted. Once again, there is very little intrusion to the original circuit; only an XOR gate is inserted into an external I/O path that typically is off the timing critical path anyway.

One remaining problem with regard to the Trojan insertion was the clock connection needed for the FFs used in the design. The red team chose not to use the regular clock signal, but instead used the input signal `InReqxSI` of the original circuit. This signal is set once a new valid data item is present on the input and thus, can directly be used as the clock for the Trojan circuitry. Additional buffers were inserted manually in the clock path of the Trojan to reduce the load on the `InReqxSI` signal.

[2]In our case, we knew that the chip would not undergo exhaustive functional tests after production, so we were able to use a slightly shorter sequence to keep the Trojan area small. In a more realistic scenario, our victim circuit would also be more complex, allowing us to hide a longer kill sequence that would withstand even a very thorough functional test with a higher probability.

[3]Since the kill sequence is only 30 bit and not 32 bit long, the first two zeros are truncated from that value.

Many hardware Trojan taxonomies exist in the literature (see, for instance, [115, 35, 65]). According to those, our Trojan can be classified as follows:

Design phase:	Fabrication
Activation:	Externally triggered
Effect:	Denial-of-service
Location:	I/O
Abstraction level:	Gate level

2.2.5 Trojan Insertion Process

The actual insertion process of the Trojan, accomplished by the red team, was performed in three steps:

1. First, we identified the I/O pads and followed their connections. The input and output circuitries of an ASIC can easily be identified and distinguished by anyone who has access to the mask-layout (due to, for instance, different transistor sizes). Furthermore, common standard cells such as FFs can be detected with relative ease by an experienced layout engineer. This also leads to the identification of critical nets such as clock and reset. We had to find four signals: one for the clock, two for the kill sequence observation, and one that will be flipped by the kill switch. Once some candidate I/O signals were identified, a suitable region on the ASIC was searched for the insertion of the Trojan cells. The goal was to find an area that was close to the connections of the target design and had sufficient room to accommodate the logic gates of the Trojan. The approximate area is highlighted in Figure 2.6.

2. In the second step, we removed some of the filler cells[4] of the Chameleon design to make room for the Trojan circuit.

3. Finally, the signal and clock routing was completed using the available gaps in the actual circuit. The manual routing was accomplished mainly on inner metal layers, namely on layer three

[4]Filler cells are added to a design during place and route to improve certain electrical characteristics. They do not affect any functionality.

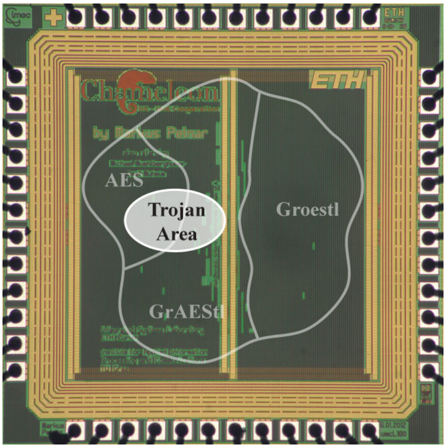


Figure 2.6: Chip photo of the fabricated Chipit ASIC.

and four of the utilized six-metal-layer technology in order to thwart potential visual inspections (not assuming the use of sophisticated delayering techniques). We cut through the original connection of input `InxDI[1]` (i.e., the *kill bit*) and connected the combinational logic as well as the shift registers of the Trojan by adhering to the design rules of the target technology.

All these steps were accomplished on the mask data in GDSII format exclusively. Therefore, any mistake during the Trojan insertion can lead to a total failure of the original circuit, which would most likely expose the attacker. As such, the adversary requires considerable technical skills to perform the attack. The data was read into Cadence Design Systems Virtuoso 5.1.41 EDA design software and the modified layout was exported in GDSII format again. Both Chameleon and its malicious counterpart *Chipit* were then fabricated using the 180 nm CMOS process by United Microelectronics Corporation (UMC). Figure 2.6 shows a photo of Chipit with the three independent components and the approximate area where the Trojan cells are inserted highlighted. Due to the integration of 30 FFs, the resulting Trojan requires a *comparatively large* area, i.e., about 190 GE, which is equal to 0.5 % of the overall Chameleon design.

Table 2.2: Distribution of chips for measurement.

Design	Trojan	Chips
Chameleon	no	#2, #3, #5, #6, #8
Chipit	yes	#1, #4, #7

2.2.6 Measurement Setup

In total, around 60 chips of both Chameleon and Chipit were manufactured. Out of these a mix of eight chips were bonded in a QFN56 package. That set contained three malicious and five genuine chips as listed in Table 2.2. These were then handed over to the unsuspecting blue team for testing. The blue team was given the datasheet for Chameleon and designed a test setup to correctly run the chip. No additional information about the mix of genuine and malicious ICs were relayed from the red team to the blue team until the end of the study. The eight chips were tested as described in the following sections.

The blue team's measurement setup mainly consisted of four parts: an adapter board with a socket for our (Trojan infected and non-infected) ASICs, a controller board, a digital oscilloscope, and a Personal Computer (PC). In order to perform side-channel analyses, we decided to design a Printed Circuit Board (PCB) that allows flexible power measurements using a QFN56 socket. This PCB was connected to a controller board. The controller board features an FPGA that was needed to provide all necessary interface signals to Chameleon/Chipit. The controller board was then connected to a PC that governs the overall measurement process. We used MATLAB® as it offers sophisticated analysis tools to perform side-channel analyses. Figure 2.7 shows a simplified overview of our setup. In the following, we describe the components and connections in more detail.

Adapter board: The adapter board has 51 pins that connect the CUT with the controller board. Only 48 pins were used by the CUTs; the remaining pins are still connected but were not used in our experiments. The main component of the adapter board is the QFN56

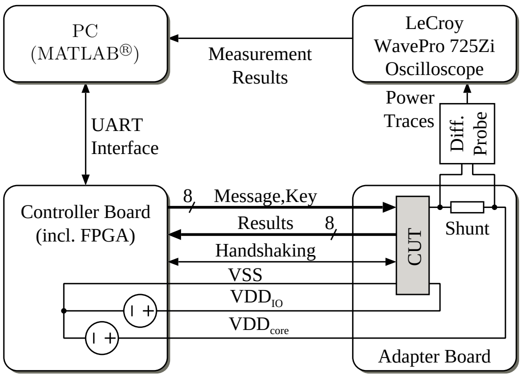


Figure 2.7: Simplified measurement setup for Trojan detection.

socket, required to easily exchange the individual CUTs. To measure the power consumption, the board provides three additional pin headers: one ground, one VDD_{IO}, and one VDD_{core} pin header. Therefore, the voltage drop across a measurement resistor can be measured with the oscilloscope in the ground or power lines of the I/O and core supply of the CUT. In our experiments, we measured the voltage drop across a $1\,\Omega$ shunt resistor in the VDD_{core} line because it contained less noise from the ground or I/O communication. In addition to the power measurement pins, all I/O pins of the CUT are available as pin headers to facilitate triggering on different I/O signals, for instance, on the start signal of the AES implementation (`StartxSI`).

Controller board: A photo of the controller and adapter board is given in Figure 2.8. The controller board communicated with a PC running MATLAB® via a Universal Asynchronous Receiver/Transmitter (UART) interface. A Xilinx Spartan-6 FPGA represents the core component of the controller board, which allows fully flexible pin assignments and controlling. In addition to the UART interface, it implemented the communication with the CUT using the following protocol: First, we selected the proper mode of operation of the CUT. We decided to target the stand-alone version of AES (`SelUnitxSI` = 1) and performed encryptions only (`SelModexSI` = 0). Second, the

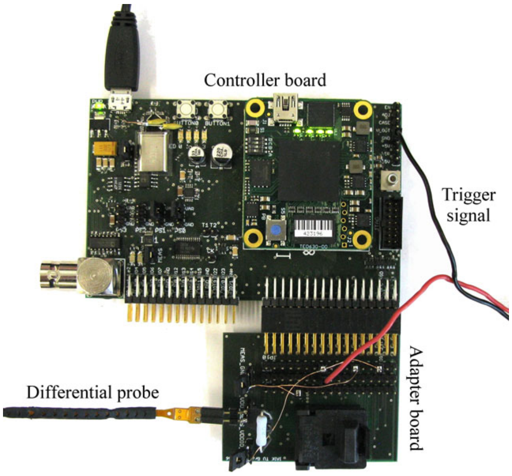


Figure 2.8: The controller board and the adapter board including the QFN56 socket for the CUTs.

PC sent a 128-bit AES key and a 128-bit plaintext block to the controller board. The data was stored in a 256-bit register and split into 8-bit chunks to interface with the CUT. Following the AES encryption, the ciphertext was again memorized in an internal register and transferred back to the PC. To verify the correctness of the ciphertext, it was compared against a reference software implementation.

Power consumption was measured using a LeCroy WavePro 725Zi oscilloscope. We used a sampling rate of 1 GS/s and captured the entire AES encryption operation (including I/O communication). As a differential probe we utilized an active probe of type LeCroy Dx20-SP. The oscilloscope was connected to the PC over Local Area Network (LAN) and transmitted the acquired power traces in sequence mode, i.e., 100 AES encryptions were executed before sending the power trace blocks over LAN. For the following power analyses, we set the baudrate to 19 200 bit/s and the CUT clock frequency to 10 MHz. Furthermore, we kept the AES key and the input constant to reduce noise that was caused by the data. For each CUT, we measured one million power traces, which took about 2.5 hours per CUT. This amount of traces was required for three reasons:

1. We expected a very low signal from the Trojan circuit as it was dwarfed by the original design.
2. The Trojan was always active and never in a dormant state, which made it hard to detect because of missing activation signals.
3. The more traces were used for averaging, the more noise was reduced, which helped us to characterize the Trojan signals.

2.2.7 Side-Channel Analysis (SCA) Results

After measuring a set of one million power traces per CUT, we performed the following analyses: First, we built templates of the measured power traces for each ASIC and applied Differential Power Analysis (DPA) techniques to identify Trojan-dependent signals. Second, we used Principal Component Analysis (PCA) to allow hardware-Trojan detection. Finally, we tried to improve the results using Support Vector Machines (SVMs) and trained them with test data to automatically allow a classification of the CUTs with a high success rate.

Building Templates

In a first observation, we wanted to answer the question if we can find major differences and high variances between the power traces of Chameleon and Chipit without any a-priori knowledge about the chips. For this purpose, we started to build power templates by calculating the mean of all power traces from each ASIC to eliminate potential measurement noise. The corresponding results can be represented using the measurements matrix **P**,

$$\mathbf{P} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{pmatrix}, \tag{2.1}$$

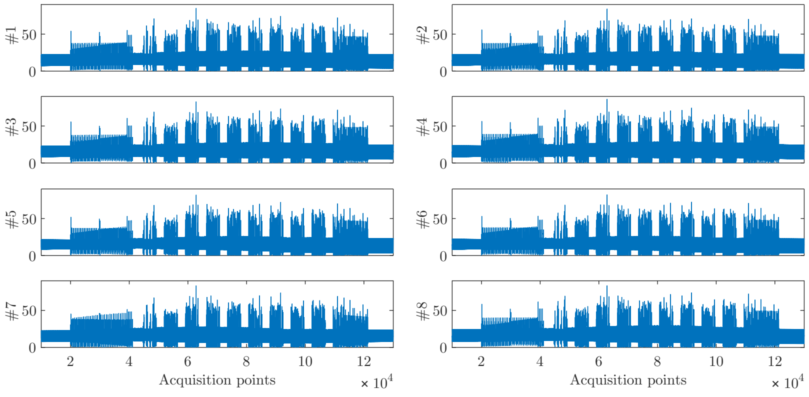


Figure 2.9: Absolute values of mean traces of all measured ASICs.

of size $m \times n$, where m refers to the number of measured chips (CUTs) and n denotes the number of acquisition points per trace. Hence, each row vector $\mathbf{p_{i,j}}$ for $i \in \{1, \ldots, m\}$ refers to the complete mean power trace of one measured CUT averaged over one million measurements. The column vectors $\mathbf{p_{i,j}}$ for $j \in \{1, \ldots, n\}$, on the other hand, contain the values of the various chips at the same acquisition point. Figure 2.9 shows the mean power traces $\mathbf{p_{1,j}}$–$\mathbf{p_{8,j}}$. From the mean traces, we can clearly identify the following operations: the loading of the input data (I/O communication) occurred at acquisition points 20 000–40 000. After that, an AES encryption was performed (we can discern the ten rounds of AES between the acquisition points 40 000 and 110 000) and the ciphertext was transmitted back to the controller board at 110 000–120 000. However, distinguishing any two distinct groups turned out to be difficult at this stage of the analysis.

Since process variation can be expected to be present in both genuine and malicious circuits, we then computed the difference between the mean traces and the total mean (i.e., from all eight chips) at each acquisition point according to equation (2.2),

$$p'_{i,j} = p_{i,j} - m_j \qquad i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}, \tag{2.2}$$

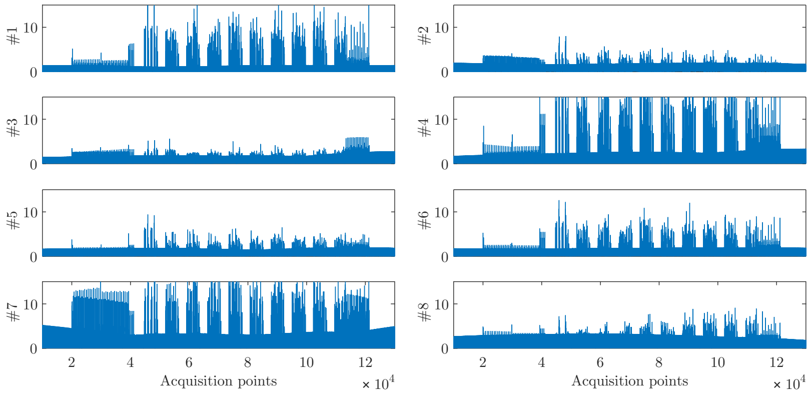


Figure 2.10: Difference between means and total mean of all CUTs, including both genuine (Chameleon) and malicious (Chipit) samples.

where m_j denotes the mean over all chips at each acquisition point j:

$$m_j = \frac{1}{m} \cdot \sum_{i=1}^{m} p_{i,j}$$

Note that we thereby did not only eliminate potential process variation, but also averaged over the power stemming from the Trojan gates contained within the malicious CUTs. Thus, depending on the ratio between process variation and Trojan impact, this step may push apart the two groups of chips more or less. See Figure 2.10 for the results of this analysis. From that figure, it seems that chips #1, #4, #6, and #7 stand out somewhat from the remaining four.

Closer analysis revealed that #1, #4, and #7 had a higher variance compared to the others. Furthermore, it was observed that there were no distinct acquisition points that provided high differences due to Trojan activity. Instead, differences were widespread over the entire power traces, which corresponded to our expectations since the injected Trojans were always active and never dormant.

In the following, we zoomed into regions with high variance to study Trojan-dependent signals more thoroughly. Figure 2.11 shows 20 acquisition points of all CUTs during a region where the differences

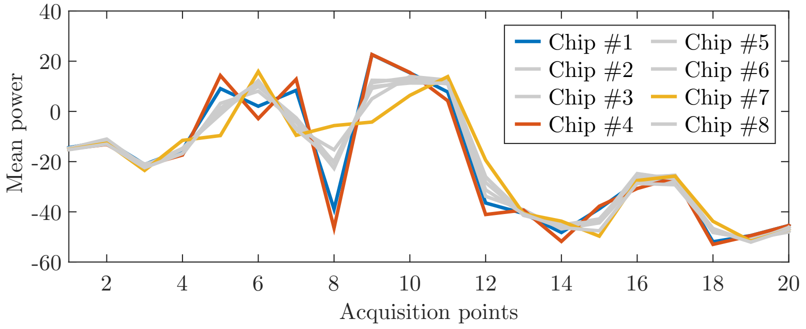


Figure 2.11: Points of interest of mean power traces.

between the data sets were comparatively high. The acquisition points of chip #1, chip #4, and chip #7 are marked in different colors (blue, red, and yellow); the remaining points are printed in gray. It became apparent that the mean power of those three samples—which were expected to be revealed as the malicious designs at that time—differed from the mean power of the Chameleon ASICs in several points. For instance at acquisition point 5, chip #1 and chip #4 consumed more power while chip #7 needed less compared to the others. The converse applied at point 6. We were able to identify such differences in many points throughout the mean power traces (usually at strong peaks that were caused, for instance, by the internal AES clock signal). However, for all differences we observed that the mean power of #1, #4, and #7 differed significantly from the other mean power traces in either the amplitude and/or time dimension. This led the blue team to their first hypothesis marking those three chips to be expected to contain the hardware Trojan.

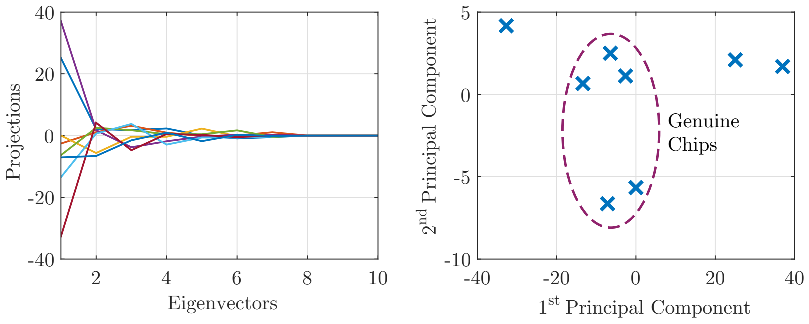
Classification Using Principal Component Analysis (PCA)

In order to strengthen the initial hypothesis of the blue team and to reduce noise and more efficiently discover Trojan-dependent signals, we then applied PCA [62]. Agrawal et al. [6] were the first to propose PCA for classifying malicious and genuine ASICs. They applied the Karhunen-Loève (KL) method to allow classification based on IC fingerprinting using simulated power traces and postponed the analysis of *actually fabricated ICs* as future work.

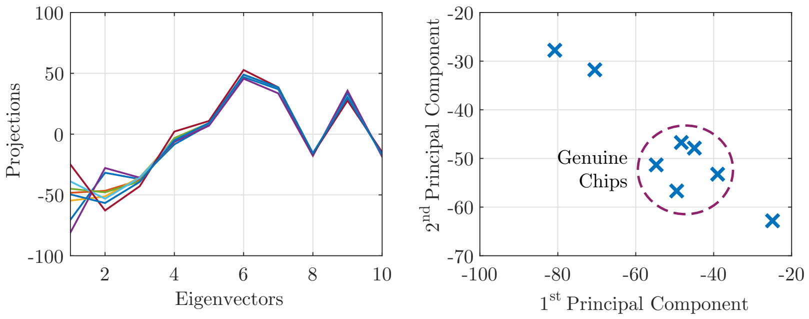
The main idea of PCA is to reduce the high dimensionality of the power traces to only a small number of features without loosing actual information about the original data set. For that, the direction of greatest variance is used as the first principal component in the new coordinate system. Previous work mainly used PCA to generate a feature matrix solely based on side-channel information obtained from genuine chips and then projected the measurements from a chip under investigation onto the newly obtained coordinate system.

In our work, we treated all manufactured ASICs on equal terms and tried to classify them into different groups by reducing the dimension of the data set with the use of PCA. For the classification we only investigated 20 acquisition points (*points of interest*) of the overall traces, where the variance was comparatively high. The chosen points of interest can be considered as samples of the larger population of power values available from a complete power trace. In fact, we chose the same points as in Figure 2.11. Hence, input dimensionality for the PCA was 20 in our experiments. The reduction of the higher-dimensional data set to fewer features was accomplished by computing the covariance matrix of the input data. Next, the eigenvectors were calculated and sorted in descending order according to their corresponding eigenvalues (i.e., we applied the scree test [33] to determine the importance of the eigenvectors). The feature matrix was then multiplied with the original traces to obtain the projected (transformed) data. Three different approaches were evaluated for building the feature matrix of the PCA.

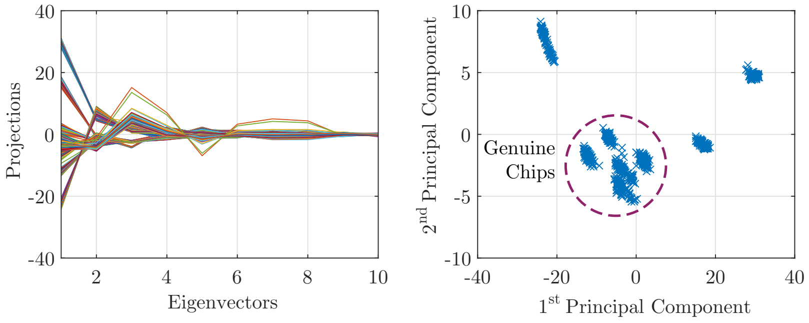
1. Feature matrix based on all traces: As a first approach, we took all power traces gained from averaging over one million measurements to eliminate measurement noise as far as possible. The left image of Figure 2.12a shows the projected power traces onto the first ten eigenvectors obtained from all eight chips. The upper row of Table 2.3 lists the corresponding first five eigenvalues. From Table 2.3 it can be observed that the first eigenvalue stood out significantly when using all power traces. In the right image of Figure 2.12a, we have plotted the projected power traces using only the first two principal



(a) Feature matrix obtained using power traces from *all* eight chips.



(b) Feature matrix obtained using power traces only from *genuine* chips.



(c) Feature matrix obtained using *grouped* traces from all eight chips.

Figure 2.12: *Left:* Projections of the power traces onto the first ten eigenvectors (ordered in descending order according to their corresponding eigenvalues); *Right:* The resulting classification using only the first two principal components for different feature matrices.

Table 2.3: Corresponding eigenvalues of the eigenvectors obtained from the different PCAs.

Used	**Number of Eigenvalue**				
Traces[†]	**#1**	**#2**	**#3**	**#4**	**#5**
All	477.8	15.8	9.9	2.8	1.3
Genuine	34.3	16.3	3.1	1.4	0.0
Grouped	237.9	14.5	10.1	1.3	1.1

[†] Power traces used to create the feature matrix of the PCA.

components. As could have already been expected from the eigenvector projections and the corresponding eigenvalues, mainly the first principal component distinguished the groups of genuine and malicious ASICs. Higher dimensions of the projections did not contribute any valuable information to the classification.

2. Feature matrix based on traces from genuine chips only: Since most of the related work suggests to use only the genuine ICs for building the feature matrix, we decided to try this approach as well. It is important to notice that this requires a-priori information, which contrasts with the approach from Figure 2.12a. As a result, the feature matrix becomes more specific for the genuine fingerprint and therefore, allows a cleaner separation between the two ASIC groups. Figure 2.12b shows the projections onto the first ten eigenvectors (left image) and the corresponding classification based on the first two principal components (right image). Obviously more information has been obtained from the second dimension of the projection compared to the first approach. Also the eigenvalues (cf. middle row of Table 2.3) indicated that the first two components contain significantly more information than the remaining ones. Moreover, this time the second dimension provides more information compared to the first one, as pointed out by the ratio between the two corresponding eigenvalues.

3. Feature matrix based on grouped traces: To artificially increase the number of available input data and reduce the averaging

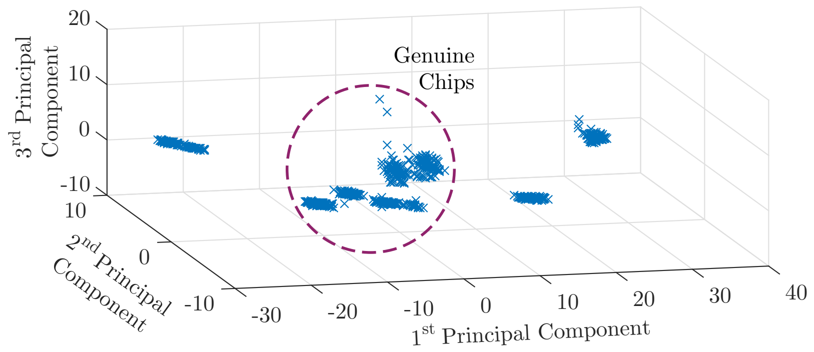


Figure 2.13: Classification of the genuine and malicious chips using PCA based on the first three principal components. The feature matrix for the analysis is obtained using 100 mean traces (each averaged over 10 000 measurements) per chip.

effect due to the large number of measurements per chip, we applied a third approach for the computation of the feature matrix. For each ASIC, we calculated 100 mean power traces, where 10 000 traces were used for averaging (i.e., we split the one million traces per chip into 100 groups of 10 000 traces each). Thus, we obtained a matrix of mean power traces with 800×20 elements when only considering our points of interest. Applying PCA to this data set resulted in the traces projected onto the eigenvectors shown in the left image of Figure 2.12c. The corresponding classification based on the first two components is illustrated in the right plot of Figure 2.12c. Although the feature matrix was built using the trace groups from all chips this time, the result was roughly as clear as the classification when building features only from the genuine chips.

The third eigenvalue (cf. bottom row of Table 2.3) indicated that the third principal component might also contain valuable information for the separation of the two groups. The resulting three-dimensional classification plot is given in Figure 2.13. Although the main eight groups of traces can clearly be identified from Figure 2.13, some outliers in the third dimension distorted the results slightly. These outliers were mainly due to the information from higher dimensions of

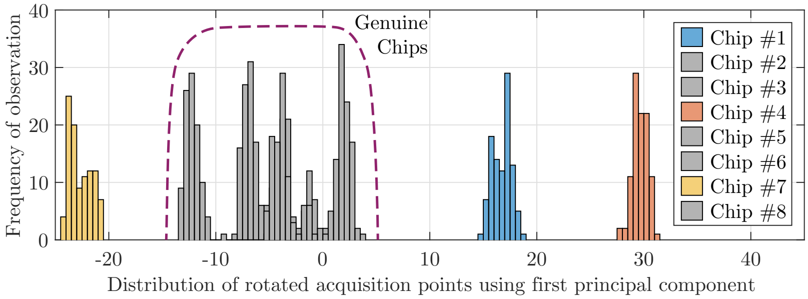


Figure 2.14: PCA-based classification using only the first principal component. The feature matrix for the analysis is obtained using 100 mean traces (each averaged over 10 000 measurements) per chip.

some of the genuine chips, as can be observed from the left image of Figure 2.12c. Since the highest eigenvalue stood out significantly for all three analyzed feature matrix constructions, we also decided to classify the 800 mean traces based on only the first principal component. Figure 2.14, showing the results of this analysis presented as a histogram, indicates that even the first component alone was enough to reach a nice separation between Chameleon and Chipit.

Closing remarks on the PCA analysis: If the variance between the acquisition points is not due to actual Trojan influences, but, for instance, because of process variation, it is most likely that PCA will not work that well for the classification anymore. As a result, PCA may mislead the user to erroneously identify chips as malicious just because they got produced on a different wafer lot. Further investigations would require close collaboration with a foundry. Nevertheless, we were able to distinguish genuine and malicious ASICs only by looking at up to the third principal component.

Support Vector Machines (SVMs) Classification

To accomplish Trojan classification more automatically, we decided to evaluate SVMs as well. Basically, SVMs can be used to tell apart a set of power traces obtained from malicious and genuine chips after

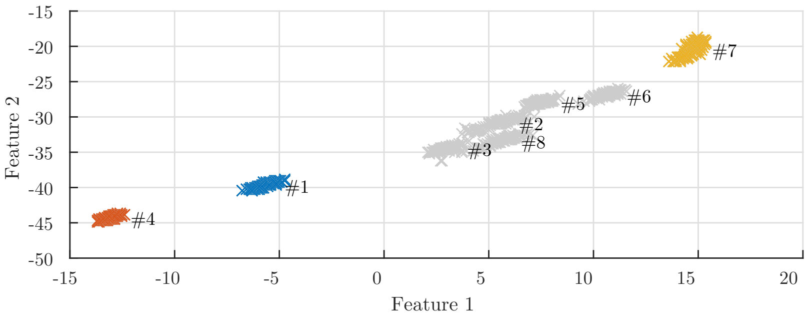


Figure 2.15: Result of an SVM-based classification using a two-dimensional feature vector.

training them with sample data. If trained successfully, they allow an efficient detection of manufactured ICs that do not map to the trained (high-dimensional) feature space of non-infected chips. Hospodar et al. [52], for example, successfully applied this technique in the context of side-channel analysis. Similar to the PCA approach, where the feature matrix was generated based on the power traces from the genuine chips, SVM classification requires a-priori knowledge regarding the distribution of the Trojan ICs.

We used the statistics toolbox in MATLAB® to perform SVM classification, which provides the built-in method `svmtrain` to train an SVM. As input, `svmtrain` takes the training data containing the feature values of the measured power traces. Similarly to PCA, this training data is represented by a matrix with R_{td} rows and C_{td} columns where R_{td} equals to the number of training data and C_{td} equals to the number of features. Second, a group vector with length R_{td} is required. This vector defines the group membership of the training data. A third parameter defines the kernel function used for the SVM. For the classification in our experiments, we applied a *quadratic* kernel function as this was found to perform best. The method `svmclassify` was used to classify new data according to the previously trained SVM. It only requires the input data, which has to be classified. Figure 2.15 shows the result of one classification of the CUTs using the SVM. The feature vector for this classification is two-dimensional. To find suitable points for the feature vector, the variance for each acquisition point was considered. Eventually, we chose the same points of interest as used for PCA to allow a comparison of the results. Again, we took the 800 traces from the grouped-traces approach of the PCA as available input data (i.e., 100 traces per CUT). Ten feature vectors from chip #1, chip #2, and chip #7 served as training data for the SVM. It turned out that these 30 out of 800 traces (i.e., 3.75 %) were sufficient to correctly classify the remaining data. A further experiment indicated that even a single trace per chip (eight traces in total) for training was also sufficient to perform a classification without errors. The resulting classification clusters are illustrated in Figure 2.15. With this approach, the Chipit ASICs (chip #1, chip #4, and chip #7) can be automatically distinguished from the Chameleon chips (chip #2, chip #3, chip #5, chip #6, and chip #8). Recall that compared to the previous classification approaches based on the difference of means and PCA, the SVM-based technique requires a-priori knowledge about which ASICs contain Trojans and which do not. If, for example, only chips without a hardware Trojan are available for training, some sort of threshold detection based on the feature vector can be used to classify new chips.

Next, Section 2.3 analyzes hardware Trojans integrated into an FPGA device. Final remarks about ASIC Trojans are provided as part of Section 2.4 and conclusions are drawn in Chapter 5.

2.3 Localization of FPGA Trojans Using Electromagnetic Radiation (EM)

Ensuring the integrity of ASICs, as described in the previous section, can be considered as the *lowest level* for which trust must be established from a digital hardware designer's point of view. However, a significant portion of today's circuits do no longer get implemented as ASICs, but employ FPGAs due to their higher flexibility and lower up-front costs. Therefore, the integrity of the configurations of the FPGAs (often referred to as the *bitstream*) in a project must also be assured to get a continuous trust chain for a hardware-based, security-critical application.

However, the FPGA's most compelling benefit, its reconfigurability, becomes one of its weakest points. A large amount of FPGAs on the market are based on SRAMs. Hence, they need to load their configuration from an external, non-volatile memory immediately after power-up. Despite the fact that the bitstream usually gets encrypted, [79, 80, 81] have shown that those methods cannot be trusted completely. Therefore, system houses need to prepare for attackers who maliciously modify the configuration before it actually gets loaded into the FPGA.

To simulate such a scenario, we decided to place a hardware Trojan into an FPGA design without knowledge of its HDL model. To do so, we use RapidSmith [71], an open source library for low-level manipulation of partially placed-and-routed FPGA designs, to alter an intermediate data format of a Xilinx Virtex-II Pro FPGA design. We show how to attach a sequential DoS Trojan to the I/O ports of an existing AES architecture. On the one hand, we aim at detecting the malicious configurations. On the other hand, we try to suggest where on the FPGA the Trojan is harder/easier to detect using EM side-channel analysis. To achieve these goals, we utilize an electromagnetic probe and step over the package of the FPGA, measuring the electromagnetic field for each step point.

Various works have confirmed the practicability of Side-Channel Analysis (SCA) for Trojan detection. However, to the best of our knowledge, at the time of conducting this experiment only [13] actually accomplished a setup including EM measurements in their experiments. As opposed to [13], who gathered EM traces from a single location on top of an FPGA, we use a localized approach by stepping over the FPGA's package using a stepper table.

2.3.1 FPGA Design Flow and Attacker Model

The top row of Figure 2.16 describes a simplified version of the Xilinx design flow [107], starting with the mapping of a netlist to the available FPGA-specific resources. An attacker with access to these parts of the development chain may incorporate malicious circuitry into an existing design by modifying the intermediate *.ncd file format. This task can be simplified by employing third-party tools such as Rapid-

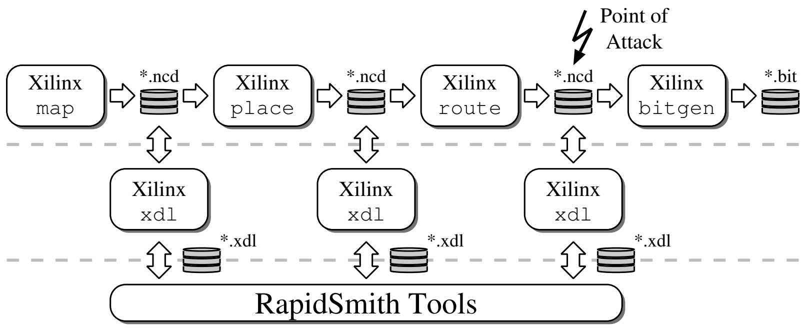


Figure 2.16: Xilinx design flow and RapidSmith interface

Smith [71], a library for low-level manipulation of partially placed-and-routed FPGA designs. RapidSmith is a set of tools and Application Programming Interfaces (APIs) written in Java, that support importing, manipulating, and exporting of FPGA designs. It is compatible with the Xilinx Design Language (XDL), a human-readable file format equivalent to the Xilinx proprietary Netlist Circuit Description (NCD). Although the Trojan insertion process can also be accomplished with Xilinx tools only, employing RapidSmith greatly simplifies the procedure.

Figure 2.16 shows that different possible entry points to the RapidSmith tools exist. For our investigations, we fed RapidSmith release 0.5.2 with a placed-and-routed design in *.xdl file format by converting the *.ncd file to an *.xdl file with Xilinx's `xdl` tool. Although RapidSmith does not allow direct manipulation of *.bit files, an adversary may also reverse engineer one using tools such as the Bitfile Interpretation Library (BIL)[5] [17] and apply a similar attack afterwards.

2.3.2 Trojan Circuit

As a target design, we selected an AES-128 architecture. Because Trojans must be kept small, we decided to implement a DoS variant. Ac-

[5]At the time of writing, BIL supported only Xilinx Virtex-5 devices.

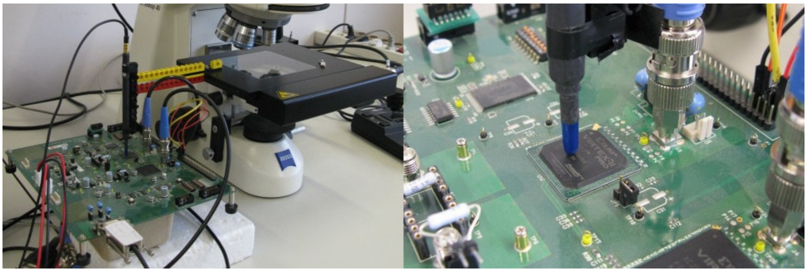


Figure 2.17: Stepper table, SASEBO-G, and HF near-field probe

tually, we used the same malicious circuit as described in Section 2.2.4 for our ASIC experiments [85], the design of which can be observed in Figure 2.5. On the target FPGA, our DoS Trojan occupied a mere 15 slices. As the original design of the AES core required 2222 slices on the Virtex-II, the 15 slices of the Trojan were equal to 0.7 % of the overall FPGA utilization. This was roughly the same proportion as in the ASIC experiment in Section 2.2.

2.3.3 Measurement Setup

Our measurement setup comprised the following parts: the SASEBO-G side-channel evaluation board [101], a digital storage oscilloscope (the LeCroy WavePro 725Zi), an EM stepper device with an EM measurement probe, and a PC running MATLAB® as controlling software. Figure 2.17 shows the SASEBO board and the EM stepper on the left side and a zoom into the EM stepping location on the right side. The SASEBO board was connected to a PC via a serial interface. It hosts two FPGAs, a Virtex-II Pro XC2VP30 and a Virtex-II Pro XC2VP7. One FPGA can be used as a controller, while the other FPGA carries the cryptographic circuit to be examined. Both FPGAs were clocked with a frequency of 16 MHz supplied by a Digimess FG100 function generator.

The communication between the two FPGAs worked as follows: First, the PC transmitted 256 bits of data (the 128-bit cipherkey and a 128-bit plaintext message) over the UART interface. This data was stored in an internal register. After that, the data was transmitted to

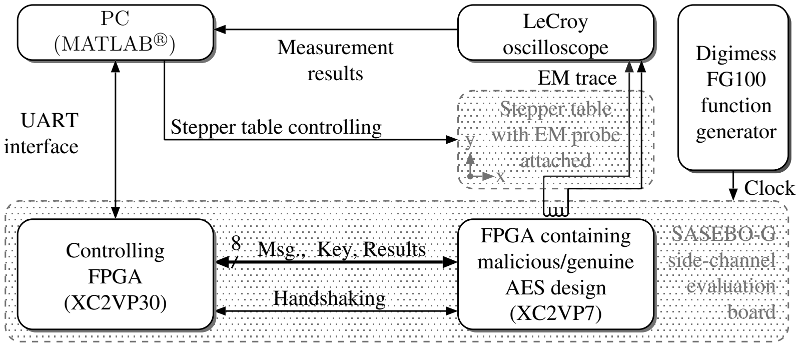

PC
(MATLAB®)
LeCroy oscilloscope
Measurement results
Digimess FG100 function generator
EM trace
UART interface
Stepper table controlling
Stepper table with EM probe attached
y
x
Clock
Controlling FPGA (XC2VP30)
8
Msg., Key, Results
Handshaking
FPGA containing malicious/genuine AES design (XC2VP7)
SASEBO-G side-channel evaluation board

Figure 2.18: Schematic view of the measurement setup for detecting the hardware Trojan in the FPGA using EM fingerprinting.

the evaluation FPGA in chunks of eight bits, where the correct order and the timing were guaranteed by a dedicated handshake protocol. Once the AES calculation had finished, the result was sent back to the control FPGA, which transferred the data back to the PC. Figure 2.18 shows a simplified overview of the setup and the implemented communication flow.

As an EM probe, we used a magnetic near-field probe from Langer EMV Technik (LF-B 3). The probe was attached to a stepper table controlled by software via MATLAB®. The stepper table allowed us to move across the FPGA. The accuracy (i.e., the number of step points and the step window) was controlled by a software application. In our experiments, we stepped over a window of about 10 by 10 mm advancing the probe in steps of 300 µm (31 steps in the x and y axis). At each step location, we measured 2500 traces of the same operation (keeping key and message data constant) and calculated the mean trace in order to reduce noise. We therefore obtained $31 \cdot 31 = 961$ mean EM traces for each implementation. The sampling rate of the oscilloscope was set to 500 MS/s and the oscilloscope was configured to record the traces in sequence mode. This means that 500 traces were recorded in a row by the oscilloscope, before being transferred to the PC. This significantly improved the overall measurement speed.

The Xilinx Virtex-II Pro XC2VP7: To successfully identify and locate the injected Trojan on the Xilinx Virtex-II Pro XC2VP7, it is necessary to have knowledge about the floorplan and the individual components included in the FPGA. The XC2VP7 features an embedded PowerPC processor that can be clearly identified in the floorplans (cf. major building block in Figure 2.19a–2.19f). The FPGA further includes eight RocketIO transceiver blocks, 1232 Configurable Logic Blocks (CLBs)[6], 44 dedicated 18×18 bit hardware multipliers, 44 18 kbit Block RAMs (BRAMs), and four Digital Clock Managers (DCMs). The entire Virtex-II Pro family is fabricated in a 130 nm CMOS process technology with nine metal layers.

2.3.4 Trojan Insertion Process

To inject the Trojan into the final FPGA configuration after placement and routing, we developed a Java program using the RapidSmith API. We generated six different FPGA configurations with the Trojan placed:

(a) in the top-right corner,

(b) in the bottom-left corner of the FPGA,

(c) in the center,

(d) distributed over the whole floorplan,

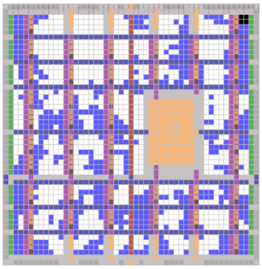
(e) on the right hand side near the I/O-lines, and

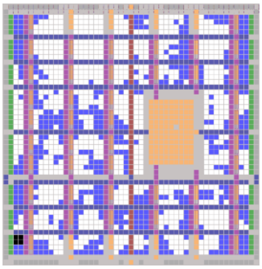
(f) automatically after the design has been completely re-routed by the Xilinx ISE place-and-route tool.

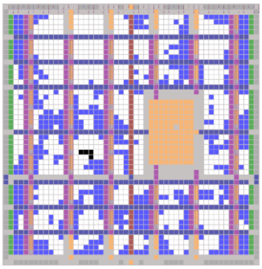
Note that the last design, i.e., the completely re-routed one, was only given for comparison as we did not expect the attacker to be so unsophisticated to insert the Trojan into an HDL description of the FPGA configuration. As this would result in completely different floorplans, SCA fingerprinting would easily reveal the departures from the original design.

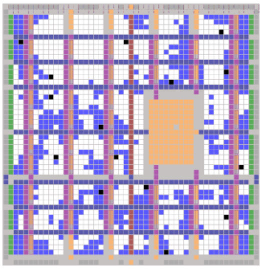
Figure 2.19 shows the FPGA utilization of the six designs after Trojan insertion and illustrates the floorplan of the Virtex-II Pro die.

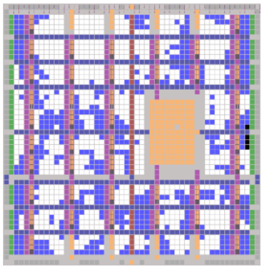
[6]Each CLB of the XC2VP7 consists of four slices and two 3-state buffers, where each slice contains two 4-input Lookup Tables (LUTs) and two FFs.


(a) Top-right


(b) Bottom-left


(c) Center


(d) Distributed


(e) Next to I/O pins

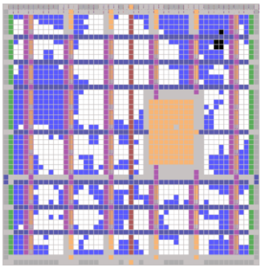

(f) Re-routed

Figure 2.19: The six FPGA designs with the Trojan placed at different locations; Occupied and not occupied CLBs are colored in white and blue, respectively; Trojan logic is drawn in black

The PowerPC is drawn as a yellow box in the center of the floorplan. BRAMs are indicated in purple color (six vertical lines) and the Digital Signal Processing (DSP) units are drawn in orange. Clocking resources are represented in brown (vertical line in the middle) or dark-blue (horizontal lines). Multi-gigabit transceivers and DCMs are located at the top and bottom sides of the BRAMs. We further marked all occupied CLBs in white color and all unused CLBs are drawn in blue. Note that the FPGA configuration illustrated in Figure 2.19f uses different CLBs than the other designs because the design has been re-routed by the Xilinx place-and-route tools following injection of the Trojan. Moreover, for the design given in Figure 2.19d, we distributed the Trojan logic to 15 different slices in 15 different CLBs.

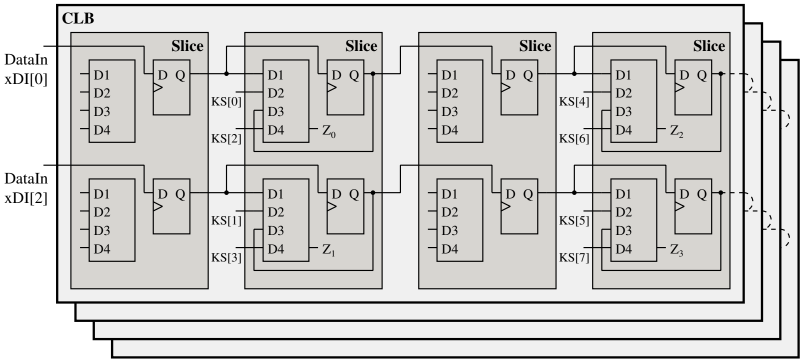


Figure 2.20: Simplified overview of the mapping of the Trojan logic into the FPGA resources. Signals `KS[x]` refer to the bits of the predetermined kill sequence (i.e., are constant).

For the remaining designs, we tried to use all unoccupied slices within one CLB to inject the Trojan (needing four CLBs) and instructed the routing tool to route only those parts that had not been routed before (thus, keeping the overall floorplan and resource requirements nearly constant). CLBs containing the Trojan logic are colored in black in Figure 2.19.

Trojan resource requirements: The resource requirements of the Trojan itself were as follows. Since in total a sequence of 30 bits (15 bits from `DataInxDI[0]` and 15 bits from `DataInxDI[2]`) had to be stored, 30 FFs were needed. A Virtex-II Pro slice contains two FFs so we required a total amount of 15 slices and thus, at least four CLBs to implement the Trojan circuit due to the memory elements. In particular, we decided that the FFs of the upper half and of the lower half of the slices should be used to store the sequences observed at signal `DataInxDI[0]` and `DataInxDI[2]`, respectively. These FFs were further connected to form the required shift registers. Figure 2.20 illustrates the mapping of the Trojan logic into the FPGA resources. For the kill-sequence comparator, we utilized the LUTs of the slices already occupied by the Trojan FFs. Because the LUTs from the top half and lower half of a slice had four inputs ($D1$–$D4$), we compared two bits of the observed sequences with one LUT. The first LUT input was used for the first bit of one of the two input sequences and the second LUT input for the first bit of the fixed kill sequence to compare with. The third input was routed to the second bit of the input sequence and the fourth input to the second bit of the kill sequence (cf. Figure 2.20). The comparison was done with the following logic function:

$$Z_i = ((D1 \wedge D2) \vee (\overline{D1} \wedge \overline{D2})) \wedge ((D3 \wedge D4) \vee (\overline{D3} \wedge \overline{D4}))$$

All intermediate results Z_i were then combined with a bitwise AND, implemented in one of the empty LUTs. Eventually, the final result was XORed to the kill bit obtained from data input `DataInxDI[1]` to invert the signal if the received input sequence matches the kill sequence.

2.3.5 Measurement Results

We started our experiment by creating a genuine EM fingerprint against which measured EM traces were later compared to distinguish malicious and Trojan-free FPGA configurations. Figure 2.21 shows the EM map of the 961 stepping points of the differences between two measurement sets of the genuine design. The EM map color-coding given in Figure 2.21 remains the same during the rest of this section to allow for easy comparisons. As expected, the minor differences—supposed to stem from measurement noise—were basically negligible. x and y axis in Figure 2.21 represent the horizontal and vertical position of the stepper table (i.e., the position of the EM probe with respect to the FPGA package).

The genuine fingerprint was then compared with the six malicious designs containing Trojans placed differently. For this purpose, we calculated the absolute difference of all measured EM traces for the 961 EM-stepping points. Furthermore, we only focused on the I/O communication of the implemented AES core for our investigations (i.e., acquisition points 20 000–40 000 printed in Figure 2.9). We limited our measurements to this interval in order to keep the analysis effort reasonably low.

As for post-processing of the traces, we applied the following techniques: First, we aligned the traces in the timewise dimension be-

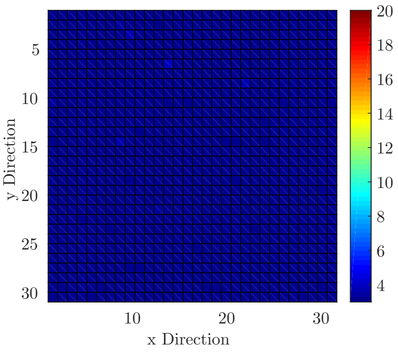


Figure 2.21: EM-signal differences of two measurement sets of the genuine (golden) design.

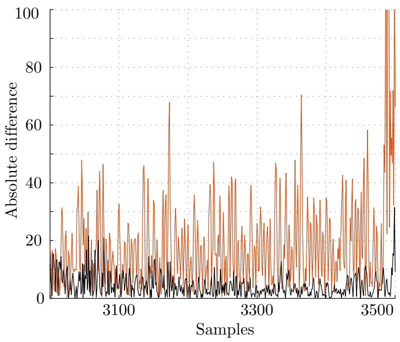
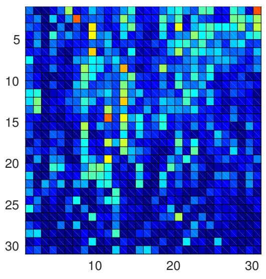


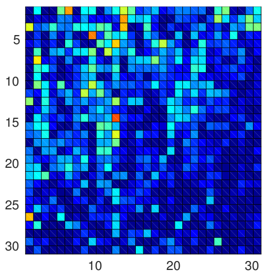
Figure 2.22: Difference of two traces at the same position before (red) and after (black) alignment.

cause they were not perfectly aligned due to noise and clock jitter. Figure 2.22 indicates the differences before and after the trace alignment. Second, to identify points of interest (i.e., points where we assumed a Trojan-dependent emission), we calculated the variance for each difference vector and considered the acquisition point with the highest value. Figure 2.23a to 2.23f depict the EM-signal differences of the genuine design and the six malicious designs. To create the two-dimensional plots, we mapped the difference vector to a matrix with 31 rows and 31 columns. Each point represents a different EM-probe location of which the color indicates the difference of the mean EM traces at the point of interest (blue means almost no difference and red indicates a high deviation).

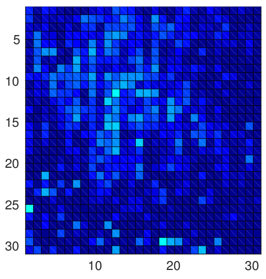
As a first observation, one can identify the high deviation of the re-routed design (f). This was expected because the entire design was automatically re-routed and therefore, its EM fingerprint significantly differed compared to the genuine design. Interestingly, it showed a high deviation in the top-right corner where the Trojan has been placed. Note that we measured the radiation over an area of about 10 by 10 mm. Therefore, the plots not only show the direct EM signals of the FPGA die (assumed to be located in the middle of the plot), but also the indirect radiation from bonding wires, including



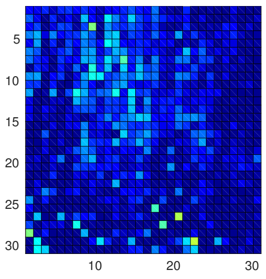
(a) Top-right



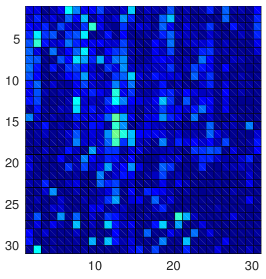
(b) Bottom-left



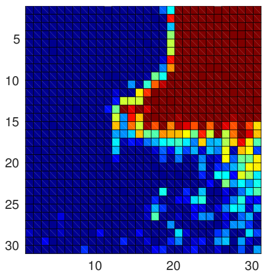
(c) Center



(d) Distributed



(e) Next to I/O pins



(f) Re-routed

Figure 2.23: EM maps taken at 31 $\times$ 31 distinct locations on top of the FPGA package, highlighting differences between Trojan-free and malicious designs for six Trojan placements.

ground lines and I/O communication. It became apparent that when the Trojan had been placed in the top-right (a) or bottom-left (b) corner of the FPGA, the Trojan-dependent signals were higher compared to the cases where the Trojan had been placed in the center (c) of the FPGA or when the malicious slices had been distributed all over the FPGA (d). The signal differences of Figure 2.23e (near I/O) seem to be higher compared to those of the centered design (c), but not as significant as the results obtained for the top-right (a) and bottom-left (b) cases. There are several possible explanations for that. One reason might be the fact that the Trojans that have been located close to VCC or ground lines have a higher influence (through EM-signal modulation or shorter wire lengths) on the power supply signals and can therefore be detected more easily. Trojans placed in the top-right (a) and bottom-left (b) corners are indeed close to many VCC and ground pins of the FPGA. Another reason might be that if the required CLBs of a Trojan are placed closely together, the signal leakage will be stronger.

We have succeeded in identifying all of the malicious FPGA configurations. However, we were not able to pinpoint the actual locations of the Trojan slices. Although all EM maps of Figure 2.23 indicate significant differences between the malicious and the genuine FPGA configurations, the locations of the differences in Figure 2.23 are not in line with the Trojan locations highlighted in Figure 2.19.

2.4 Final Remarks

For side-channel-based Trojan detection methods, usually a genuine design is required to create a reference fingerprint. Because ASIC fabrication is a time-consuming and expensive process, both a Trojan-free and a malicious variant of the CUT are hardly ever available. Our pair of Chameleon and Chipit ASICs is one of the very rare cases, where both of them were actually taped-out. As a result, already other institutions showed great interest in the chips and based their analysis methods on our ASICs [38].

So far we applied our EM-based approach for localizing hardware Trojans just on FPGAs. Despite the fact that we could not precisely pinpoint the malicious logic on the FPGA with our experimental setting, it would be interesting to apply an enhanced measurement setup to localize ASIC Trojans as well.

3 An ASIC for Assessing DPA Countermeasures

Outline. This chapter starts with an introduction about why integrated circuits must be protected against physical attacks in Section 3.1. Next, Section 3.2 discusses preliminaries about countermeasures against Differential Power Analysis (DPA). Because our implemented authenticated encryption scheme is based on Keccak, Section 3.3 introduces *sponges* in general and the Keccak-f permutation in particular. In Section 3.4, we present Zorro, an ASIC designed and fabricated exclusively for assessing DPA countermeasures on a real chip. Our results provided throughout Section 3.5 are twofold. On the one hand, we present hardware figures of our ASIC that targets applications in the field of resource-constrained environments. On the other hand, we give initial results from attacks against the actual chip, including a comparison about the quality of hiding and masking techniques. Finally, we summarize this chapter in Section 3.6.

3.1 Introduction

Assuring the integrity of either a fabricated ASIC or the bitstream of a configured FPGA, as discussed in Chapter 2, can be considered the *lowest level* for which trust must be established in a hardware-based security system. However, when it comes to applications of symmetric encryption, another component in the system must be investigated from a hardware security point of view. In order to share the secure key between two communicating parties, often smart cards, Universal Serial Bus (USB) sticks, or other electronic devices are used for distribution. Therefore, the key-hosting hardware component must be secured against attacks from potential adversaries as well.

Besides cryptoanalytical vulnerabilities, analysis techniques of implementations (often referred to as *attack techniques*, since they may reveal unintended internal secrets) have shown to be one of the major weaknesses for hardware devices providing keyed[1] cryptographic primitives. One such primitive is Authenticated Encryption (AE) [14], which tries to achieve the two goals of confidentiality and authenticity using a common approach. Previous methods have tried to provide these two necessities independently from each other, for instance, with the use of Block Ciphers (BCs) and Message Authentication Codes (MACs). However, several of these attempts turned out to be rather inefficient and suffered from security issues [29, 32]. Hence, researchers have started to develop new hybrid algorithms that offer the desired goal of AE, for instance, as part of the ongoing Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [1].

A group of primitives that has gained increasing popularity in the recent past are modes based on a fixed-length permutation. Especially the family of sponge constructions [21] and its closely related design principle, the duplex construction [26], represent well-established alternatives to traditional BC-based primitives. The best known representative of this family is the sponge-based hash algorithm Keccak, which has emerged in 2012 as the winner from the SHA-3 competition [93], initiated by the National Institute of Standards and Technol-

[1]By *keyed primitives*, we refer to any primitive that processes a secret (key) and therefore, needs to be protected against implementation attacks.

ogy (NIST). Moreover, permutation-based schemes are very flexible and can be used to realize cryptographic constructs such as Pseudo-random Number Generators (PRNGs), MACs, or AE systems. As a result, in this chapter we present Zorro, an ASIC exclusively developed for the purpose of assessing masking and hiding countermeasures on a real chip. More specifically, Zorro contains several AE primitives based on the Keccak-*f* permutation. Measurements acquired from the actually fabricated ASIC are used to analyze potential weaknesses and the impact of the implemented countermeasures.

3.1.1 Requirements and Vulnerabilities of Pervasive Hardware Devices

Ubiquitous hardware components, such as the key-distributing token of a symmetric encryption device, are usually limited in terms of silicon area, power, and energy. While the former is mainly due to the manufacturing costs of the component—roughly proportional to the chip area—the latter is often because of the way the devices are powered. Since passive smart cards are powered by electromagnetic fields radiated from a reader device, low peak power is a major requirement. For battery-powered components, on the other hand, the energy needed is of paramount importance. We plan to use our ASIC for applications in resource-constrained environments. Therefore, our main goal is to minimize the area of the overall chip (incl. the intended countermeasures) as far as possible.

Securing the processing of sensitive data is of particular interest for pervasive hardware devices, which are often accessible by the general public. Due to their availability in the public domain, an adversary can take measurements from such devices in order to mount various implementation attacks. Hence, countermeasures must be added to these devices to thwart potential attacks. Differential Power Analysis (DPA) [68] turned out to be one of the most powerful implementation attacks available today. Therefore, we taped-out an ASIC containing several DPA countermeasures in a mature 180 nm CMOS technology by UMC. Afterwards, we analyzed the Keccak-based AE architectures based on measurements acquired from the real chip.

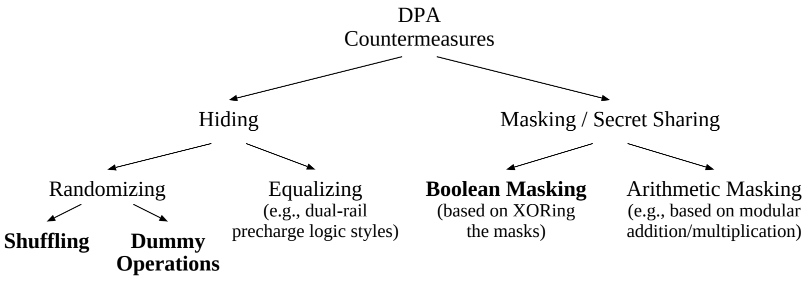

DPA Countermeasures
Hiding
Masking / Secret Sharing
Randomizing
Equalizing (e.g., dual-rail precharge logic styles)
Boolean Masking (based on XORing the masks)
Arithmetic Masking (e.g., based on modular addition/multiplication)
Shuffling
Dummy Operations

Figure 3.1: Taxonomy of DPA countermeasures. The techniques investigated are highlighted in bold.

3.2 DPA Countermeasures

Analysis techniques based on the power side channel can basically be divided into Simple Power Analysis (SPA) and DPA. As opposed to SPA, DPA usually involves various statistical post-processing methods based on multiple power traces. Power-analysis attacks basically work by exploiting the dependency of a cryptographic device's instantaneous power consumption from the data being processed. Therefore, an obvious way to discourage such attacks is to minimize this correlation. In general, DPA countermeasures can be categorized into *hiding* and *masking* techniques, both of which can be considered as standard methods for state-of-the-art security-critical electronic devices today [74]. Figure 3.1 provides a taxonomy of DPA countermeasures as discussed in the following sections.

3.2.1 Hiding

Hiding tries to minimize the dependency between the processed data as well as the corresponding operations and the power consumed either by *randomizing* or by *equalizing* the current drain of a device.

Randomizing: This technique can be achieved by randomizing the points in time when a certain operation is executed. Acquired power traces so become misaligned and must be adjusted in a preprocessing step, making any attack more difficult. Typical methods to randomly distribute the execution of operations over time are *shuffling* or the insertion of *dummy operations*.

With shuffling, operations within an algorithm are re-ordered randomly without changing the overall output. Dummy operations, on the other hand, do not operate on the actual data, but are added to randomize a device's power consumption. They operate on some dummy data. As opposed to shuffling, dummy operations decrease an implementation's performance due to additional computations.

Equalizing: While randomizing the power consumption is usually achieved by changing the point in time when a certain operation is executed, equalizing tries to even out current drain demand by balancing power dissipation. The ultimate (theoretical) target is to reduce the Signal-to-Noise Ratio (SNR) all the way to zero, thereby leaving a potential attacker completely with random noise, which makes attacks impossible. However in practice, equalizing the power consumption of all executed operations of a device is far from trivial. One way to achieve it is to use logic styles such as *Dual-Rail Precharge (DRP)*. *Sense Amplifier Based Logic (SABL)* [110] and *Wave Dynamic Differential Logic (WDDL)* [111] are just two examples thereof.

3.2.2 Masking

In contrast to hiding, masking does not only try to alter the power signature of a device, but changes the actual data being processed. Since the power dissipation due to the processing of the masked data is expected to be independent from the original secrets, it is not required to hide the data-dependency in the power signature anymore. The basic idea behind masking is to alter every secret data value s (e.g., a secret key or plaintext) with a mask m, resulting in the masked intermediate value $s_m = s * m$. The easiest and most prominent operation for that purpose is the exclusive-or ($\oplus$) operation, known as *Boolean masking*. Other typical operations to conceal the intermediate values are modular addition or modular multiplication. Then the resulting masking scheme is referred to as *arithmetic masking*.

Albeit masking is widely known as one of the two main DPA countermeasures (besides hiding), there is a more generic approach of it known as *secret sharing*. With secret sharing, the data to be concealed is split into d shares. Hence, masking with a mask m and a masked intermediate value s_m can be thought of a secret sharing scheme with two shares $s_1 = s_m$ and $s_2 = m$. While all but one of the shares need to be generated randomly, the last one is computed according to:

$$s_d = s_1 \oplus \ldots \oplus s_{d-1} \oplus s$$

Secret sharing based on two shares (i.e., masking) may seem to protect against first-order[2] DPA attacks at first glance. However, previous work [75] revealed that due to the presence of *glitches*, at least three shares are required to reach first-order resistance in hardware.

In contrast to hiding, secret sharing (or masking more specifically) suffers from the drawback that with an increasing number of shares, the required resources also increase correspondingly. Moreover, performance can suffer due to masking, since all of the applied shares need to be processed. Finally, it is important to note that while masking linear functions (e.g., using Boolean masking) can be easily achieved since $f(s \oplus m) = f(s) \oplus f(m)$, applying it to non-linear functions is usually not straightforward.

3.3 Keccak and the Sponge Family

An important group of cryptographic primitives, which received a lot of attention in the recent past, is the family of sponges [24]. In its original form, a sponge construction takes an input of arbitrary length and computes a fixed-length output as illustrated in Figure 3.2. It basically consists of the following two main phases, indicating the similarities to an actual *sponge*:

Absorbing phase: During this phase, the input data is *absorbed* into the state block by block. Handling an input block is done by XORing it to the first r bits of the internal state, followed by a call to the underlying permutation f.

[2]The order of a DPA attack refers to the number of intermediate values considered to build the hypothesis for guessing the secrets.

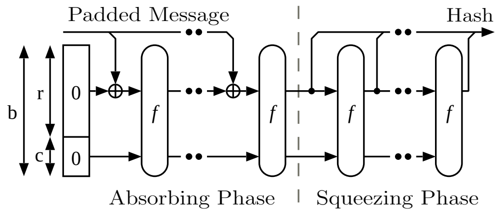


Figure 3.2: Sponge construction.

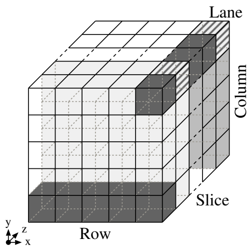


Figure 3.3: State

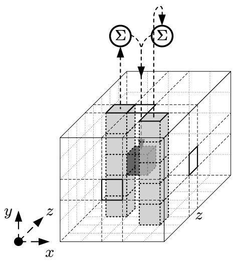
Squeezing phase: Once all input data has been absorbed, the sponge construction provides the output data by *squeezing out* one block after another. If longer output values than r bits are demanded, the state is fed into the permutation several times.

Originally the sponge construction was developed to be used as a hash function based on an iteratively re-used fixed-length permutation. Later, the family of sponges was subdivided into the sponge construction itself [21] and the closely related duplex construction [23, 26]. Since the core of both primitives is the permutation denoted by f, it is briefly discussed in the following section.

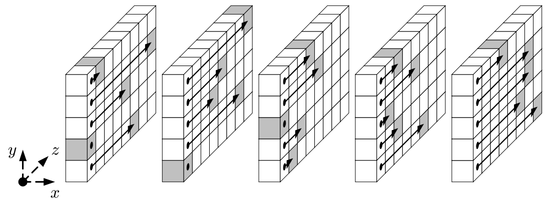
3.3.1 The Keccak-f Permutation

Keccak-f operates on a state with a fixed size of b bits. This state consists of two parts: First, the *rate* r, specifying the number of input bits, which are processed in one iteration. Therefore, r relates to the speed of the computation. Second, the last c bits of the state denote the *capacity*, determining the attainable security level of the construction. Hence, the length of the capacity is determined according to $c = b - r$.

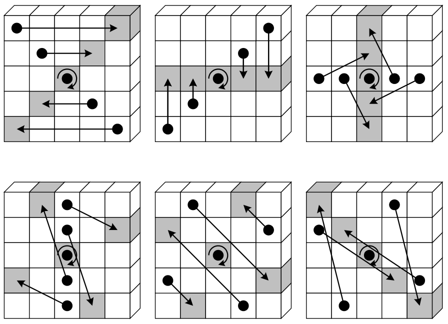
The authors of Keccak defined Keccak-f for the following seven state sizes: $b = 25 \times w$, where $w = 2^{\ell}$ and ℓ ranges from 0 to 6. The state is illustrated in Figure 3.3 and is organized as a $5 \times 5 \times w$ matrix with three dimensional coordinates (x,y,z). We call a set of w bits with given (x,y) coordinates a *lane*, a set of 5 bits with given (y,z) coordinates a *row*, 5 bits with given (x,z) coordinates a *column*, and



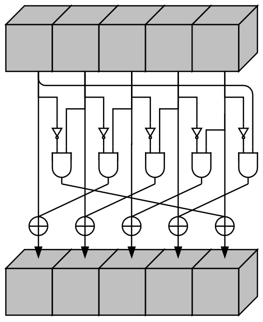
(a) θ



(b) ρ



(c) π



(d) χ

Figure 3.4: The round operations of the Keccak-f permutation [20].

the 5×5 matrix for a given (z) coordinate a *slice*. The Keccak-f function further consists of $12 + 2\ell$ rounds that are made up of five steps as depicted in Figure 3.4 and briefly discussed below.

θ **:** Provides diffusion by linearly mixing the parity of two nearby columns (from two neighboring slices) and adds it to a bit of another column.

ρ **:** Provides inter-slice dispersion by rotating all lanes by a predefined offset.

π **:** Breaks the horizontal and vertical alignment by transposing the 25 lanes according to a fixed pattern.

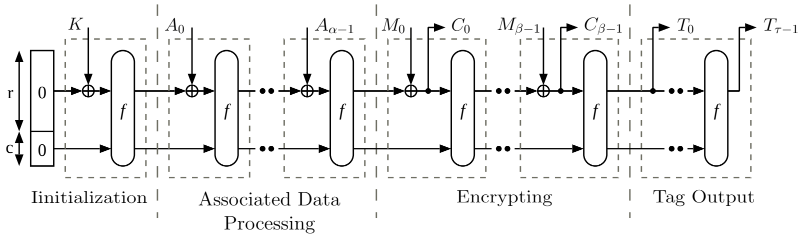


Figure 3.5: Simplified SpongeWrap authenticated encryption (also known as the *wrapping* operation).

χ **:** This represents the non-linear part of Keccak-f. The five bits of each row are combined using AND gates and inverters and the shifted result is added to the row.

ι **:** A w-bit round constant is added (XORed) to one of the lanes of the state.

For an in-depth discussion of the permutation and a description of the operations, we refer the reader to [25].

3.3.2 SpongeWrap

Several different permutation-based primitives have been published throughout the last couple of years. We are going to concentrate on a mode called SpongeWrap [23], which uses a duplex construction [26] and the previously described Keccak-f permutation to create an AE scheme. Figure 3.5 depicts the functionality of this mode, which works in four phases:

1. **Initialization:** During initialization, the state is cleared and loaded with the cipherkey K by a call to the permutation f.

2. **Associated Data Processing:** As part of this phase, Associated Data (AD) (i.e., data which must be authenticated but not encrypted) is absorbed into the state. An example for AD can be protocol header information, which needs to remain readable for routing devices.

3. **Encrypting:** The encryption step is quite similar to the second phase, with the only exception that for every processed plaintext block (i.e., data which needs to be authenticated and encrypted), the corresponding ciphertext is provided as an output as illustrated in Figure 3.5.

4. **Tag Output:** In the last phase, no input data is handled. Instead, the integrity-assuring authentication tag T is generated.

The process of absorbing a key K, the associated data A, and the plaintext P and squeezing out the ciphertext C and the authentication tag T is referred to as *wrapping*. The corresponding decryption step is known as *unwrapping* and takes a given K, A, and C and outputs P as long as T is correct. If the two tags do not match, an error will be dumped, but no plaintext will be provided.

3.3.3 Masking the Sponge

Devices using KECCAK-f in a keyed mode, including AE schemes such as SPONGEWRAP, must be protected against implementation attacks like DPA. Especially when they are part of key-distributing components (e.g., smart cards or other embedded systems) they can easily be accessed by the general public. Hence, countermeasures like hiding and masking become mandatory for such implementations.

The authors of KECCAK proposed to implement a secret sharing technique to protect keyed KECCAK instances based on three shares [22, 19]. Interestingly, Bilgin et al. [28] reported that the implementation of the masking of the non-linear χ operation presented in [22, 19] does not provide provable security and thus, is not secure against first-order DPA attacks. As a countermeasure, they proposed to inject fresh random bits in a 3-share implementation or to add an additional share (4-share version) that avoids the need of fresh randomness. Apart from the un-keyed KECCAK implementations available in literature [18, 66, 99], the smallest reported masking-secured designs so far require more than 30 kGE [19, 22, 28].

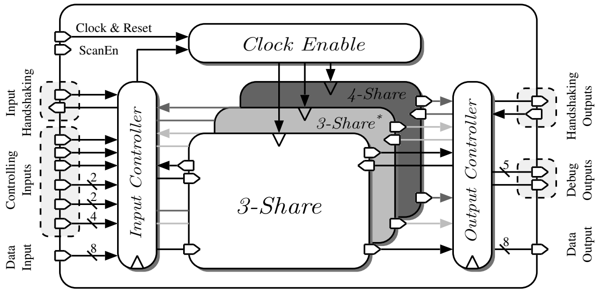


Figure 3.6: Top-level architecture of ZORRO.

3.4 ZORRO - An ASIC Assessment Platform for DPA Countermeasures

To assess hiding and masking countermeasures for a SPONGEWRAP AE primitive on a real ASIC, we went through the effort of designing a completely new Integrated Circuit (IC) exactly for that purpose. Our ASIC, called ZORRO [88], is intended to be used as an evaluation platform for investigating the quality of various DPA countermeasures for an AE system based on the KECCAK-f[1600] permutation.

We expected the SPONGEWRAP architectures, hosted on ZORRO, to be part of an IC used in resource-constrained devices such as smart cards or Radio-Frequency Identification (RFID) systems. Therefore, one of our major goals during the development of the SPONGEWRAP designs was, to keep the circuit complexity as small as possible. As discussed in Section 3.3.3, three different masking versions have previously been presented for the non-linear χ part of KECCAK. Hence, we decided to place three distinct architectures on ZORRO that only differ in their masking scheme. This allows us to analyze all three approaches independently from each other with respect to potential weaknesses. Figure 3.6 shows a block diagram of the top-level design entity of ZORRO, including the three distinct architectures named *3-Share*, *3-Share**, and *4-Share*. The differences of the three distinct designs regarding the masking of the non-linear χ function are listed

Table 3.1: Differences of the three independent designs on Zorro regarding the implemented masking scheme of the non-linear χ function. According to the number of shares, the size of the RAM varies.

Design	**Masking of χ**	**# Shares**	**Re-Masking**	**RAM**
3-Share	Bertoni et al. [22, 19]	3	$\times$	608×8
3-Share*	Bilgin et al. [28]	3	✓	608×8
4-Share	Bilgin et al. [28]	4	$\times$	816×8

in Table 3.1, including the two Random-Access Memory (RAM) versions employed on Zorro.

To assure that meaningful power traces can be measured for each architecture separately, the *Clock Enable* entity contains clock gating cells that drives the clock for the one entity selected exclusively. Moreover, the *Input Controller* forwards the input signals solely to the currently activated design, thereby avoiding any unwanted switching activity within the deactivated architectures (i.e., *silencing* has been applied to the disabled units). With this setup, we are able to obtain meaningful power measurements of each design without significant noise from the deactivated units. The *Output Controller* is responsible of forwarding the output signals of the respective unit once an input data block has been processed. With a few extra debug outputs, Zorro provides additional information about ongoing internal processes. Data to and from the chip can be transmitted via an 8-bit data bus, controlled by a four way handshake protocol. All three architectures can either operate as encryptor or decryptor. Moreover, each of them offers four different modes of operation with regard to the enabled DPA countermeasures:

- In ***Normal Mode (NM)***, no DPA countermeasures are enabled at all. Therefore, only parts of the RAM are actually used (since no shares are required) and the SpongeWrap construction works fully unprotected. Measurements based on this mode serve as a reference for the protected alternatives.
- When running in ***Hiding Mode (HM)***, the user can choose how many *dummy rounds* the architecture enabled should per-

form (up to 15). A single dummy round always corresponds to a full KECCAK round. Thus, the runtime in HM significantly increases when the number of dummy rounds is raised. Simultaneously, the data transfer to and from the RAM gets shuffled using eight different possibilities. Thus, the number of time instances ti, where the leakage can appear for a configuration using j dummy rounds, is calculated according to:

$$ti = 8 + 8 \cdot j, \quad j \in [1 \ldots 15]$$

For each RAM we reserved a couple of additional entries, which are not initialized. These words are used as inputs when executing the dummy rounds. Thereby no correlation between the actual state and the measured power traces should be observable at all. For the remainder of this chapter we use HM-j to denote ZORRO running in hiding mode with j dummy rounds.

- Once the ***Masked Mode (MM)*** is selected, the activated architecture actually operates on the shares. According to the designs' names, the *3-Share*, *3-Share**, and *4-Share* unit apply Boolean masking with the non-linear part of KECCAK masked as discussed in Section 3.3.3. In the following, we use MM-3 when talking about the architectures applying three shares and MM-4 in the case of four shares, respectively.

- The most secure mode, supported by each of the three architectures, is the ***Secure Masked Mode (SMM)***, which combines the countermeasures of HM and MM. Contrary to NM and HM, where only a third/fourth of the RAM entries are actually used (as well as the uninitialized entries for the dummy rounds), in MM and SMM all entries are required for processing. We further on refer to ZORRO running in SMM based on i shares and j dummy rounds using the following notation: SMM-i-j

3.4.1 *3-Share*, *3-Share**, and *4-Share* Designs

Since the *3-Share*, *3-Share**, and *4-Share* hardware architectures differ only very slightly in the masking scheme utilized, we will further on

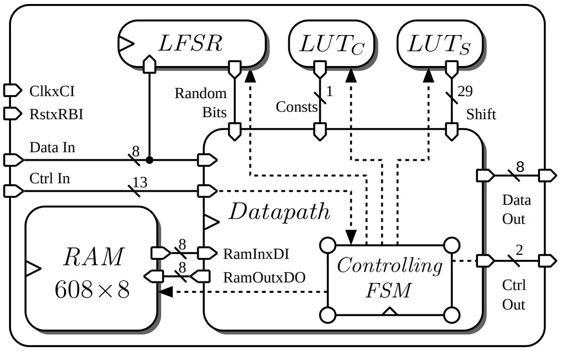


Figure 3.7: Design of the *3-Share* entity.

solely discuss the *3-Share* version and point out the differences to the other two architectures where necessary.

We aimed at designing a low-area, DPA-secure authenticated encryption system based on KECCAK. Because of these goals, the layout of the memory, required to store the KECCAK state, was of utmost importance. Moreover, the secret sharing countermeasure implemented works on the algorithmic level and thus, the required memory for the state increases with each share. We favored, therefore, a RAM macrocell over a register file built from Flip-Flops (FFs). We keep both the round constants of the ι function and the shift offsets of the ρ function in Lookup Tables (LUTs). Figure 3.7 illustrates the uppermost hierarchy level of the *3-Share* entity, including the state RAM, the LUTs, and the datapath entity, which gets controlled by a Finite-State Machine (FSM). Moreover, Figure 3.7 shows the Linear Feedback Shift Register (LFSR), constructed by the primitive polynomial given in equation (3.1).

$$x^{32} + x^7 + x^3 + x^2 + 1 \tag{3.1}$$

The LFSR gets initialized with an external seed. Its output is, on the one hand, required for determining whether to perform a dummy operation or not. On the other hand, the LFSR is needed to generate the random bits for the re-masking in the *3-Share** architecture. Overall, 42 random bits are required per input block (39 for the dummy operation conditions and three for shuffling RAM addresses).

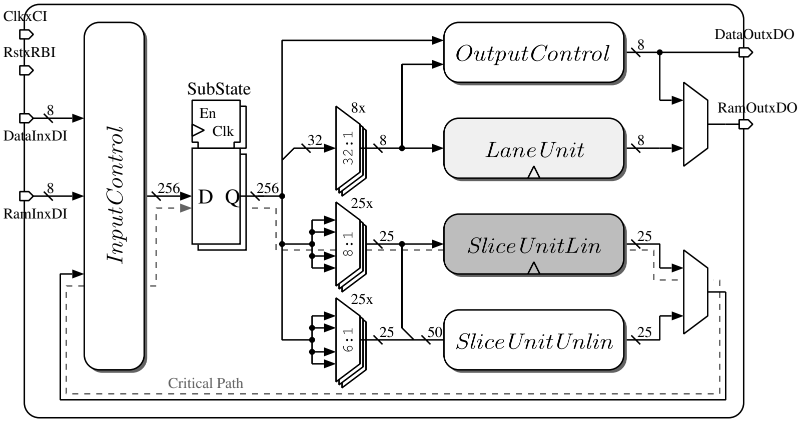


Figure 3.8: Datapath of the *3-Share* entity (controlling signals omitted).

The *3-Share* architecture contains a 608 × 8 RAM (cf. Figure 3.7) for storing state and shares. Basically, a secret sharing scheme for Keccak based on three shares would require only 4800 bits (three times the state size). We use the additional eight bytes of the RAM as inputs for the dummy operations during the hiding mode and therefore, keep these memory locations uninitialized. Thereby, none of the dummy operations computes on the actual payload of the chip and hence, no correlated power figures should be observed.

For the initial masking of the *3-Share* (*4-Share*) entity, the chip receives 3200 (4800) random bits to initialize two (three) shares followed by the plaintext. The last share is the plaintext XORed with the shares previously initialized. The implementation of the Keccak-f[1600] permutation is based on a combined lane and slice processing, similar to that proposed by Pessl and Hutter [99]. Figure 3.8 shows the architecture of the *Datapath* unit of the *3-Share* entity. We use the *SubState* register to buffer lanes and slices currently being processed.

3.4.2 RAM Allocation

As proposed by Bertoni et al. [18], storing the bits of lanes and slices in an interleaved form allows efficient processing of the data when choosing a small datapath width, meanwhile keeping the size of the required buffer register at a minimum. We also make use of this technique and store four bits of two slices in each RAM word (i.e., two bits of four lanes). Since we need four lanes at a time, this results in a buffer register of 256 bits. Unfortunately, the state consists of 25 lanes and thus, not all lanes can be stored in this interleaved form. We decided to store the first lane in a linear way, as this lane is not influenced by the ρ operation and hence, can be skipped for this function. With this memory allocation, we waste a small amount of clock cycles when loading data of the first lane. However, we can keep the size of the *SubState* register comparatively small. To avoid switching back and forth between slice-based and lane-based operations as much as possible, we make use of the rescheduling approach proposed in [99], where the authors distinguish between the following three different types of *rounds*:

$$R_1 = \theta \times \rho \qquad R_{2\ldots24} = \pi \times \chi \times \iota \times \theta \times \rho \qquad R_{25} = \pi \times \chi \times \iota \quad (3.2)$$

Appendix 3.A provides a more detailed description about the architecture of the Keccak-*f* operations. Moreover, we discuss the data transfer protocol with Zorro in Appendix 3.B.

3.5 Results

The results of our work are twofold. First, we present our hardware figures of Zorro and provide actual ASIC performance numbers of the *3-Share*, *3-Share**, and *4-Share* design. Second, we provide practical results of DPA investigations on our AE system using power traces obtained from the real chip.

3.5.1 Hardware Figures and Comparison

We used VHDL to code the Register-Transfer Level (RTL) model of Zorro and Mentor Graphics' ModelSim version 10.2a to verify functional correctness. Synthesis results were obtained from Synopsys' Design Compiler version 2012.06 for a mature 180 nm CMOS technology by UMC. The designs were synthesized using a standard cell library by Faraday Technologies under typical case conditions. Backend design steps were carried out with SoC Encounter by Cadence. Area results will be given in Gate Equivalents (GEs), where one GE equals the area of a two-input NAND gate of the target standard cell library (= 9.3744 µm^2).

In order to provide a fair comparison between the results of Zorro and related work as well as meaningful numbers for an actual chip to be taped out, we present two different area numbers. First, we provide synthesis results without considering any Design for Testability (DFT) techniques.[3] Second, we include the area numbers after all backend design steps have been successfully completed and therefore, the results include DFT circuitries for RAM tests as well as scan FFs to enable Automated Test Pattern Generation (ATPG).

Figure 3.9 provides an area/time (AT) plot of the synthesis results of the three different architectures (i.e., area numbers for different clock constraints of the designs). Based on the isolines, indicating a constant AT product, it can be observed that for a clock period below 4 ns, the area requirements of all architectures increase significantly. Moreover, the efficiency of the architectures in terms of the AT product does no longer improve. Knowing that timing data would somewhat deteriorate during the upcoming backend design, we chose a maximum frequency of 200 MHz for Zorro. The critical path of the design, running through the *SliceUnitLin* entity, is highlighted using a dashed line in Figure 3.8. From Figure 3.9 it can be observed that the relative areas of the three architectures remain quite the same. This was expected since a major part of the overall area is occupied by the RAM that scales proportionally with the number of shares. Other differences between the three designs with regard to their logic components are almost negligible. Table 3.2 lists an area breakdown of the Zorro ASIC after synthesis for a clock period of 5 ns. It shows that our *3-Share*, *3-Share**, and *4-Share* architectures require 13.4 kGE, 13.9 kGE, and 16.2 kGE, respectively. Table 3.3 provides

[3]Note that such numbers can depart significantly from those of a finalized chip ready for tapeout, depending on the implemented design.

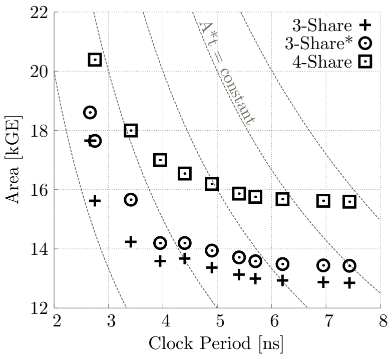


Figure 3.9: AT plot of Zorro's three different architectures obtained after synthesis.

Table 3.2: Area breakdown of the Zorro ASIC (results based on synthesis numbers at 5 ns).

Component	**Size [GE]**	**Size [%]**
3-Share	13 370	30.5
Datapath & FSM	7300	16.7
RAM	4680	10.7
LFSR	300	0.7
SliceUnitLin	480	1.1
Others	610	1.3
3-Share*	13 940	31.8
4-Share	16 190	37.0
I/O Interface	320	0.7
Zorro Total	43 820	100.0

a comparison between Zorro and related Keccak-based ASIC designs in the field of low-area and DPA-security. Two reasons make a sound comparison between the Zorro designs and related work somewhat difficult. First, because of the very generic reason summarized in Remark 3.1.

Remark 3.1 (No misleading comparisons)**.** *We avoid in-depth comparisons of our hardware architectures with results from previous work. The main reason for this is that we believe that such comparisons only make sense as long as the same target technology is being used. Even then, a fair comparison is almost infeasible, since too many aspects may diverge significantly during the development of the designs (e.g., standard cell library utilized, skills of designers, initial specifications, use of application, design priorities).*

Second, our goal was to provide a flexible evaluation platform for DPA countermeasures. Hence, additional resources were required to support enabling and disabling all the different modes of operation of Zorro. That overhead would, of course, not be required in a dedicated DPA-secure design of a Keccak-based AE scheme.

For the actual tapeout-version of Zorro, we added a couple of DFT circuitries to support testing. This, the insertion of the required

Table 3.3: ZORRO compared to related ASIC designs. Numbers are based on synthesis results.

Source	**Techn.** [nm]	**Size** [GE]	f_{max} [MHz]	Γ[†] [Cycles]
Designs w/o DPA Countermeasures				
Pessl and Hutter [99][‡]	130	5522	61	22 570
Bilgin et al. [28][§]	180	10 800	555	1600
ZORRO in Normal Mode[‡]	180	13 370	200	21 888
3-Share-Secured Designs w/o Re-Masking				
Bertoni et al. [22][§]	130	95 000	200	72
ZORRO *3-Share* Architecture[‡]	180	13 370	200	113 184
3-Share-Secured Designs w/ Re-Masking				
Bilgin et al. [28][§]	180	33 100	553	1625
ZORRO *3-Share** Archit.[‡]	180	13 940	200	113 184
4-Share-Secured Designs				
Bilgin et al. [28][§]	180	43 100	572	1600
ZORRO *4-Share* Architecture [‡]	180	16 190	200	149 640

[†] Cycles per data item for one KECCAK-*f* permutation

[‡] 1088 bit blocks § 1024 bit blocks

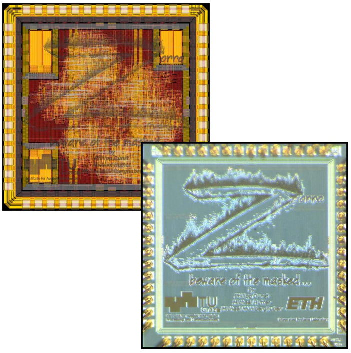


Figure 3.10: Chip layout and photo of ZORRO.

Table 3.4: Post-layout key properties of ZORRO.

Property	
Technology (UMC)	180 nm
Supply (Core/Pad)	1.8 V/3.3 V
Max. Frequ. (f_{max})	200 MHz
Required Size	45.5 kGE
Est. Power Cons. @ f_{max}	
3-Share	17.3 mW
3-Share*	19.7 mW
4-Share	20.8 mW
Cycles/data item (Normal/Masked)[1]	
3-Share	21 888/113 184
3-Share*	21 888/113 184
4-Share	21 888/149 640

[1] Requ. cyc. for one KECCAK-f perm.

buffers, and the fact that after the backend design a realistic wire-load model was available, lead to an increase in area to 14 kGE, 14.5 kGE, and 17 kGE for the *3-Share*, *3-Share**, and *4-Share* architectures, respectively. Figure 3.10 shows the final layout of ZORRO as well as a photo of the fabricated chip. Table 3.4 provides the key properties of ZORRO.

3.5.2 DPA Attacks on ZORRO

For analyzing the DPA resistance of ZORRO, we applied several analysis techniques to evaluate the applied countermeasures based on the following measurement setup.

Measurement Setup

To acquire power traces of ZORRO, the voltage drop across a 1 Ω resistor in the core supply line was measured with an *AP034* differential probe by *LeCroy*. This setup allows to minimize the noise created by, for instance, I/O activity of the chip because it features separate supply lines for both core and pads. A *PicoScope 6404c* oscilloscope was used to capture the power traces which were then stored on a

Personal Computer (PC) for further analyses. The ASIC was clocked with 10 MHz and a sampling rate of 1 GS/s resulted in 100 acquisition points per clock cycle. Zorro provides an 8-bit data bus for communication. Therefore, the measurement setup was basically quite similar to the one presented in Section 2.2.6. A controlling FPGA provided the following data, received from a PC, to the ASIC:

- Cipherkey, associated and message data
- Random numbers to initialize the shares
- Configuration data for the countermeasures

First Power Traces

The left plot in Figure 3.11 shows a measured power trace of an entire Keccak-f permutation of Zorro running in NM. It contains all 24 rounds separated by a dotted vertical line (including one additional round at the end where ρ is skipped). The right plot in Figure 3.11 depicts a zoom into the first round. We separated the slice and lane processing phases with a dashed vertical line as well as the eight slice-processing iterations (Zorro processes the 64 slices in eight blocks) by dotted vertical lines. The same was done for the six lane-processing sequences (the 24 lanes are handled in blocks of four). The time interval where the θ step of the first round takes place is of special interest because the power-analysis attacks presented next target the θ step. Only the first θ step was recorded for the power analysis attacks to keep the analysis effort reasonably small.

Detection of First-Order Leakage

As an initial step, we wanted to confirm that there are no first-order leakages when running Zorro in masked mode based on measurement results. For that, we applied Welch's t-test [47]. More specifically, we focused on the non-specific (fixed vs. random) t-test, which is part of a set of multiple leakage tests known as the *Test Vector Leakage Assessment (TVLA)* [12]. The t-test is used to decide if two groups G_A and G_B of measurements were drawn from populations with the same mean (*null hypothesis*) assuming a black-box model. For this decision, the t-value is calculated according to equation (3.3), where

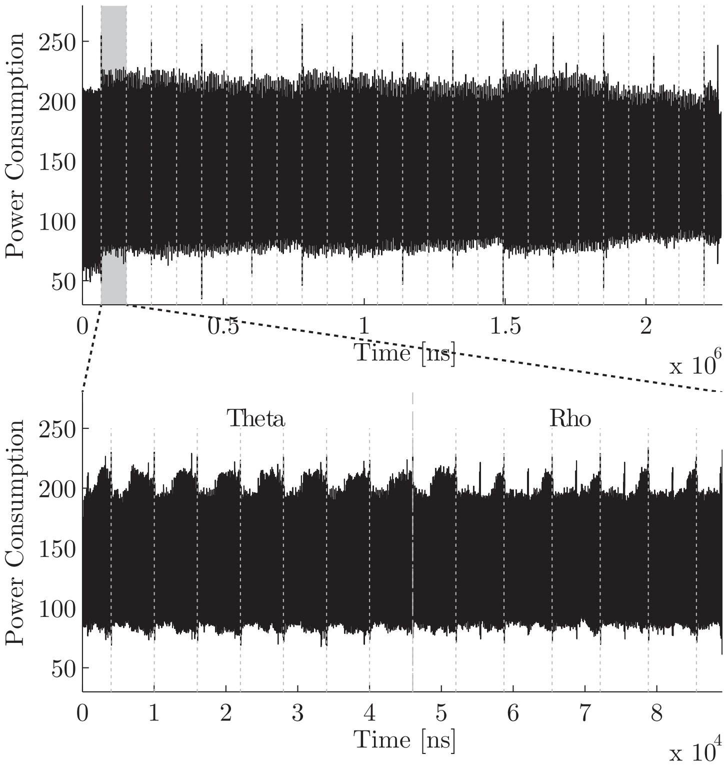


Figure 3.11: *Top:* Power trace of an entire KECCAK-f permutation while ZORRO is running in NM; *Bottom:* Zoom into the first round, computing θ and ρ.

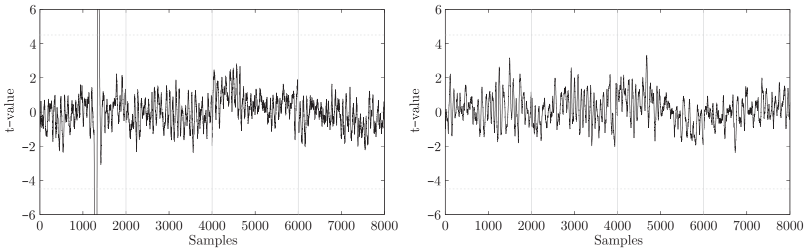


Figure 3.12: First-order leakage detection of the *4-Share* design. *Left:* Shares initialized with zeros; *Right:* Shares initialized with random values; vertical lines define the time intervals where the four shares are processed.

$\mu()$ denotes the sample mean, σ^2 the sample variance, and n_i the number of samples in the group i.

$$t = \frac{\mu(G_A) - \mu(G_B)}{\sqrt{\frac{\sigma^2(G_A)}{n_A} + \frac{\sigma^2(G_B)}{n_B}}} \tag{3.3}$$

If the value of t exceeds a specific threshold, the null hypothesis is considered false. According to the recommendations in [47], we have used a threshold value of ± 4.5. If t exceeds that value at any point during the computation of the algorithm, the null hypothesis can be rejected with 99.999 % probability.

As a first experiment, we verified that there actually is a first-order leakage within the fabricated ASIC by initializing the shares with zeros. To apply the fixed vs. random t-test, two groups of stimuli were generated for ZORRO. On the one hand, the plaintext was fixed to a certain input value to obtain G_A. On the other hand, random inputs were provided to create G_B. The result of this analysis for the four-share implementation is shown in the left image of Figure 3.12. Vertical lines in the plot separate the time intervals for the processing of the four shares. It is clearly visible that the t-value exceeds the ± 4.5 threshold during the execution of the first share. The outcome of the experiment is as expected, since the first share processes the input data without randomization due to the initialization of the remaining shares with zeros. Thus, there exists first-order leakage during the processing of the first share.

The same experiment has then been repeated with the shares initialized with random values. The right plot in Figure 3.12 depicts the result for this second experiment. 100 000 measurements have been used for each group. To avoid any influence of the measurement setup due to, for instance, temperature drifts, the traces for each group have been recorded in an alternating manner (i.e., 10 measurements for G_A, 10 measurements for G_B, 10 measurements for G_A, ...). It is clearly visible, that with this number of measurements no first-order leakage can be identified. The t-value does not exceed the ± 4.5 threshold at any time instance; the minimum and maximum values are -2.38 and 3.32, respectively.

The maximum of 100 000 available traces was determined mainly for two reasons. First, we wanted to keep the measurement effort reasonably small. Second, we assume that for most target applications of ZORRO, the number of encryptions can be limited to this value. Whether this is achieved by restricting the number of encryptions with the use of the communication protocol employed or by updating a potential session key frequently, is beyond the focus of this analysis. For the two architectures applying three shares, the same first-order leakage validation was conducted. The results were similar to those achieved for the *4-Share* implementation, i.e., no first-order leakage could be observed for up to 100 000 measurements when enabling the masking countermeasures.

Remark 3.2 (Interpretation of t-test results)**.** *The presence of a first-order leakage does not imply that key-dependent data is responsible for that leakage. It can therefore not be concluded that key bits can be extracted from the device with the use of Side-Channel Analysis (SCA) techniques. The opposite, however (i.e., not indicating any kind of first-order leakage), heavily suggests that the investigated device offers solid security boundaries for the number of traces being taken.*

Attack Scenario on ZORRO

To compare the effort required for a successful attack against the protected implementations on ZORRO, we conducted power measurements and applied standard Correlation Power Analyses (CPAs) based on the Pearson correlation coefficient [30]. For the rest of this section, ρ_c indicates the correlation coefficient of the correct key guess. CPAs presented herein focus on the first round of Keccak-f. In particular, we targeted a storage operation of the 256-bit *SubState* register (cf. Figure 3.8) that stores key-dependent intermediate values during the θ step. The decision to initially target the θ transformation was motivated by the modified round schedule given in equation (3.2). In the first round, θ is the only slice-based transformation, leading to a simple power model. For our attack, we assume the initial state $\boldsymbol{S}_{init}$ to be

$$\boldsymbol{S}_{init} = K||M||0^c\,,$$

where K refers to the cipherkey and M denotes a part of the state chosen freely by an attacker. The last c bits of $\boldsymbol{S}_{init}$ represent the initial capacity, which are all specified to be zero.[4]

With $r = 1088$ bits and $|K| = 256$ bits, the length of the message part in the initial state is 832 bits ($|M| = r - |K| = 1088 - 256 = 832$ bits). Furthermore, we use the following notations throughout the rest of this section:

$\boldsymbol{S}^j$	Current state of share j
S_z	Slice number z of state

In the case of a plain implementation without countermeasures, the initial state $\boldsymbol{S}_{init}$ is processed by the round transformations of Keccak-f. Each slice S_z contains four unknown key bits. The remaining 21 bits are known by the attacker and $21 - \frac{c}{64} = 13$ bits, the message part M, are expected to be freely chosen per slice. For a secret sharing implementation using d shares, the round transformations of Keccak-f are performed on the shares $\boldsymbol{S}^1$, $\boldsymbol{S}^2$, …, and $\boldsymbol{S}^d$ respectively. In the first execution step, the shares $\boldsymbol{S}^1 \ldots \boldsymbol{S}^{d-1}$ are initialized with random values and $\boldsymbol{S}^d$ is calculated according to $\boldsymbol{S}^d = \boldsymbol{S}_{init} \oplus \boldsymbol{S}^1 \oplus \cdots \oplus \boldsymbol{S}^{d-1}$. The fact that only random data is processed makes it hard for an attacker to generate hypothetical intermediate values that are required for a DPA attack.

[4]How a potential adversary actually achieves this use case is out of focus of this work. It might be due to an incorrectly implemented usage of Zorro within an upper-layer protocol or with the use of inserted faults.

Due to the modified round schedule given in equation (3.2), the θ transformation is the first linear transformation applied on $\boldsymbol{S}_{init}$. To minimize the resources required, the architectures on ZORRO perform θ on one slice per clock cycle, leading to a total of 64 clock cycles when running in NM and $d \cdot 64$ clock cycles for the d-share implementations. Note that the linear transformations can be applied on each share individually, which makes it possible to process shares sequentially.

As the θ transformation processes sensitive data, for which the key remains constant for every encryption, it becomes a potential target for a DPA attack. Preliminary experiments confirmed that ZORRO leaks intermediate values according to the Hamming distance power model. Since only one slice is processed per clock cycle, Hamming distances between 0 and 25 are possible. Moreover, due to the inherent iterative design of the architecture, the algorithmic noise is significantly smaller compared to an implementation where the whole state is transformed in one clock cycle.[5] To calculate the θ transformation for S_z, the information of two neighboring slices is required. Thus, each slice operation targeted reveals information about four key bits. Since θ processes two slices in parallel, we can efficiently target eight key bits (2×4 bits of two different slices) by evaluating the power models of 256 key hypotheses.

Normal Mode

We started our attacks running ZORRO in NM (i.e., with no countermeasures enabled). The number of power traces required to successfully accomplish this attack served as a basis for the experiments on our ASIC with with the countermeasures enabled. The CPA attack on the NM was performed with 1000 power traces, leading to $\rho_c = 0.7$. This ρ_c indicates that less than 50 measurements are sufficient to distinguish the correct from the wrong key hypotheses [74].

[5] In general, small datapaths—as within the ZORRO architectures—are actually tricky with regard to their side-channel resistance, since the noise level is lower compared to designs, where multiple operations are computed in parallel.

Hiding Mode

Next, we activated hiding on the Zorro ASIC. The number of dummy rounds has been set to one (HM-1), which means that zero or one dummy operation is randomly inserted in front of the first Keccak-*f* permutation working on real data. Moreover, as soon as hiding is activated, read/write operations from/to the RAM are randomly shuffled. As a result, the operation targeted can appear at 16 different time instances ti. According to [74], ρ_c should decrease by a factor of $\frac{1}{ti}$ compared to the unprotected case. When taking into account $\rho_c = 0.7$ from the unprotected case and $ti = 16$, this leads to an expected value for the protected implementation of

$$\rho_{c,theory} = \frac{\rho_c}{ti} = \frac{0.7}{16} = 0.044\,.$$

Practical attacks on the HM yielded $\rho_{c,pract} = 0.049$, which agrees with theory.

Next, we applied *windowing* [37] to counteract the hiding techniques. In general, windowing sums up the power consumption of all moments in time when the attacked intermediate value can appear. Hence, for HM-1 16 time instances had to be combined. As discussed in [74], this should increase ρ_c by a factor of $\sqrt{ti}$ for our attack. Table 3.5 lists ρ_c obtained from our practical measurements and compares them with the results from theory. It contains numbers for three different hiding modes, each analyzed with and without windowing applied. Most results from our experiments are in line with theory. The only exception is the significantly larger correlation coefficient obtained for HM-1 when windowing is applied. Without windowing, no significant correlation peaks were observed for HM-15 with up to 100 000 measurements. Hence, for HM-15 without windowing we only provide theoretical numbers in Table 3.5.

Masked Mode

In a next experiment we performed power-analysis attacks targeting the first θ step on Zorro running in MM (all hiding countermeasures deactivated). 1st-order CPA attacks using 100 000 power traces captured from the ASIC did not succeed. No significant correlation peaks could be observed.

Table 3.5: Results for the power-analysis attacks on ZORRO running in HM. $\rho_c = 0.7$ from the unprotected mode (NM) served as reference for the correlation coefficients.

Mode	***ti***[†]	**Wind.**[‡]	$\rho_{c,theory}$	$\rho_{c,pract}$
HM-1	16	no	0.044	0.049
HM-1	16	yes	0.176	0.237
HM-2	24	no	0.029	0.031
HM-2	24	yes	0.152	0.160
HM-15	128	no	0.005	-
HM-15	128	yes	0.062	0.057

[†] Number of different time instances *ti*
[‡] Windowing applied or not

However, due to the clear patterns in the power traces, the time instances, where the first θ steps of each share are performed, can be identified with little effort. By combining the revealed time instances, a 3rd-order CPA attack has been mounted. The centralized product combining, as suggested by Prouff et al. [100], has been used as combination function. As shown in Figure 3.13, this attack results in a significant correlation peak for the correct key hypothesis with $\rho_c = 0.016$. Figure 3.14 shows the course of the correlation coefficients for all key guesses. With less than 70 000 measurements the correct key hypothesis can be distinguished from the wrong key hypotheses. Note that since the modifications between the *3-Share* and *3-Share** implementation solely affect the χ step (and not the θ operation targeted herein), the results of the 3rd-order CPA are identical for both three-share based architectures.

Comparing Masking and Hiding Modes

Our analysis results confirm that both hiding as well as masking increase the effort for an attack. Attacks on the implementation using hiding also succeed without any modification of the traces (e.g., windowing). Attacks on the masked implementation do not succeed with-

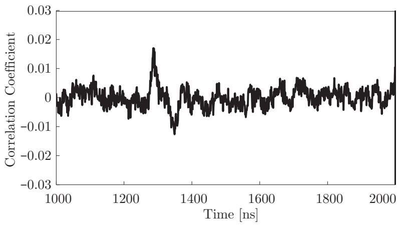


Figure 3.13: 3rd-order CPA result for the correct key guess using 100 000 ASIC traces (Zorro running in *masked mode*).

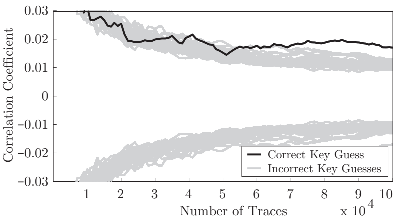


Figure 3.14: Course of the correlation coefficient of Zorro running in *masked mode* (3rd-order CPA).

out combination of the traces, at least not if the shares are calculated sequentially during the first θ step, as it is the case within Zorro. To reach the same security level with hiding as with masking, theoretically 240 dummy rounds would be required. Each additional dummy round leads to eight additional time instances for the targeted operation to appear, so $ti = 240 \cdot 8 = 1920$ for 240 dummy rounds. As a consequence, ρ_c would decrease to $\frac{0.7}{\sqrt{1920}} = 0.016$, which is equivalent to the ρ_c achieved with a 3rd-order CPA, targeting the masked implementation. However with 240 dummy rounds, the runtime of the implementation operating in hiding mode exceeds that of the masked mode significantly. To get around this drawback, a shorter dummy operation than the complete Keccak round can be chosen. Table 3.6 summarizes the number of required measurements N_{meas} for a successful key recovery. For the unprotected case (NM), less than 100 measurements are sufficient, for the hiding mode with the weakest protection HM-1, $N_{meas} = 285$, and for the hiding mode with the highest protection (HM-15), $N_{meas} = 4\,925$ respectively. Note that N_{meas} for HM-1, HM-2, and HM-15 all assume that windowing is applied. Successful attacks targeting the masked mode (MM-3) require 70 000 measurements. According to [74], 240 dummy rounds (HM-240) would also require $N_{meas} = 70\,000$ to obtain similar correlation coefficients. However, this mode is not supported by Zorro.

Table 3.6: Number of required measurements to successfully attack the different ZORRO modes.

Mode	N_{meas}
NM	<100
HM-1	285
HM-2	625
HM-15	4925
HM-240[†]	70 000
MM 3-share	70 000

[†] Theoretical value; not supported by ZORRO

3.6 Summary

DPA countermeasures are often analyzed on general-purpose processors using software implementations, FPGA platforms, or simulations only. During this chapter we have presented ZORRO, an ASIC developed and manufactured for the sole purpose of assessing DPA countermeasures on a real chip. We designed ZORRO to be applicable in systems with constrained resources such as RFID or smart cards. Thus, for the implemented AE primitives based on the KECCAK-f permutation, we aimed at low area as the primary design goal. Our smallest SPONGEWRAP architecture requires 14 kGE, including DFT circuitries, and offers several hiding and masking countermeasures that can be enabled and disable at will. Furthermore, we have presented initial DPA results, comparing the strength of the different countermeasures. We showed that with hiding or masking alone, ZORRO can be successfully attacked with 100 000 traces or even less.

Chapter Appendix

3.A Round Operations

When Zorro operates in NM, the four slice-based round functions of Keccak-f (θ, π, χ, and ι) are exclusively calculated in the *SliceUnitLin* within a single clock cycle for a whole slice. The applied round schedule requires to calculate the result of θ, $\pi \times \chi \times \iota \times \theta$, and $\pi \times \chi \times \iota$. As illustrated in the left image of Figure 3.A.1, all three operations can be accomplished within the *SliceUnitLin* with the use of bypass multiplexers. Calculations of the linear round functions of the Keccak-f permutation are equal for both the normal mode and the masking-secured modes. Here, each share can be computed in sequential order (e.g., in R_1 the theta step is performed three or four times sequentially to process the three or four shares, respectively).

Due to the fact that the non-linear χ function requires inputs from more than one share, the processing of this function slightly differs. For the hardware implementation of the *3-Share* architecture, we follow the approach presented by Bertoni et al. [22] and compute the result for two input slices in a single cycle within the *SliceUnitUnlin*

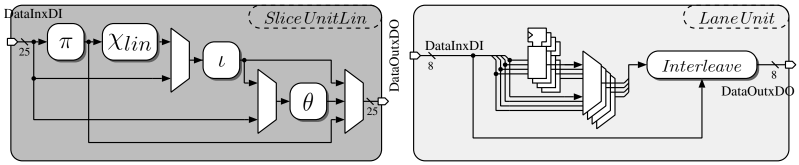


Figure 3.A.1: Datapath of the *3-Share* entity (controlling signals omitted).

entity. For the lane-based operation (ρ), we aimed at calculating its output byte by byte. This allows us to combine it with the RAM write operation. Thanks to the chosen RAM allocation, multiples of 2-bit-wide shift operations of lanes can easily be accomplished with the addressing of the memory. The special storage structure provides information about four lanes per RAM word (byte), and the *SubState* register can hold up to four lanes simultaneously. Unfortunately, each lane has a different shift offset. Hence, different bit pairs of the buffered lanes must be used to compensate the differences between the offsets. The different compensation offsets can be precalculated and are stored in the LUT_S entity (see Figure 3.7) for each lane quadruple. With these values, multiples of 2-bit-wide shift operations, and the offset between the different lanes can be compensated. What is left is a possible shift by one bit. Therefore, 4 one-bit-registers with surrounding multiplexers are used. If a lane is shifted by one bit, the high bit of the chosen bit pair is stored in the 1-bit register. The low bit is shifted one bit to the left and the old content of the one bit register is used as the new low bit. This is done for each bit pair of the buffered lanes. The result is stored back to the RAM in interleaved form. The responsible unit for the lane-based operation is called *LaneUnit* (cf. right image of Figure 3.A.1).

3.B Data Transfer Protocol

Due to the interleaved storage format of the lanes and the slices in the state RAM, data blocks must be provided to Zorro in a pre-defined order. Details about the data transmission follow. Assume we want to transfer the 1088-bit block of data below (given in hexadecimal representation, omitting intermediate bytes due to a lack of space, and each block representing one 64 bit-wide lane):

```
Lane 1:     1736712d8bc69dee
Lane 2-5:   b0dcd17a74223ffc aa08..3e0f f584..51a9 ad97..9bb9
Lane 6-9:   0557ea42de175cfe e478..39a1 12dc..0bc5 eb5a..31e0
Lane 10-13: e84abd82d8383d61 8b31..1482 63fe..2563 4e53..484c
Lane 14-17: 205fa600b47b9a0f 545c..2c44 7734..3c41 67ea..cb0e
```

Then the first lane is transmitted linearly byte by byte. After that, the next bytes to be transferred to ZORRO are constructed from the next four lanes. Therefore, let us write those four lanes in binary notation, omitting the last bytes as follows:

```
Lane 2: 1011 0000 1101 1100 1101 0001 0111 1010 0111 0100 ...
Lane 3: 1010 1010 0000 1000 0111 1010 0010 1100 1000 1110 ...
Lane 4: 1111 0101 1000 0100 0011 1000 0000 1010 0001 0010 ...
Lane 5: 1010 1101 1001 0111 0011 1011 1001 1000 0011 0100 ...
               ||
               |- Bit 0 from Lane 2 (L2B0)
               |- Bit 0 from Lane 3 (L3B0)
               |- Bit 0 from Lane 4 (L4B0)
               |- Bit 0 from Lane 5 (L5B0)
               |
               - Bit 1 from Lane 2 (L2B1)
               - Bit 1 from Lane 3 (L3B1)
               - Bit 1 from Lane 4 (L4B1)
               - Bit 1 from Lane 5 (L5B1)
```

Furthermore, let LiB0 and LiB1 denote the first and second bit of the first byte of the i-th lane ($i \in \{2 \ldots 5\}$), respectively. Then the next byte to be transmitted to ZORRO gets derived from the corresponding four lanes as follows:

```
L5B0 L4B0 L3B0 L2B0 L5B1 L4B1 L3B1 L2B1 = 1100 0010 = 0xC2
```

The next byte will be concatenated from the third and fourth bit of the first byte of these four lanes and so forth. Algorithm 1 describes the previously described transmission protocol more formally. Based on this transmission protocol, the interleaved storage of two bits of four lanes can be assured per RAM entry.

Algorithm 1 ZORRO data transfer protocol.

Input: Input block *Inp* with a length of 1088 bits. Note that we expect the bits to be transmitted from the very left (0-th bit) to the very right (1087-th bit).
Output: The byte to be transmitted to the ZORRO chip via signal `InxDI`.

```
$LaneQuad = 0$                                   ▷ Counter for lane quadruple
for $i = 0$ to 135 do                            ▷ Iterate through all 136 bytes of the block
    if $i < 8$ then
        ▷ The bytes of the first lane are transmitted one after another.
        InxDI[7:0] = $Inp[i \cdot 8 : i \cdot 8 + 7]$
    else
        ▷ Since the bits of a single transmitted byte are constructed from bits of four consecutive lanes, we must find out when a new lane quadruple (i.e., lane 2 to 5, lane 6 to 9, lane 10 to 13, and lane 14 to 17) starts.
        if $(i - 8) > 0$ && $(i - 8) \mod 32 = 0$ then
            $LaneQuad++$
        end if
        ▷ LSB of every fourth byte is determined different than the other LSBs.
        if $(i - 8) \mod 4 = 0$ then
            ▷ First, we must add the offset, which stems from the first lane.
            $lsb = 64$
            ▷ Next, we must add an offset of 256 bits (i.e., four lanes) in case we have already transmitted some quadruples previously.
            $lsb = lsb + LaneQuad \cdot 256$
            ▷ The LSB of the first byte of a new lane quadruple is the 6th bit of the first lane of the lane quadruple. Hence, we need to add an offset of 6.
            $lsb = lsb + 6$
            ▷ Finally, the offset for the respective byte must be added.
            $lsb = lsb + 2 \cdot ((i - 8) \mod 32)$
        else
            ▷ The LSB of the other bytes are computed from their predecessors.
            $lsb = lsb - 2$
        end if
        ▷ Compute the remaining bits of the current byte based on the LSB.
        InxDI[7] = $Inp[lsb + 3 \cdot 64 + 1]$
        InxDI[6] = $Inp[lsb + 2 \cdot 64 + 1]$
        InxDI[5] = $Inp[lsb + 1 \cdot 64 + 1]$
        InxDI[4] = $Inp[lsb + 1]$
        InxDI[3] = $Inp[lsb + 3 \cdot 64]$
        InxDI[2] = $Inp[lsb + 2 \cdot 64]$
        InxDI[1] = $Inp[lsb + 1 \cdot 64]$
        InxDI[0] = $Inp[lsb]$
    end if
end for
```

4 High-Throughput AEAD Architectures

Outline. In Section 4.1 we first give an introduction to CAESAR, the *Competition for Authenticated Encryption: Security, Applicability, and Robustness.* From that competition, potential successors for GCM—the current de-facto standard of authenticated encryption algorithms—should emerge. Throughout Section 4.2 related work about high-speed ASIC designs of GCM and CAESAR competitors is discussed. Predetermined specifications and requirements for our high-throughput ASIC architectures developed are presented as part of Section 4.3. Moreover, we briefly describe our own GCM design and the CAESAR architectures in this section. Thereafter, we provide our results in Section 4.4, including a comparison between all CAESAR designs and our GCM reference architecture. Eventually, a summary and a brief discussion is given in Section 4.5.

4.1 The CAESAR Competition

Confidentiality, integrity, and authenticity are three of the main cryptographic goals required for secure communication. In the past, researchers and engineers often tried to assure these goals separately, for instance, by using block ciphers and Message Authentication Codes (MACs). Some of these approaches resulted in severe security problems [69, 29, 113, 32, 96, 42]. Therefore, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [1] was launched in order to find combined algorithms, providing the desired service of Authenticated Encryption (AE) to prevent such problems. Simultaneously, the performance of today's algorithms should be increased.

The ultimate goal of the competition is not to find a single winner, but a portfolio of algorithms, supporting Authenticated Encryption with Associated Data (AEAD)[1]. These finalists are expected to serve as alternatives to the Advanced Encryption Standard (AES) [94] running in the Galois/Counter Mode of Operation (GCM) [45, 76], which currently represents a de-facto standard (further on referred to as *GCM-AES* or just as *GCM*). Another widely accepted mode of operation, which uses a block cipher to create an AE primitive and might be replaced by some of the CAESAR winners, is the CBC-MAC mode of operation [44]. The two algorithms have become highly popular because the National Institute of Standards and Technology (NIST) has officially recommended them. Since then, they have been adopted in technologies and protocols such as WiFi 802.11 [58] and IPSec [114].

CAESAR was initiated in 2014, with 57 candidates being submitted to the challenge. Throughout the first round of the competition the main focus of the evaluation process was to find cryptoanalytical weaknesses in the algorithms. Vulnerabilities found during such analyses usually lead to a *knock-out* of the respective participant. Moreover, a first comparison with regard to the software performance of the algorithms was accomplished based on their C reference code [2, 90]. Since the focus of those software models was clarity and not performance, results of these comparisons have to be taken with caution, as

[1]By *associated data* we refer to data that needs to be authenticated but not encrypted by the AE algorithm.

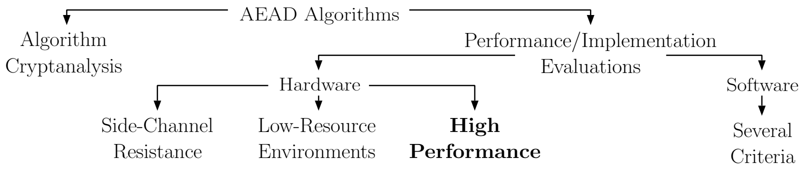


Figure 4.1: Potential evaluation criteria for the AEAD algorithms of the CAESAR competition, highlighting the investigated sub-criteria.

some authors may have optimized their implementations significantly more than others. Out of the 57 algorithms initially submitted, 30 made it to the second round. As of this writing, second-round participants are competing to proceed to the final third round, which should eventually result in the announcement of the winner portfolio at the end of 2017.

For the second round, the authors of the algorithms are required to provide reference hardware implementations as well. This should simplify evaluations of the candidates for different Application-Specific Integrated Circuit (ASIC) technologies and Field-Programmable Gate Array (FPGA) devices. Figure 4.1 shows a hierarchy of potential evaluation criteria that might be taken into consideration by the CAESAR committee throughout the selection process. It indicates that a variety of decision points may determine which competitors advance to the next round. Even within the group of hardware-related criteria, various target fields of application can be crucial for the selection process. We contribute to the competition by analyzing several of the candidates with regard to high-throughput ASIC architectures.

As a first step, we investigate the circuit complexity needed to reach a throughput of 100 Gbit/s under the *asymptotic* use case. This scenario assumes an infinitely large message to be processed by the AE algorithm, such that any processing required during initialization or finalization of a cipher, can be neglected.

Second, the high-speed architectures developed are analyzed in view of more realistic applications. We use Ethernet [54] as an example link-layer protocol and present the performances of the hardware architectures for different packet size distributions. We do not only

Table 4.1: Comparison of related work for high-speed ASIC architectures of GCM-AES.

Design	**AES Cores**	**Techn.** [nm]	**Size** [kGE]	f_{max} [MHz]	Θ [Gbit/s]	**Efficiency** [kbit/s/GE]
Yang et al. [119]	1	180	498.7	271.0	34.7	69.6
Satoh et al. [103]	4	130	979.3	317.5	162.6	166.0
Zhang et al. [120]	1	130	547.0	764.5	97.9	179.0
Mozaffari-Kermani et al. [83]†	8	65	630.0‡	613.0	25.1	39.8

† The most efficient architecture from [83] has been chosen. Numbers are given for n (input blocks) = 1. Interestingly, the throughput of their architectures decreases with the number of input blocks. We believe this to be some typo.

‡ The area of one GE in the cell library utilized ($2.08\,\mu m^2$) is significantly larger than that of one GE of our library ($1.44\,\mu m^2$). This may result in misleading comparisons when using GEs as a metric.

compare the chosen CAESAR candidates against each other, but also against a previously created GCM-AES reference architecture.

4.2 Related Work

Due to the popularity of GCM-AES, several high-speed hardware architectures have been presented in the past, targeting different ASIC technologies. Table 4.1 summarizes the results of those works and compares them in terms of throughput-per-area. Albeit all of the designs aimed at high performance, they basically differ from each other in three main points:

1. **Number of parallel cores:** To achieve the high throughputs, the architectures make use of different numbers of parallel AES cores. As a consequence, also the expected input widths and the structure of the Galois field multiplier vary.

2. **Supported key lengths:** While the GCM-AES designs by Yang et al. [119], Zhang et al. [120], and Mozaffari-Kermani and Reyhani-Masoleh [83] only consider the 128-bit version of AES, the work by Satoh et al. [103] also supports the 192-bit and the 256-bit version of the block cipher.

3. **Pipeline stages:** High-speed hardware implementations of AES typically unroll all rounds of the block cipher, separated by pipelining registers. Some designers apply pipelining even on the *sub-round* level to achieve the desired throughput. As a result, the circuit complexity due to the pipelining registers increases significantly.

Apart from the ASIC implementations of GCM-AES, a number of FPGA works have shown that high throughputs can be achieved with reconfigurable hardware as well [72, 121]. Speeds of 100 Gbit/s and even beyond were achieved on state-of-the-art FPGAs [51, 86, 87].

While hardware designs of GCM have been extensively studied, only few authors have addressed the hardware efficiency of the CAESAR candidates so far.[2] In the specification of Minalpher [102], the authors presented several architectures, reaching a maximum of 9.9 Gbit/s with a size of 16.7 kGE on a 45 nm technology. Šijačić et al. [105] provided further results on lightweight implementations of Minalpher as well as of PRIMATEs [7]. The first version of the submission document of SCREAM [49] contained various architectures, targeting a 65 nm process technology. The fastest design processes data with a throughput of 5.2 Gbit/s.[3] Initial hardware figures of POET [4] for ASICs and FPGAs were presented in [78], resulting in a maximum throughput of 768 Mbit/s for encryption, using a 180 nm technology by UMC. A low-area ASIC implementation of AEGIS [117] was presented by Schilling et al. [104]. Their design required 13.6 kGE based on a 130 nm technology, runs with a maximum clock frequency of 100 MHz, and achieves a throughput of 65 Mbit/s. In [48], Gross et al. presented a Threshold Implementation (TI) [92] and an unprotected

[2]This is mainly because the second round of CAESAR still comprises 30 candidates competing against each other, and the deadline for the Hardware Description Language (HDL) implementations is approaching only hesitantly.

[3]Unfortunately, the authors do not specify the standard cell library utilized, which makes a comparison even more difficult.

version of Ascon [43] on a 90 nm technology by UMC. Their fastest unprotected design needs 25.8 kGE and processes data with up to 13.2 Gbit/s. Chakraborti et al. [34] published a high-throughput design of TriviA, which achieved 91.2 Gbit/s on a 65 nm technology, requiring a size of 24.4 kGE.

Due to the reasons given in Remark 3.1 in Section 3.5.1, this chapter mainly provides results that compare the hardware architectures developed against each other but not with related work. The only exception to this rule is a brief comparison of our GCM-AES architecture with the results from previous work.

4.3 Assuring a Fair Comparison

As mentioned in Section 4.1, we focused our investigations on high-performance ASIC architectures. However, to allow an arguably fair comparison, further constraints must be defined in advance. Therefore, the next two subsections provide an overview of our environmental assumptions and general requirements that we have determined for the development of our hardware architectures.

4.3.1 Environmental Assumptions

Our environmental assumptions are briefly outlined below. A more comprehensive discussion can be found in Appendix 4.B.

Cipherkey changes frequency: We assume the cipherkey to change rather infrequently. Therefore, the processing time for doing so is expected to be negligible. Note that the architectural adaptations, required to get from a *hardly-ever-key-changing* design to a more *one-time-pad-like* approach, heavily depend on the algorithm utilized. Hence, no generic statement can be made on how much effort is required to adopt such modifications.

Data stream type: For a 100 Gbit/s transmission to be encrypted and/or authenticated, we expect to have what we call a *single-*

Public Message Number (PMN)[4] stream (sometimes referred to as a *single-message stream* in the literature as well). By that we mean a stream, where the actual majority of data stems from the Associated Data (AD)/message data of a single PMN. Therefore, the throughput cannot be increased by just adding multiple instances of an algorithm in parallel.

Data size availability: As for the available data size, we distinguish between two different use cases. First, we look at what we call the *data at rest* scenario, sometimes also referred to as *local* or *storage encryption* to design our 100 Gbit/s architectures. This scenario is similar to the asymptotic use case introduced in Section 4.1. We then use the developed designs to provide results on how these architectures perform under different *data in motion* applications based on Ethernet.

4.3.2 General Architecture Requirements

Our main goal for this chapter is to compare a state-of-the-art GCM architecture using AES-128 as the underlying block cipher with several CAESAR candidates. To do so, we determined a couple of architectural criteria, which had to be satisfied by all of our designs developed to ensure a fair comparison between the algorithms:

Supported cipher versions and modes: Many of the competitors offer multiple modes of operation. Supporting all of them would have represented an unnecessary overhead at this early phase of the competition. Thus, we decided to implement only the primarily recommended versions of the algorithms as found in the respective submission documents. Moreover, all designs had to support both encryption and decryption, thereby providing all features to enable a full communication.

Small I/O delays: All architectures developed should have short input and output delays compared to their internal operating clock

[4]During the remainder of this chapter, we denote any kind of *supporting data*, required for the AE modes (i.e., Initialization Vectors (IVs), nonces), by the more generic term *PMN*.

period. This is to ensure that candidate architectures can be integrated into larger systems without much timing hassles. By *short* we mean that several gate delays in an input or output signal path are not considered a problem. However, for longer I/O paths in the range of the clock period, registers have been added to cut the paths apart. Note that those registers might be removed if the data inputs/outputs of the architectures are directly attached to another sequential element.

Similar I/O interfaces: Since our architectures are expected to be used in streaming applications, we decided that all of them should communicate with their environment using the AXI4-Stream Protocol [8]. Let *source* be the name of any circuit that can feed one of our designs with data. A corresponding downstream circuit, waiting for data from our designs will be referred to as *destination*. Potential circuits for source and destination are First In, First Out (FIFO) buffers, that provide the data from/to a 100 Gbit/s stream. Appendix 4.C contains more details about the applied AXI interface.

Stallable architectures: Assuming that a downstream circuit is always ready to accept data from the AEAD architectures is not a practical assumption (a subsequent FIFO might be full or other circuits might not be ready to process the provided data). Hence, we decided that all of the algorithms must be stallable.

Technology-independence: In order to be able to synthesize all of the designs for different target platforms, we avoid using any technology-dependent components, such as memory macrocells. This should, on the one hand, allow a more meaningful comparison between the implemented candidates. On the other hand, this can also be beneficial when comparing our architectures against future upcoming ASIC implementations of the algorithms investigated. Moreover, this requirement is by no way restrictive as most memory macrocells suffer from longer read/write delays compared to Flip-Flops (FFs), which usually makes them undesirable for high-throughput applications.

4.3.3 Our Hardware Architectures

We decided to develop our own GCM-AES architecture from scratch. The resulting design should serve as a reference for the CAESAR candidate analysis. Although we could have used an existing GCM architecture available in literature as a basis, this would not have resulted in a fair comparison. Since a lot of the previously designed architectures did not have a certain goal in mind, an evaluation based on such a reference would have been almost meaningless.

Below, we provide a short description of our 100 Gbit/s architectures of GCM-AES and of our investigated CAESAR candidates. As of this writing, the CAESAR competition is in its second round and 30 candidates still compete against each other in the contest. Because of the multitude of algorithms submitted, we only chose a promising subset of competitors to actually evaluate them against the GCM-AES reference architecture. Architecting and implementing the five CAESAR candidate circuits along with the GCM-AES reference was not only indispensable for our investigations but also represented a substantial effort. Yet, we have decided to relocate the full details about the algorithms and the architectures developed to Appendix 4.D. Readers can so focus on our results and findings without being distracted by too many technicalities.

GCM-AES: Our GCM reference architecture is based on a single, fully unrolled AES-128 core, pipelined after each round. Since we assume that key changes take place rather rarely, we iteratively reuse the combinational logic required to derive one roundkey from another. The 11 roundkeys (incl. the original cipherkey) are stored in FFs. For both the cipher part and the key expansion of AES we adopted the Canright S-box [31], which is known to have the best area/timing trade-off from literature.[5] As for the authentication part, we make use of a completely combinational bit-parallel Galois field multiplier.

AEGIS and MORUS: Since the basic structure of AEGIS [117, 118] and MORUS [116] is quite similar, we discuss them together

[5]All CAESAR candidate architectures discussed in this chapter that employ (parts of) the AES round also use the Canright S-box.

hereafter. Our architectures developed for both of the algorithms are based on a fully combinational state update function. Although AEGIS uses eight AES rounds in its update function, no pipelining registers are required. This is because in contrast to AES, AEGIS processes a 256-bit input block in parallel through the eight AES rounds. Neither of the two candidates requires a distinct key expansion, because they handle the cipherkey as part of the state update function.

ICEPOLE: The round operation of ICEPOLE [82] has a rather short combinational delay. Hence, it turned out that a single instance of it in our hardware architecture is sufficient to achieve the asymptotic throughput of 100 Gbit/s. Therefore, our resulting design mainly contains one ICEPOLE round, a couple of registers for the state and the I/Os, and some glue logic.

NORX: The core components of our NORX architecture are eight G permutations, which make up one F function. Since NORX64-4-1 is based on four F functions, four clock cycles are required to process a complete 768-bit input block. Besides the G permutations, mainly the registers for the state and the I/Os add up to the overall circuit complexity. Our architecture serves as the official reference hardware design for the second round submission of NORX and is available from [84].

Tiaoxin − 346: Similar to AEGIS, the update function of Tiaoxin − 346, denoted by Update, is based on several AES round function calls. For our architecture, we have implemented the state update function fully combinational. One iteration of Update requires six AES rounds. As the AES rounds are processed in parallel, no pipelining registers are needed.

4.4 Results and Comparison

All circuit architectures have been developed in VHDL and have been verified against their C reference implementations using Mentor Graphics' Questa Sim 10.4c. Synthesis numbers are based on results from Design Compiler 2015.06 by Synopsys using a 65 nm CMOS

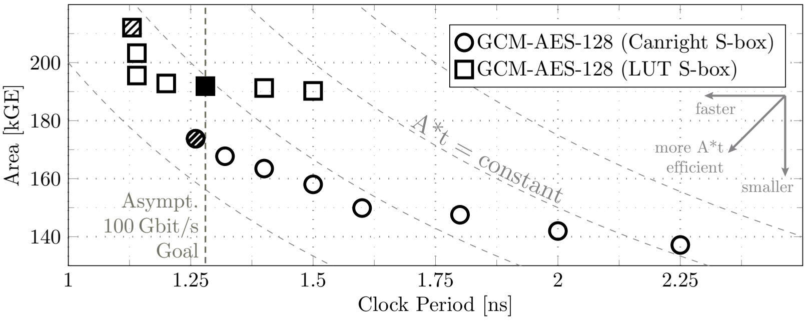


Figure 4.1: Required area of the final GCM-AES architectures for different clock constraints (AT plot). Numbers are based on post-synthesis results using a 65 nm standard cell library by UMC. The vertical dashed lines indicate the asymptotic 100 Gbit/s goal.

technology by UMC and a standard cell library from UMC under typical conditions. Area numbers are given in terms of gate equivalents (GE), where one GE equals the size of a two-input NAND gate of the standard cell library utilized (= 1.44 µm^2).

We present our results in three different sections. First, we will provide the results of our GCM-AES reference architecture. Next, our CAESAR designs developed will be compared with each other and with the GCM architecture for the data at rest use case. Eventually, all designs will be analyzed for the data in motion scenario using Ethernet as an example protocol.

4.4.1 GCM-AES Reference Architecture

Figure 4.1 shows an AT plot of our GCM-AES core based on synthesis results. We provide numbers for two different types of the AES S-box. First, we created the S-box using a straightforward constant array in VHDL (further on denoted as *LUT version*). With this approach all the effort for implementing the logic of the fully-combinational S-box is shifted over to the synthesis tool. Second, we obtained results based on the Canright [31] S-box, which uses subfields to implement the required GF(2^8) inversion and is known to have a better area/timing trade-off. While the composite-field S-box according to Canright requires less area compared to the LUT implementation, it is a little bit slower, which might increase the critical path of the AES-128 design and thus, that of the overall GCM architecture undesirable.

As for the clock period, we consider two different points of interest for our analysis:

1. On the one hand, we look at the clock period required to reach the asymptotic throughput of 100 Gbit/s. This point of interest is further on referred to as the *100 Gbit/s performance.*

2. On the other hand, since most of the designs can be clocked faster than what is required for a throughput of 100 Gbit/s, we also look at the absolute maximum clock period. Results given for this point of interest will be denoted by *maximum throughput performance.*

Remark 4.1 (Maximum throughput performance)**.** *The results for the asymptotic 100 Gbit/s performance represent the actual goal we had in mind during the development of our architectures. The maximum performance variants have been obtained by imposing different timing constraints during the synthesis process, the Register-Transfer Level (RTL) code remained the same.*

We mainly provide the numbers for the maximum throughput performance because of two reasons. First, they might become of interest if even higher throughputs than 100 Gbit/s are demanded for the asymptotic use case. And second, by pushing the clock frequency towards its maximum, we can achieve throughputs of 100 Gbit/s and beyond also for finite message lengths (as discussed in the following sections).

The AT plot in Figure 4.1 indicates that the 100 Gbit/s architecture of the Canright version of GCM-AES (the circular marker in Figure 4.1 lying almost on the vertical dashed line) shows the best results in terms of the AT product (indicated by the curved, dashed lines, which mark a constant product).

Remark 4.2 (Backend design considerations)**.** *Note that tight synthesis results such as those for the GCM-AES architecture based on the Canright S-box might become problematic (or even impossible to* *achieve) throughout an actual backend design run. However, since our main goal in this work is a comparison based on synthesis results and to compare the algorithms against each other, we believe these numbers provide a reasonable basis for further investigations.*

Basically, the LUT version of GCM-AES runs faster than its Canright counterpart (cf. Figure 4.1). However, it does not provide any advantages in terms of throughput-per-area. Therefore, we have just listed it in case throughput is of utmost importance for the target application.

Table 4.1 summarizes our GCM-AES results and compares them against previous work on high-speed ASIC architectures. The last column, entitled f_{max}/f_{100} *Ratio*, indicates how much faster our architecture can be clocked compared to the frequency required for the 100 Gbit/s asymptotic throughput. As such, it can be seen as an indicator of how well a resulting architecture meets its asymptotic throughput requirement. From Table 4.1 it becomes obvious that our GCM-AES architecture developed significantly outperforms previous designs.[6] One of the reasons is that our design is not based on multiple cipher and finite-field multiplier stages to reach a high throughput, which contrasts with previous work. Instead, it merely uses a single instance of each of them. Moreover, thanks to the fully-combinational Galois field multiplier, no additional large registers are needed for the authentication part of GCM. The majority of 128-bit-wide registers are only needed for pipelining the AES-128 cipher, the roundkey memory, and for registering inputs and outputs.

As a result, our GCM-AES architecture serves as a sophisticated reference for the CAESAR candidate designs to be developed on the one hand. On the other hand, it represents the most efficient high-speed ASIC design in terms of throughput-per-are available to date.

4.4.2 Data at Rest

As for the data at rest use case, we investigate three different aspects of our hardware architectures developed. First, we present a compar-

[6]Recall that it is not in our interest to compare results from distinct technologies, which is why we do not delve too deep into the comparison with related work here.

Table 4.1: Comparison of our synthesis results for the GCM-AES implementations targeting a 65 nm ASIC technology with related work. Numbers of our architectures are given for both an asymptotic throughput of 100 Gbit/s and the absolute maximum achievable throughput.

Design		**Max. Throughput Performance**†					**100 Gbit/s Performance**					f_{max}/ f_{100} **Ratio**
	Techn.	**Size**	f_{max}	**Θ**	**Efficiency**		**Size**	f_{100}	**Θ**	**Efficiency**		
	[nm]	[kGE]	[MHz]	[Gbit/s]	[kbit/s/GE]	[%]‡	[kGE]	[MHz]	[Gbit/s]	[kbit/s/GE]	[%]‡	
Our Work												
GCM (Canr.)	65	173.8	793.7	101.6	⊘ 584.6	**100**	170.8	781.3	100.0	⊘ 585.6	**100**	1.02
GCM (LUT)	65	211.2	885.0	113.3	▨ 536.3	92	191.9	781.3	100.0	■ 521.1	89	1.13
Related Work												
Yang et al. [119]	180	498.7	271.0	34.7	69.6	12	-	-	-	-	-	-
Zhang et al. [120]	130	547.0	764.5	97.9	179.0	31	-	-	-	-	-	-
Satoh et al. [103]	130	979.3	317.5	162.6	166.0	28	-	-	-	-	-	-
Mozaffari-Kermani et al. [83]§	65	630.0	613.0	25.1	31.1	5	-	-	-	-	-	-

† The absolute maximum throughput, reached by pushing the clock frequency to its maximum.

‡ Relative efficiency in terms of throughput-per-area compared to the GCM-AES architecture (Canright S-box).

§ The most efficient architecture from [83] has been chosen. Numbers are given for n (number of input blocks) = 1. Note that their throughput decreases with an increasing number of input blocks.

ison of the designs in terms of cycle count and latency. Second, we use throughput-per-area as a metric to analyze the efficiency of our architectures. Finally, we compare the performance of our designs, considering throughput as a function of the message size.

Comparison in Terms of Cycle Count and Latency

The cycles required to process a certain data item, denoted by Γ, and the latency L are two basic properties of a hardware architecture. Most obviously, Γ needs to be small for high-speed designs. Although L must not necessarily be small to reach high throughputs, we tried to minimize it as far as possible. The main reason for this is that with a small latency, the number of required pipelining registers in the datapath can be minimized as well. As a result, the efficiency in terms of throughput-per-area can be maximized. In our analysis, we distinguish between the following types of Γ, referring to the number of clock cycles required to process ...

Γ_K:	... the cipherkey K.
Γ_{Init}:	... the public message number (i.e., IVs or nonces) and for the initialization phase.
Γ_A:	... a block of associated data.
Γ_M:	... a block of message data (i.e., plaintext or ciphertext).
Γ_{Fin}:	... the last input block and for the finalization phase. The last input block can be a block of AD or message data, or some other input required by the AEAD algorithm (e.g., the lengths of A or M).

As for the latency of the architectures, we differ between the following two values. Let L_M denote the number of clock cycles from an input block being entered into the AE architecture until the corresponding result becomes available at the output [63]. Furthermore, let L_T refer to the number of clock cycles between accepting the last input block and providing the authentication tag T. Table 4.2 lists the different types of Γ and L for our architectures developed. Since we focused on the asymptotic throughput, Γ_K, Γ_{Init}, and Γ_{Fin} play a minor role for this phase of the analysis. However, we tried to minimize Γ_A and Γ_M to reach the desired goal of 100 Gbit/s.

Table 4.2: Cycles per data item Γ and latency L for all data at rest architectures developed as part of the CAESAR competition.

Design	**Blk. Size**	**Cycles per Data Item**					**Latency**	
		Γ_K	Γ_{Init}	Γ_A	Γ_M[†]	Γ_{Fin}	L_M	L_T
	[bit]	[cycle]					[cycle]	[cycle]
Reference Architecture								
GCM-AES	128	11	11	1	1	1	2	2
CAESAR Candidates								
AEGIS-128L	256	10	10	1	1	7	2	8
ICEPOLE	1024	12	12	6	6	12	8	14
MORUS-1280-128	256	16	16	1	1	8	2	9
NORX64-4-1	768	8	8	4	4	4	5	5
Tiaoxin − 346	256	15	15	1	1	20	2	21

[†] Γ_M is usually the same for accepting data items at the input as for releasing data items at the output.

The results from Table 4.2 indicate that all CAESAR candidates investigated are based on larger block widths than GCM-AES. Especially the permutation-based competitors ICEPOLE and NORX operate on much wider blocks, which allows them to process the data in multiple clock cycles. A major disadvantage of all CAESAR designs compared to the GCM reference architecture is the higher number of clock cycles needed for the finalization (Γ_{Fin}). This becomes of particular interest when investigating the throughput for finite message data lengths (i.e., when Γ_{Init} and Γ_{Fin} will have a decisive impact to the overall performance).

Throughput-per-Area Metric

We now provide results of our architectures in terms of throughput-per-area based on synthesis numbers. Table 4.3 lists all designs, including area and throughput figures for a 65 nm CMOS technology by UMC. The efficiency is given as an absolute and as a relative value compared against the GCM-AES reference architecture.[7]

[7]Note that Appendix 4.E provides synthesis results in the form of an AT plot for every core component of each of the investigated candidates (i.e., update function, permutation, etc.) and the overall algorithm. These plots may serve as a

Table 4.3: Synthesis results for the GCM-AES reference implementation and the CAESAR candidates investigated. Numbers are given both for an architecture that achieves an asymptotic throughput of 100 Gbit/s and for a second architecture that maximizes throughput. Data refers to a 65 nm ASIC technology by UMC and a standard cell library by UMC under typical conditions.

Design	**Sec. Lvl.**[§]	**Blk. Size**	**Max. Throughput Performance**[†]					**100 Gbit/s Performance**				**f_{max}/ f_{100} Ratio**
			Area	**f_{max}**	**Θ**	**Efficiency**		**Area**	**f_{100}**	**Efficiency**		
	[bit]	[bit]	[kGE]	[MHz]	[Gbit/s]	[kbit/s/GE]	[%][‡]	[kGE]	[MHz]	[kbit/s/GE]	[%][‡]	
Reference Architecture												
GCM	128	128	173.8	793.7	101.6	584.6	**100**	170.8	781.3	585.6	**100**	1.02
CAESAR Candidates												
AEGIS128-L	128	256	107.6	724.6	185.5	1724.2	295	62.9	390.6	1589.2	271	1.88
ICEPOLE	128	1024	96.8	1351.4	230.6	2383.4	408	61.4	585.9	1627.6	278	2.31
MORUS-1280-128	128	256	51.4	1818.2	465.5	9052.6	1548	32.7	390.6	3058.2	522	4.65
NORX64-4-1	256	768	57.4	595.2	114.3	1992.0	341	45.2	520.8	2214.9	378	1.14
Tiaoxin − 346	128	256	104.2	598.8	153.3	1471.3	252	70.9	390.6	1410.4	241	1.53

[†] The absolute maximum throughput, reached by pushing the clock frequency to its maximum.
[‡] Relative efficiency in terms of throughput-per-area compared to the GCM-AES architecture.
[§] Note that security levels refer to confidentiality of the plaintext exclusively.

From Table 4.3 it becomes apparent that all CAESAR algorithms require significantly less area than GCM-AES to achieve the goal of 100 Gbit/s. Consequently, the candidates are also way more efficient in terms of throughput-per-area than the current de-facto standard. They outperform GCM by a factor of 2.4 to 5.2.

The last column of Table 4.3 provides the ratio between f_{max} and f_{100}. Thus, it can be interpreted as a metric for *how good* the critical path fits the clock period, needed to achieve the asymptotic throughput of 100 Gbit/s. It indicates that several candidate architectures can be clocked much faster than what is needed for the asymptotic goal. Therefore, it may seem appropriate to build alternative designs based on multiple instances of the core components of the algorithms. However, this would increase the circuit complexity required and, as a result, would decrease the efficiency. Hence, such architecture transformations are not considered in our work.

As for the maximum throughput performance, the efficiency of all CAESAR candidates, except that of NORX, increases even further in comparison to GCM-AES. MORUS should be particularly emphasized here, as it achieves an efficiency 15 times better than that of GCM. This is mainly due to its very simple state update function, which allows clock frequencies of up to 1.8 GHz.

Area and throughput trends: Figure 4.2 provides synthesis results near the architectures' maximum frequencies for several clock constraints and the resulting throughput and area numbers. It depicts the trend of the efficiency for all of our architectures developed. The dashed lines indicate a constant efficiency and therefore, provide a good overview about how the designs perform in terms of throughput-per-area near their maximum operating frequencies. Figure 4.2 confirms our findings that most authors seem to have taken into consideration throughput-per-area as a metric for optimizing their algorithms. We identified three major reasons how the investigated CAESAR candidates achieved this performance gain compared to GCM:

1. **Hardware-friendly round functions:** Some candidates are based on round functions as simple as a couple of logic gates.

starting point for deriving performance results for other architectures than the ones presented herein.

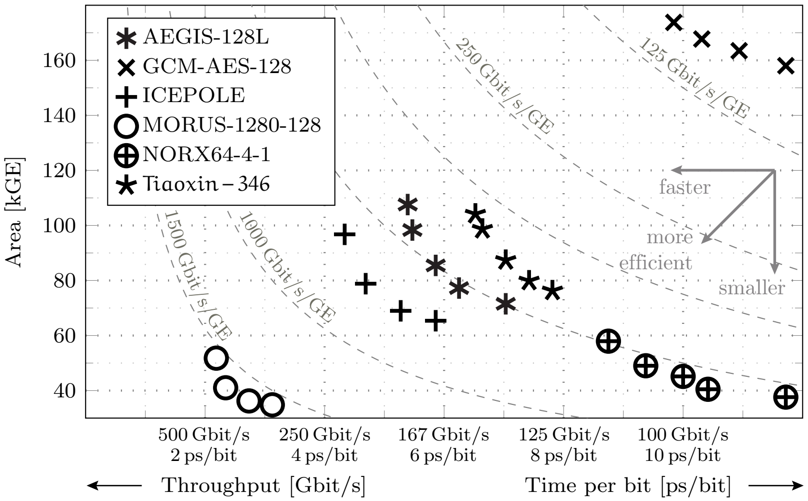


Figure 4.2: Synthesis results of all CAESAR candidate architectures in terms of throughput and area occupation when operated near their maximum clock frequencies.

Such highly hardware-friendly approaches allow to clock the architectures at a very high rate (or compute multiple rounds in a single clock cycle). Since simultaneously the circuit complexity can be kept small, the resulting designs are quite efficient in terms of throughput-per-area.

2. **Avoid separate authentication components:** GCM requires two distinct constructs to provide the goal of authenticated encryption (i.e., a block cipher and a Galois field multiplier). The CAESAR candidates, however, avoid using separate components for encryption and authentication. This is reflected in the circuit complexity of the architectures developed.

3. **Parallel round processing:** Similar to GCM, some of the candidates are based on full AES rounds. Though, they do not iteratively process the input blocks through the AES rounds. Instead, they handle them in parallel. Therefore, no pipelining

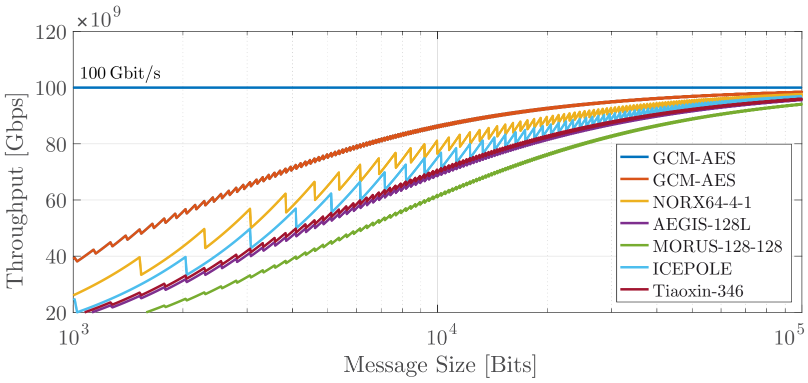


Figure 4.3: Throughput results for different message sizes of the 100 Gbit/s performance architectures.

registers are required, which would increase the circuit complexity significantly.

Throughput as a Function of the Message Size

The numbers presented so far do not take into account the actual length of the message to be encrypted, but assume very long input data. For the 100 Gbit/s asymptotic figures, theoretically an infinitely long message is required to compensate the overhead, caused by initialization and finalization phases. Such assumptions are common in related work when designing high-throughput hardware architectures. Nevertheless, the reader should be aware that the targeted 100 Gbit/s throughput will practically never be reached when only looking at the designs for f_{100}. This is illustrated in Figure 4.3, showing the throughput of the 100 Gbit/s architectures as a function of the message size. The throughput plotted in Figure 4.3 can be computed according to

$$\Theta = \frac{|Message|}{\Gamma_{Message} \cdot t_{lp}},$$

where $\Gamma_{Message}$ denotes the clock cycles required to process the whole message and t_{lp} the longest path of the architecture, respectively. As

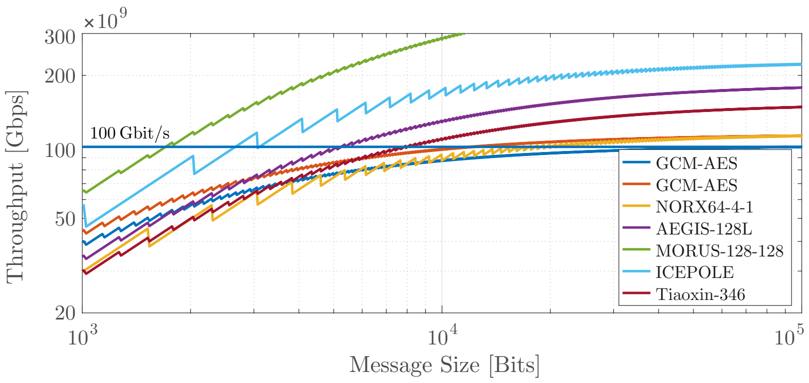


Figure 4.4: Throughput results for different message sizes of the maximum throughput architectures.

opposed to the results presented in the previous section, the numbers in Figure 4.3 now take into account the time required for initialization and finalization. As a result, the 100 Gbit/s designs converge against the target throughput, but will practically never reach it. Obviously the designs with time-consuming initialization and finalization phases (cf. Table 4.2) perform much worse when running with f_{100}. For instance, for a message length of 10 kbit, GCM-AES achieves a throughput of 85 Gbit/s. MORUS, however, barely passes 60 Gbit/s.

On the other hand, when considering the maximum throughput at f_{max}, those architectures that can be clocked much faster than what is required for the asymptotic throughput goal, stand out significantly. Figure 4.4 indicates that this especially applies to MORUS and ICEPOLE. The former already achieves a throughput of 100 Gbit/s for a message size of approximately 180 bit.

4.4.3 Data in Motion

We now present how the CAESAR candidate architectures developed perform when considering typical types of communication protocols. On the lowest level, we investigate link encryption using Ethernet

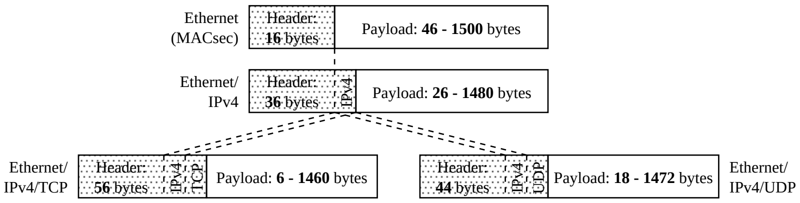


Figure 4.5: Typical data sizes for different types of communication protocols. The shaded parts represent the header, to be processed as AD, while the payload is expected to be authenticated and encrypted.

with a frame format according to IEEE 802.1AE.[8] Furthermore, we provide results for IPv4 using both TCP and UDP as the transport-layer protocol. Figure 4.5 shows how much associated data needs to be authenticated for the different protocols and how much message data is required to be authenticated and encrypted for typical frame structures.

Payload Size of Interest

In order to determine what payload sizes should actually be considered for the data in motion analysis, we investigated typical packet size distributions of various protocols. High-throughput hardware implementations can often be found in data center communications as required for enterprises dealing with a huge amount of data or university campuses. As for the packet size, related work [98, 16, 15] indicates that most of the layer-2 and layer-3 traffic shows a bimodal distribution. Also for more application-specific scenarios, such as video streaming, this bimodal distribution is confirmed [5], although the two peaks move further and further apart. We analyzed our AEAD architectures developed for the following packet size distributions (plotted in Figure 4.6):

Ethernet: Compared to the other investigated packet size distributions, this distribution describes the overall packets sent via the

[8]For this section, we expect the reader to be familiar with the Ethernet basics provided in Appendix 4.A.

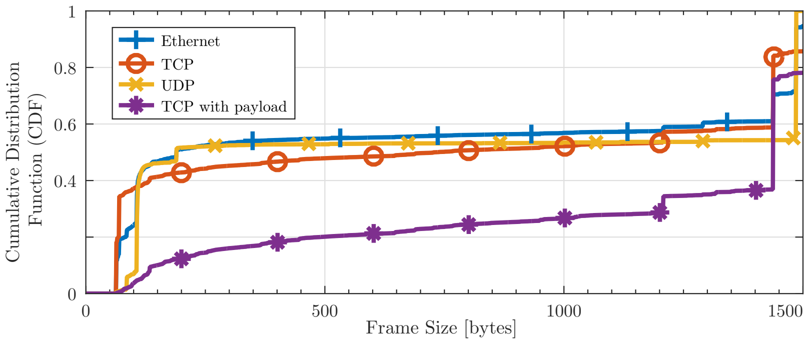


Figure 4.6: Typical bimodal distribution of packet sizes for Ethernet, TCP, and UDP communications between data centers. Source of the plotted data is a university campus [15].

link layer. Since the only layer-2 protocol used in the traces captured is Ethernet, this is equal to the overall Ethernet traffic.

TCP: Only the TCP packets are considered for this type of packet size distribution.

UDP: Although not as ubiquitous as TCP, we also present numbers of a UDP-only packet size distribution.

TCP with payload: This distribution contains TCP packets with a non-empty payload exclusively.

In addition to the packet size distributions provided in Figure 4.6, we also include numbers for the following packet sizes in our analysis:

Maximum-size Ethernet frames: It is assumed that only maximum size Ethernet packets are transmitted (i.e., frames with a payload of 1500 bytes).

Jumbo frames: Compensating the time-consuming initialization and finalization phases of ciphers can naturally be done by increasing the size of the actual payload. Hence, we also provide numbers for the use case where maximum-size Jumbo frames (i.e., frames

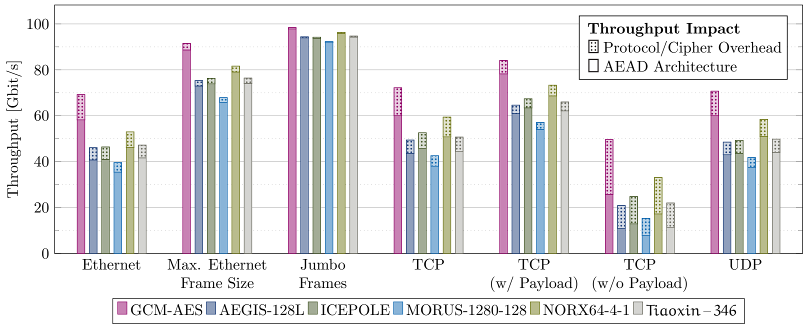


Figure 4.7: Throughputs for the hardware architectures developed of the AE ciphers taking into consideration the header lengths and the frame size distributions shown in Figure 4.5 and 4.6, respectively. The given numbers apply for the asymptotic 100 Gbit/s designs.

with a payload of 9000 bytes) are sent between the communicating parties exclusively.

TCP without payload: A huge amount of today's communication is due to very small packets without payload at all. Typical examples are acknowledgment (ACK) and synchronization (SYN) as well as finish (FIN) frames. Therefore, we also consider TCP packets with an empty payload in our analysis. Note that neither encrypting nor authenticating these small packets might be reasonable if their confidentiality and integrity is of no importance.

100 Gbit/s performance: We have analyzed the throughputs of all hardware architectures developed for the previously presented frame sizes. Figure 4.7 illustrates the numbers when running the designs with f_{100}. Basically, two different values can be obtained from it for each of the candidate architectures and the GCM-AES reference design. First, the throughput achieved by the actual cipher architectures themselves. And second, the throughputs reached when considering the overall protocol (i.e., taking into account the additional performance due to protocol and cipher overheads as discussed in

Appendix 4.B.2). The latter is represented by the dotted areas in Figure 4.7.

The numbers confirm the results obtained in the last part of Section 4.4.2. The CAESAR candidate architectures suffer from their significantly longer initialization and finalization phases compared to the GCM design. As a result, they perform worse than GCM-AES, especially for typical Ethernet, TCP, and UDP communications, since here also a lot of small frames are considered.

While the throughput converges against the 100 Gbit/s threshold for all of the algorithms when assuming Jumbo frames exclusively, GCM-AES is the only AEAD architecture that achieves 50 Gbit/s with a typical bimodal *Ethernet* frame size distribution. In case of a TCP communication without payload (*TCP w/o Payload*), MORUS does not even reach a throughput of 10 Gbit/s when looking at the performance of the AEAD architecture only. However, as indicated by Figure 4.B.2 in Appendix 4.B.2, the throughput roughly doubles for such small Ethernet packets when taking into account the overhead due to the protocol and the cipher. Recall that for the numbers presented, we assumed header data sizes (i.e., the associated data to be processed by the ciphers) as given in Figure 4.5. The remaining data within the Ethernet frames was expected to be payload data that needs to be both authenticated and encrypted. Although the provided header sizes represent typical values for the respective protocol, in practice they may diverge slightly depending on the application being used.

Maximum throughput performance: The results for the algorithms running at their absolute maximum frequencies (i.e., the *maximum throughput performances*) are given in Figure 4.8. We can see that MORUS is the only cipher that actually reaches the 100 Gbit/s threshold for all frame sizes, except for the TCP packets without payload. However, recall that this implies that the MORUS architecture must be clocked with 1.8 GHz, which is a rather optimistic goal when actually taping-out the design for the 65 nm technology targeted.

The data in motion analysis has shown that the significantly higher efficiency in terms of throughput-per-area of the CAESAR algorithms,

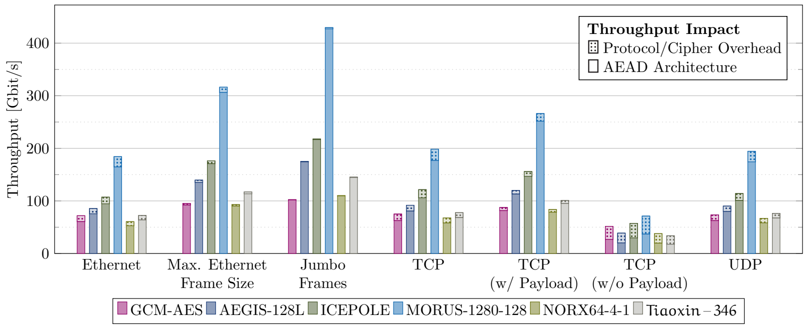


Figure 4.8: Throughputs for the hardware architectures developed of the AE ciphers taking into consideration the frame size distributions shown in Figure 4.6 and header lengths as illustrated in Figure 4.5. The given numbers apply for the maximum frequency designs.

observed during the data at rest evaluation, cannot directly be transferred to a real-world communication protocol such as Ethernet. Although some candidates still outperform GCM when operated with their maximum clock frequency, the factor of improvement is significantly smaller compared to the throughput-per-area comparison.

4.5 Summary and Discussion

In this chapter we have compared five second-round candidates of the CAESAR competition with a previously designed GCM-AES reference architecture. Our goal was to achieve an asymptotic throughput of at least 100 Gbit/s. The results from our analysis indicate that all competitors investigated are significantly more efficient in terms of throughput-per-area than GCM when looking at very long input messages. They outperform the current de-facto standard by a factor of up to 15. This is mainly achieved because of much simpler algorithm structures, which allow higher operating clock frequencies meanwhile the circuit complexity remains comparatively low.

When considering finite message lengths, however, it turned out that due to their long finalization phases, the advantages of the investigated CAESAR competitors decreases substantially. Some of them even fall behind GCM-AES.

In general, we do not believe that any candidate architecture that suffers from an internal feedback loop will ever significantly outperform counter-mode-based algorithms like GCM for short messages. All of our CAESAR designs presented in this chapter suffer from this drawback. For those of them, where the feedback loop is only present in the initialization or finalization phase, fully unrolling all of the required iterations might be appropriate. It remains an open question if this approach results in hardware architectures that are still competitive with GCM in terms of throughput-per-area. We expect that minor improvements by a factor of 2–5 will not convince industry to immediately switch from a well established algorithm like GCM-AES to any of the emerging newcomers.

Chapter Appendix

4.A Ethernet Revisited

Since Ethernet [54] is used as an example protocol throughout Chapter 4, we briefly recall the basics in this section. Encryption on layer 1 or 2 of the Open Systems Interconnect (OSI) model, usually referred to as *link encryption*, is a prominent way to get around the often complex protocols on the overlaying network layers.

Much like Fibre Channel (FC), Synchronous Optical Networking (SONET), Synchronous Digital Hierarchy (SDH), and others, Ethernet belongs to the fastest-growing communication protocols available today. With link speeds of 40 Gbit/s and 100 Gbit/s [56], it is suitable for applications such as data-center connections and high-bandwidth backup solutions. Mainly due to its relative efficiency and economy, Ethernet is the predominant protocol on the market. Therefore, AEAD algorithms, that want to play a major role in future mass markets, must be suitable for encrypting sensitive Ethernet communications efficiently too.

Recall an Ethernet traffic as depicted in the top image of Figure 4.A.1. Besides the actual packet, an Ethernet communication contains an Interpacket Gap (IPG) with a length of at least 12 bytes. See the middle image of Figure 4.A.1 for the format of the Ethernet packet, which consists of the actual frame, the preamble, and the Start Frame Delimiter (SFD), often considered to be part of the preamble. As can be observed from the bottom image of Figure 4.A.1, we have omitted the optional Virtual Local Area Network (VLAN) tag as defined in IEEE 802.1Q [57] for our investigations, since we solely consider the simplest Ethernet frame possible.

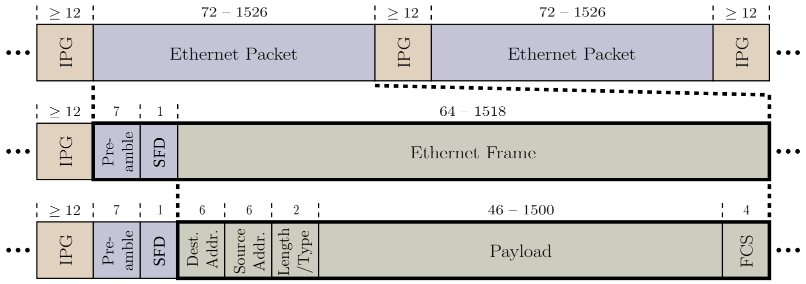


Figure 4.A.1: *Top:* Ethernet traffic including Interpacket Gap and Ethernet packets; *Middle:* Ethernet packet format; *Bottom:* Ethernet frame format; Both packet and frame formats are given according to IEEE 802.3 [55] and sizes are provided in bytes.

4.A.1 IEEE 802.1AE or MACsec Standard

In order to compare the AEAD algorithms on the protocol level, we followed the Ethernet frame format provided in IEEE 802.1AE [53], also known as the *MACsec* standard. As illustrated in Figure 4.A.2, MACsec extends the header of the original Ethernet frame with the *SecTAG* field and adds the Integrity Check Value (ICV) between the payload and the Frame Check Sequence (FCS). The SecTAG field contains, among others, the Packet Number (PN) and the Secure Channel Identifier (SCI). These two elements are of special interest when investigating AEAD algorithms since they contain the 96-bit IV required for encrypting a frame using GCM. Moreover, the third byte of the SecTAG element, hosting the TAG Control Information (TCI) and the Association Number (AN), contains a bit denoted by E, indicating whether the payload should just be authenticated, or authenticated and encrypted. Currently GCM-AES is the only cipher suite defined in the MACsec standard, and hence used by default. Therefore, Figure 4.A.2 also depicts how the IV and the authentication tag for GCM are mapped into the Ethernet packet. For detailed information about the frame formats, we refer the reader to IEEE 802.3 [55] and IEEE 802.1AE [53].

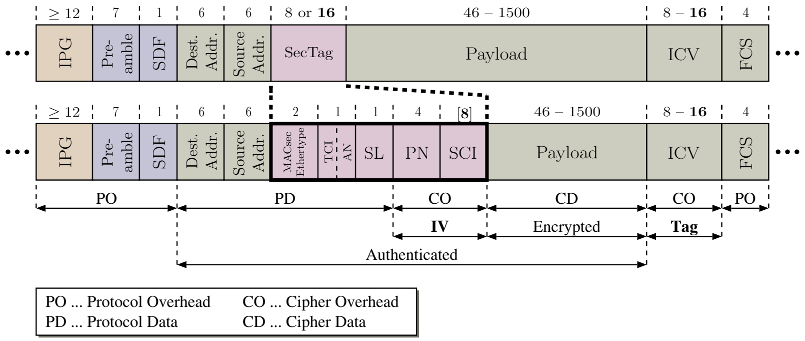


Figure 4.A.2: *Top:* Format of an Ethernet packet according to IEEE 802.1AE [53]; *Bottom:* Structure of the *SecTAG* element; Bold numbers/strings indicate the sizes/properties used for GCM-AES (the default cipher suite of IEEE 802.1AE).

4.B Environmental Assumptions (Extended Discussion)

In Section 4.3.1 we only provide on overview about our environmental assumptions. Below we discuss two of them, namely the data stream type and the available data size, more thoroughly. Because we use Ethernet as an example protocol for the data in motion use case throughout the remainder of this section, ensure you are familiar with the Ethernet basics given in Appendix 4.A.

4.B.1 Data Stream Type

In contrast to single-PMN data streams, as investigated in this work, for so-called *multi-PMN* streams the throughput can be increased just by adding multiple instances of the respective design. The difference between the two types is illustrated in Figure 4.B.1. Examples for multi-PMN applications include client/server systems, where a server has to process multiple client streams. Related work often suggests that the approach of multi-PMNs is a straightforward way to achieve the desired throughput for a cryptographic algorithm. The effort re-

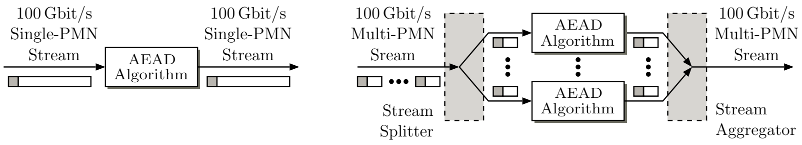

100 Gbit/s Single-PMN Stream
AEAD Algorithm
100 Gbit/s Single-PMN Stream
100 Gbit/s Multi-PMN Sream
Stream Splitter
AEAD Algorithm
AEAD Algorithm
Stream Aggregator
100 Gbit/s Multi-PMN Sream

Figure 4.B.1: Single- (left) vs. multi-PMN (right) stream applications

alizing the entities denoted *Stream Splitter* and *Stream Aggregator* in Figure 4.B.1 should not be underestimated however. Especially synchronization of the different data streams and their merging can become challenging.

4.B.2 Data Size Availability

We distinguish between two use cases that differ in the available data size. First, we look at what we call the *data at rest* scenario, sometimes also referred to as *local* or *storage encryption* to design our 100 Gbit/s architectures. We then use the developed designs to provide results on how these architectures perform under different *data in motion* use cases based on Ethernet.

Data at Rest

In this scenario, usually large amounts of data are available at one location (i.e., the size of the AD and message data is expected to be huge). Thus, the processing time for initialization with the PMN as well as finalization (to generate the authentication tag), becomes negligible compared to the actual data processing time (i.e., it is similar to the previously mentioned asymptotic use case scenario).

Observation 4.1 (Practicability of data at rest)**.** *Although many references on hardware architectures for AEAD algorithms assume a data at rest scenario, not many real-world applications exist.*

As a consequence, data at rest may serve as an initial benchmark. However, without determining the frequency of cipherkey and PMN changes and the size of AD and message data, it becomes difficult to create sound comparison results.

Data in Motion

As a second use case, we investigate the performance of various hardware designs of the AEAD algorithms under the *data in motion* scenario, using Ethernet as an example. We assume a frame format as discussed in Section 4.A.1. A few more assumptions have been made:

Assumption 4.1 (Single-type payload)**.** *For our Ethernet investigations, we only consider situations where the payload is either just authenticated or authenticated and encrypted, but do not consider any split versions as specified in the current MACsec standard [53][9].*

Based on the frame format according to IEEE 801.AE and Assumption 4.1, we can conclude that there are a minimum of 16 bytes of associated data stemming from the Ethernet header that need to be authenticated for every frame. Data within the payload of the frame can either just be authenticated (if the bit E in the header is equal to zero) or authenticated and encrypted ($E = 1$).

Assumption 4.2 (Fixed PMN length)**.** *The current MACsec frame format does not provide any elements for PMNs larger than 96 bits. Therefore, we assume all of the AEAD algorithms to make use of a 96-bit PMN for our analysis. Although many of the CAESAR candidates specify a 128-bit PMN for the primarily recommended cipher versions, only such an assumption allows a sound comparison. Because a 96-bit PMN might be concatenated with a 32-bit counter value in order to reach the recommended 128 bits, also this assumption should not pose a significant restriction at all.*

As a result, the Ethernet frame format given in Figure 4.A.2 is used as a basis for computing the required throughput for all AEAD designs for the data in motion use case.

Protocol throughput: The overall throughput figure of an Ethernet connection includes the actual data to be encrypted by the AEAD

[9]The current MACsec standard allows cipher suites to specify a so-called *confidentiality offset*, which declares up to the first 50 bytes of the payload of an Ethernet frame to be only authenticated and the remaining payload to be authenticated and encrypted.

algorithms as well as the PMNs transferred. Moreover, the Ethernet header (to be treated as associated data by the ciphers) and the protocol overhead must be taken into consideration. Therefore, we distinguish between the following four different types of data (cf. bottom image of Figure 4.A.2):

Protocol Overhead (PO): The overhead due to Ethernet, which must not be processed by the ciphers at all, but contributes to the overall protocol throughput.

Cipher Overhead (CO): In contrast to the PO, the cipher overhead is needed by the AEAD candidates and includes PMNs and authentication tags.

Protocol Data (PD): Parts of the Ethernet header (i.e., source and destination address and several header bytes) have to be authenticated for every frame. We refer to these bytes as *Protocol Data.*

Cipher Data (CD): The actual payload, which needs to be handled by the AE algorithm is denoted as *Cipher Data.*

With these four data types, the throughput for a communication based on MACsec frames can be computed according to

$$\Theta = \underbrace{\frac{|PO| + |CO|}{\Gamma_{Frame} \cdot t_{lp}}}_{\substack{\text{Must not be processed} \\ \text{by the AEAD algorithm}}} + \underbrace{\frac{|PD| + |CD|}{\Gamma_{Frame} \cdot t_{lp}}}_{\substack{A \text{ and } M \text{ to be processed} \\ \text{by the AEAD algorithm}}} , \tag{4.1}$$

where Γ_{Frame} and t_{lp} denote the number of clock cycles required to process the corresponding Ethernet frame and the longest path in the hardware architecture investigated, respectively. Γ_{Frame}, on the other hand, can be determined using

$$\Gamma_{Frame} = \Gamma_{Init} + \left\lceil \frac{|A|}{BW_A} \right\rceil \cdot \Gamma_A + \left\lceil \frac{|M|}{BW_M} \right\rceil \cdot \Gamma_M + \Gamma_{Fin} , \tag{4.2}$$

where Γ_A and Γ_M represent the number of clock cycles required to process a block of associated and message data, respectively. BW_x denotes the number of bits that can be processed simultaneously by a cipher architecture for input data type x. The value Γ_{Init} refers

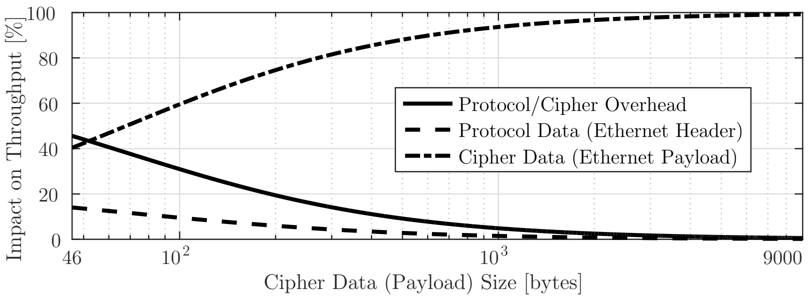


Figure 4.B.2: Impact of the different Ethernet communication parts on the actual throughput as a function of the frame payload size, assuming a frame format according to Figure 4.A.2.

to the number of clock cycles required for initialization, such as processing the PMN.[10] Eventually, Γ_{Fin} represents the number of clock cycles required for finalization, usually needed for AEAD algorithms to generate the authentication tag.

Payload size impact: One might suppose that the larger the payload of an Ethernet frame becomes, the larger the achievable overall throughput becomes. Although this assumption can be considered correct with regard to the AEAD cipher architecture, it is often neglected that the throughput for protocols gets determined by the overall data being transmitted. Figure 4.B.2 illustrates the impact of the protocol and cipher overhead to the overall throughput compared to the actual cipher data. Especially for small payload sizes, such as 46 bytes, the throughput roughly doubles due to the overhead, which sums up to 52 bytes for an Ethernet frame as depicted in Figure 4.A.2. As payload size increases, the impact of the AEAD architecture outweighs that of the protocol/cipher overhead and the overall performance gets largely determined by the throughput of the cipher engine.

[10]Recall that we ignore the number of clock cycles required to process the cipherkey as we assume key exchanges take place infrequently and, thus, can be neglected.

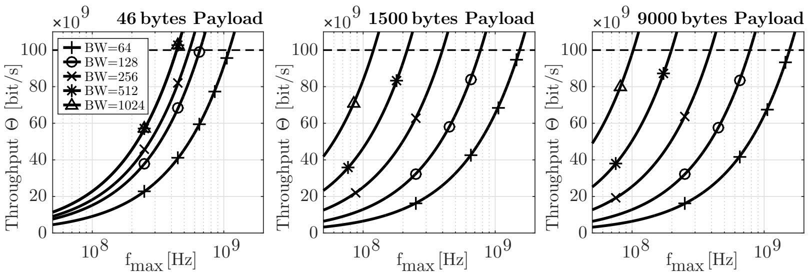


Figure 4.B.3: Throughput as a function of the maximum clock frequency for Ethernet using a MACsec frame format. The given numbers assume an AEAD architecture with $\Gamma_{Init} = \Gamma_A = \Gamma_M = \Gamma_{Fin} =$ 1 cycle. Results are plotted for different block widths (BW) and Ethernet payload sizes; *Left:* Payload = 46 bytes; *Center:* Payload = 1500 bytes; *Right:* Payload = 9000 bytes

Observation 4.2. *Although a hardware architecture for an AEAD algorithm may not reach the 100 Gbit/s for very small input sizes by itself, when put into a protocol like Ethernet, it may reach the desired throughput of the communication due to the protocol/cipher overhead. However, the larger the payload becomes, the higher the impact of the actual cipher architecture becomes and hence, predominantly determines the overall throughput.*

Required clock frequency: Based on equation (4.1) and (4.2), we can roughly identify the properties of the hardware architectures (i.e., the block width, the operating clock frequency as well as the different Γs) required to reach the target throughput of 100 Gbit/s. From Figure 4.B.3 we can see that the impact of the protocol/cipher overhead leads to a throughput that does not scale proportionally with increasing block widths for small frame payloads such as 46 bytes. For larger payloads like 1500 bytes (center image) or Jumbo frames of length 9000 bytes (right image), it is mainly the AEAD architecture that determines the overall throughput.

Figure 4.B.3 also indicates that for an architecture with $\Gamma_{Init} = \Gamma_A = \Gamma_M = \Gamma_{Fin} =$ 1 cycle and a block width of 64 bits, a clock

Table 4.C.1: AXI4-Stream Protocol signals description

Signal	Width	Description
`TValid`	1	Valid signal, indicating valid data on `TData`
`TReady`	1	Ready signal, indicating readiness for data
`TUser`	16	Data-describing control signal
`TData`	128–1024	Actual data signal

frequency of above 1 GHz would be required to pass the 100 Gbit/s goal. The reader should be aware that achieving such timings for a cryptographic algorithm on the 65 nm technology targeted is rather optimistic. Comparing the center and right image of Figure 4.B.3 with the left plot indicates that for larger frames, faster clock frequencies are needed to reach the threshold of 100 Gbit/s. This is because the protocol overhead no longer substantially impacts the actual cipher performance. It is important to keep in mind that the picture significantly changes for small payload sizes when hardware architectures are considered with higher Γ_{Init}, Γ_{Fin}, or $\Gamma_{A/M}$.

4.C AXI4-Stream Architecture Interface

As mentioned in Section 4.3.2, all AEAD architectures communicate with their environments via the AXI4-Stream Protocol. Data widths of the algorithms range from 128 bits to 1024 bits. The type of data provided to the candidate designs via the `TData` signal can be identified using the 16 control bits of the `TUser` signal. All input data, including cipherkeys, associated data, plaintexts/ciphertexts, public message numbers, and data lengths (where required) must be provided via the input AXI protocols from the source. The resulting ciphertexts/plaintexts and the corresponding authentication tag are made available via the output AXI interface to the destination. Table 4.C.1 contains a detailed description of the I/O signals of the AXI interface utilized. The only properties of the interface that change from algorithm to algorithm, are the width of the `TData` signal and the meaning of the `TUser` signal to the candidate architecture.

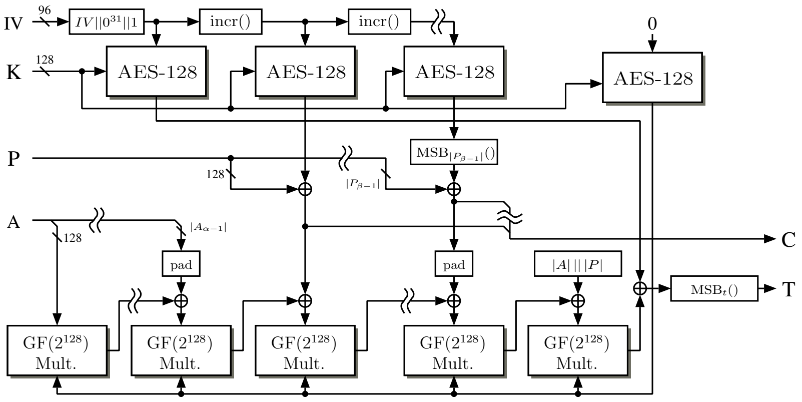


Figure 4.D.1: Simplified encryption process of GCM using AES-128 as the underlying block cipher and a 96 bit-wide IV.

4.D Our AEAD Architectures

In this section, we provide a short description of each of the investigated AEAD algorithms. Moreover, we present simplified block diagrams, depicting the structure of our 100 Gbit/s architectures.

4.D.1 GCM-AES Reference Architecture

GCM [45] uses a block cipher with an input width of 128 bits to provide confidentiality and assures integrity with the use of a universal hash function, called GHASH [45], based on a 128 bit Galois field multiplication. Figure 4.D.1 illustrates the encryption process of GCM using AES-128 as the underlying block cipher.[11] The inputs to the block cipher are counter values and thus, can be precomputed independently of the presence of the actual input data to be encrypted/authenticated. Therefore, the number of cipher and finite-field multiplier blocks, necessary to reach the target throughput, is mainly

[11]In the remainder of this section, GCM-AES always refers to GCM using the 128-bit version of AES if not stated otherwise.

limited by the input block width and the speed of the Galois field multiplier chain.

The Advanced Encryption Standard (AES)

AES is a well-established block cipher standardized by NIST [94] and other institutions. The algorithm operates on data blocks of 128 bit and supports three different cipherkey sizes, namely 128 bit, 192 bit, and 256 bit. Our investigations exclusively deal with the 128 bit version, hereafter referred to as AES-128 or simply AES. More details about the other cipher versions as well as an in-depth explanation of the algorithm can be obtained from [94].

Cipher Operations AES-128 comprises ten rounds, each operating on a 128-bit internal state S that can be represented as a 4×4 matrix of bytes. $S_{i,j}$ denotes the byte of row i and column j with $i, j \in \{0, \ldots, 3\}$. The state matrix gets initialized with the input data (one plaintext block). Each cipher round is made up of four different transformations, called *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*. The only difference between the ten rounds are the so-called *roundkeys* used for the AddRoundKey transformation (cf. Figure 4.D.2). Moreover, the MixColumns transformation is omitted in the final round. Prior to the ten rounds, an initial AddRoundKey transformation is executed. The four operations of the cipher round are defined as follows:

SubBytes: SubBytes performs a byte-wise substitution of the state using a substitution box (S-box). For the actual values of the S-box, we refer the reader to [94].

ShiftRows: Each byte of a row of the state is cyclically shifted to the left by the index of the row (zero-based). Therefore, the first row does not change, the bytes of the second row are rotated one byte to the left, and so on.

MixColumns: This operation can be understood as a column-by-column multiplication modulo $x^4 + 1$ in the finite field GF(2^8), looking at the columns of the state as polynomials of the Galois

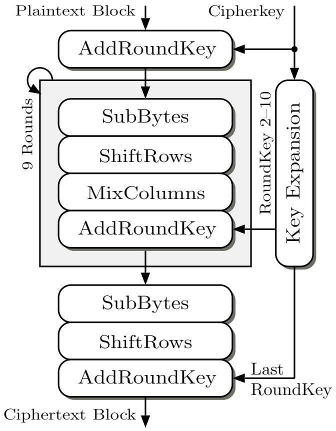


Figure 4.D.2: AES-128 encryption with the initial AddRoundKey, the 9 intermediate rounds, and the slightly different final round.

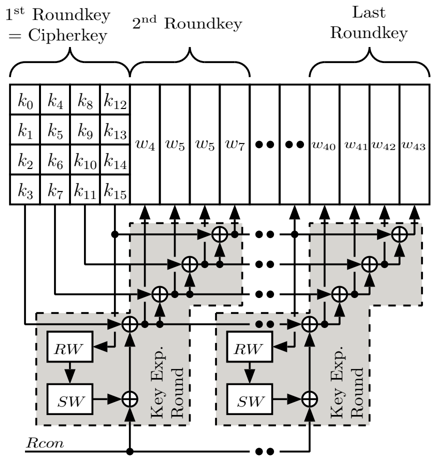


Figure 4.D.3: AES-128 key expansion; k_i denote the bytes of the key, w_j the words of the computed roundkeys, and *RW* and *SW* the *RotWord* and *SubWord* operations, respectively.

field. Additional information about MixColumns is available from the official standard [94].

AddRoundKey: The AddRoundKey function is a bitwise XOR operation of the state bits and the bits of the current roundkey.

Key Expansion Since AES-128 consists of ten cipher rounds and an initial AddRoundKey transformation, it requires 11 roundkeys. All 128-bit wide roundkeys are derived from the main cipherkey using a *Key Expansion*. Expanding the cipherkey to the roundkeys works in a column-by-column approach, where the first four columns represent the original cipherkey. Each of the first four columns contains four bytes of the cipherkey and the remaining columns are derived as shown in Figure 4.D.3. The *RotWord* and the *SubWord* operations of the key expansion are defined as follows:

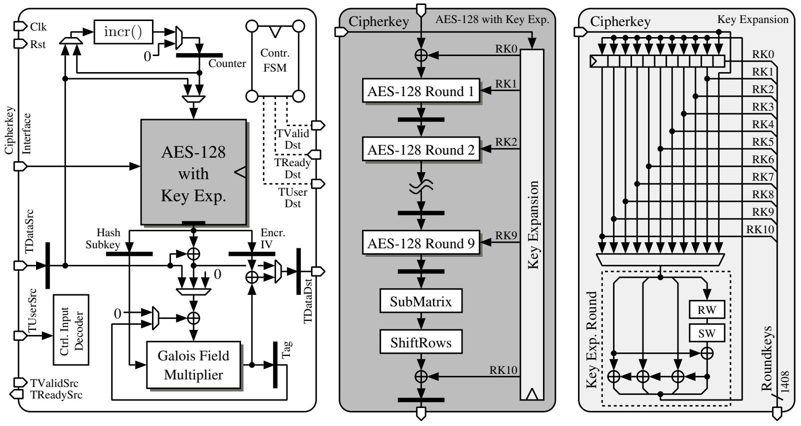


Figure 4.D.4: *Left:* GCM design based on one AES-128 core and one bit-parallel finite-field multiplier; *Middle:* Fully unrolled AES-128 architecture; *Right:* AES-128 key expansion; Bold bars indicate 128-bit registers; RW = RotWord operation, SW = SubWord operation;

RotWord: The RotWord function takes one column of the expanded key and performs a circular shift (rotation) of the bytes such that $[k_i, k_{i+1}, k_{i+2}, k_{i+3}]$ becomes $[k_{i+1}, k_{i+2}, k_{i+3}, k_i]$.

SubWord: All four bytes of a column get substituted using the same S-box as utilized throughout the cipher rounds.

Rcon denotes the round constants required for the key expansion. They differ from one round to another and can be obtained from [94].

Our 100 Gbit/s Data at Rest Architecture

It is well known from related work that high-throughput GCM-AES designs are usually based on multiple stages of AES and finite-field multiplier cores. For our reference architecture, it turned out that thanks to the 65 nm CMOS technology utilized, we were able to reach the asymptotic 100 Gbit/s goal with a single stage. The newly developed architecture is illustrated in Figure 4.D.4. It basically contains two main components. First, a fully unrolled AES-128 cipher core, pipelined after each round. The AES core also hosts the key expansion, including registers to store all the roundkeys. Due to our assumption that key exchanges take place infrequently, we iteratively reuse the logic for deriving one roundkey from another.[12] The second main component of the GCM design is a fully combinational, bit-parallel finite-field multiplier, required to compute the authentication tag. The controlling of the architecture is accomplished with Finite-State Machines (FSMs) located at the top level of the design. They control both the actual datapath and the communication with the design's environment using the AXI4-Stream interfaces. All other control signals, for instance, the *enable signals* for the cipher state and roundkey registers, are generated with the use of simple shift registers.

We investigated the proposed GCM-AES architecture using two different AES S-boxes to reduce the required area as far as possible. First, we created the S-box using a straightforward constant array in VHDL (further on denoted as *LUT version*). With this approach all the effort for implementing the logic of the fully-combinational S-box is shifted over to the synthesis tool. Second, we obtained results based on the Canright [31] S-box, which uses subfields to implement the required GF(2^8) inversion and is known to have the best area/timing trade-off from literature. While the composite-field S-box according to Canright requires less area compared to the LUT implementation, it is a little bit slower, which might increase the critical path of the AES-128 design and thus, that of the overall GCM architecture undesirable. As can be observed from the AT plot of AES in Figure 4.D.5, both the LUT-based version as well as the Canright S-box reach the asymptotic throughput of 100 Gbit/s. However, while the Canright design hardly achieves the targeted performance, the LUT-based version can be clocked even faster.

The AT plot in Figure 4.D.6 indicates that the bit-parallel Galois field multiplier is expected not to be part of the critical path of the resulting GCM architecture. Related work [103, 39, 83] often suggested to use different types of finite-field multipliers to reduce the required area stemming from the GHASH part of GCM, including Karatsuba-Ofman [64] multipliers. Such changes may improve the ef-

[12]For applications with very frequent key changes, a key expansion design with multiple *Key Exp. Round* entities should be considered.

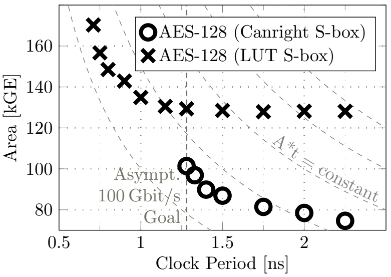


Figure 4.D.5: AT plot of the fully-unrolled AES-128 design for both the LUT and the Canright S-box based on synthesis results for the 65 nm target ASIC technology.

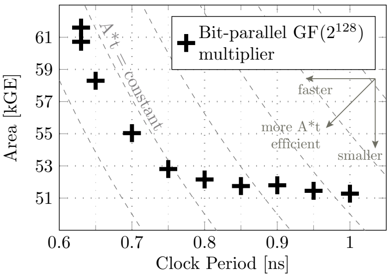


Figure 4.D.6: Synthesis results of the fully-combinational, bit-parallel finite-field multiplier in GF(2^{128}), required for the GHASH of GCM.

ficiency in terms of throughput-per-area a little bit. However, for our investigations we kept the bit-parallel multiplier as its critical path lies significantly below that of the AES core. This gives us more freedom with regard to the timing for the logic on the top level of the GCM architecture. Furthermore, by choosing a different GF(2^{128}) multiplier type, the latency may increase and hence, also the overall GCM-AES latency would increase unnecessarily. The additional *Tag* register in Figure 4.D.4 enables us to support processing of incoming header data even if the previous authentication tag has not yet been obtained by the destination from the *TDataDst* memory. If this feature is not required, the 128 bit *Tag* register can be removed.

4.D.2 AEGIS and MORUS

The basic structure of AEGIS [117, 118] and MORUS [116] is quite similar, which is why we discuss them together in the following. They mainly differ in the state update function utilized and in the number of rounds applied in the initialization and finalization phases.

Three different versions of AEGIS were proposed by their authors in the specification document, namely AEGIS-128L, AEGIS-128, and AEGIS-256. The former of these represents the primary recommenda-

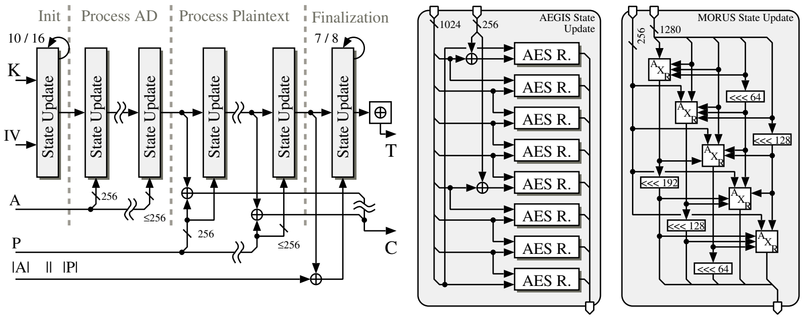


Figure 4.D.7: *Left:* General overview of the encryption process of both AEGIS-128L and MORUS-1280-128; *Center:* State update function of AEGIS-128L; *Right:* State update function of MORUS-1280-128

tion and is the only version of AEGIS considered for our investigations. The left image of Figure 4.D.7 gives an overview of the encryption process of both AEGIS and MORUS. It basically operates on a 1024-bit-wide state. The update function consists of eight AES rounds as depicted in the center image of Figure 4.D.7. Throughout each state update, AEGIS-128L processes 256 bit of associated or message data. Initialization and finalization require ten and seven iterations of the state update, respectively. The structure of AEGIS indicates that it aims at platforms, providing fast implementations of the AES round. Especially with hardware accelerations like the recent AES New Instruction Set (AES-NI) [50], high-performance software realizations of AEGIS are easily possible.

Compared to AEGIS, MORUS aims at achieving high performance on devices supporting AVX instructions as well as on custom hardware. Similarly to AEGIS, its major internal component is a state-update function, consisting of very simple logic operations (see right image of Figure 4.D.7), which gets applied several times on the state. The primary recommendation of MORUS—referred to as *MORUS-1280-128*—operates on a 1280-bit state, takes a 128-bit cipherkey, and processes 256 bit of AD or message data in each update step. In contrast to AD and message processing, where only a single call to

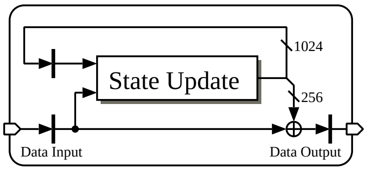


Figure 4.D.8: Simplified overview of the AEGIS-128L hardware architecture.

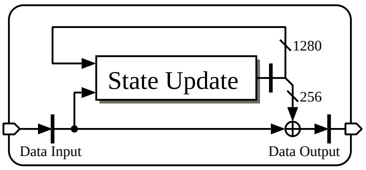


Figure 4.D.9: Simplified overview of the MORUS-1280-128 hardware architecture.

the update function is required, initialization and finalization need 16 and eight calls, respectively.

Our 100 Gbit/s Data at Rest Architectures

The architectures developed for both AEGIS and MORUS are based on a fully combinational update function and the controlling is accomplished using FSMs. In contrast to AES, which sequentially processes the incoming data through several rounds, AEGIS-128L *absorbs* 256 bit of data by processing them through eight AES rounds in parallel. Hence, in our AEGIS hardware architecture, illustrated in Figure 4.D.8, there is no need for pipelining within the state update function in order to achieve the asymptotic throughput of 100 Gbit/s. Moreover, the key expansion can be omitted since the cipherkey is already processed throughout the initialization phase (see left image of Figure 4.D.7). Thanks to the omission of the roundkey generation, the resulting area of the AEGIS architecture mainly boils down to the actual state update function and a couple of logic gates and registers around it. The AES round, used for AEGIS-128L, is equal to the one we presented in Section 4.D.1 for the GCM-AES reference architecture (i.e., a fully unrolled one based on the Canright S-box).

Similarly to the AEGIS-128L design, we developed a MORUS-1280-128 architecture with a single update function (see Figure 4.D.9). Due to the very hardware-friendly design of the update function of MORUS, the resulting design can be clocked at a high rate. We refer to Figure 4.E.3 in Appendix 4.E for an AT plot of the state update function only. A side effect of the simple state update of

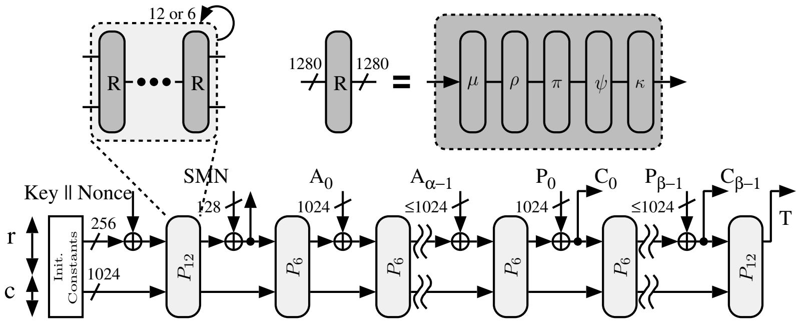


Figure 4.D.10: Simplified encryption process of ICEPOLE (padding is omitted due to simplicity reasons)

MORUS is that the surrounding glue logic (required to realize the overall algorithm functionality) as well as the registers need way more area compared to the update operation logic itself.

Note that input as well as output registers can be omitted for both the AEGIS-128L and MORUS-1280-128 architecture, in case I/O timings are considered uncritical. However, the critical path running through a complete state update function would then include the I/Os of the architecture.

4.D.3 ICEPOLE

Several of the CAESAR candidates are permutation-based algorithms, one of which is ICEPOLE [82]. We solely focus on the primarily recommended version of it, which is further on referred to as *ICEPOLE*. It follows the monkeyDuplex approach [27] with an internal state size of 1280 bit, which is organized as a $4 \times 5 \times 64$ bit array. ICEPOLE uses a 1026-bit-wide rate r, two bits of which just serve to pad each input block, and a 254-bit capacity c. The core element of the algorithm is the permutation P, an iterated version of the round function R. While for initialization and finalization 12 R rounds are used, denoted by P_{12}, only six rounds are needed during the other iterations. Figure 4.D.10 provides a simplified overview of the encryption process of ICEPOLE. The actual round function consists of five core opera-

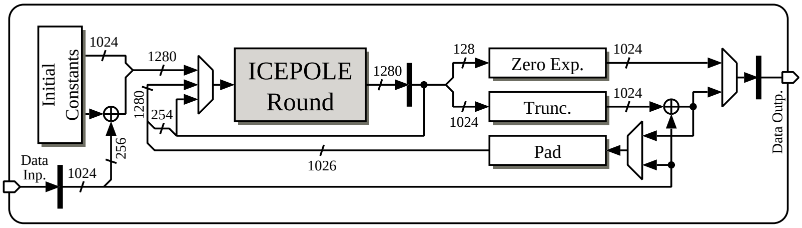


Figure 4.D.11: Simplified top-level architecture of the single-round-function-based ICEPOLE design (omitting controlling logic).

tions (μ, ρ, π, ψ, and κ). For a detailed description of these functions and the overall algorithm we refer the reader to [82].

Our 100 Gbit/s Data at Rest Architecture

We evaluated various hardware architectures for ICEPOLE [9], using different numbers of rounds implemented in hardware. Thanks to the very fast round operation, it turned out that a single instance in hardware is enough to reach the asymptotic throughput of 100 Gbit/s. Figure 4.D.11 shows the top-level design of the resulting, single-round based architecture. Note that due to our assumption about small I/O delays, we had to insert another output register to reduce the state-to-output timing. Although the output path would not have run through the ICEPOLE round otherwise, it would have reached almost the same length as the clock period. The reason for this is because the propagation delay of the *Trunc* entity is rather complex. That unit is responsible for truncating the 1024-bit-wide rate to the expected output length on the byte level. The actual ICEPOLE round is a completely combinational implementation and the whole design is controlled using FSMs.

4.D.4 NORX

NORX [10] is a permutation-based candidate, also following the monkeyDuplex approach [27]. Although a dedicated version for high-performance applications of the algorithm has been proposed, we solely

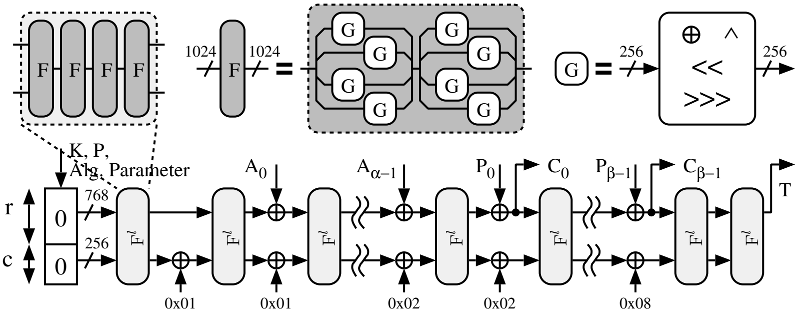


Figure 4.D.12: Simplified encryption process of NORX64-4-1.

considered NORX64-4-1 (further on referred to as *NORX*), as this is the primary recommendation of the authors. It operates on a $b = 1024$-bit-wide state, made up of a 4×4 array of 64-bit words. The 768-bit-wide rate r is used to absorb the incoming data by XOR-ing it. The remaining 256 bit of the state represent the capacity c, which is responsible for the actual security level of the algorithm. Figure 4.D.12 illustrates the encryption process of NORX, omitting the part for processing potential trailer data (i.e., AD after the plaintext/ciphertext) as this was not part of the CAESAR requirements. NORX64-4-1 performs four round transformations, denoted by F^l. The core component of the F function is the G permutation, which in turn consists of a couple of logic operations such as XOR, AND, and several shift and rotate operations. The hexadecimal numbers shown in Figure 4.D.12 represent the *domain separation constants*, which determine the type of the next incoming data. Detailed information about NORX can be obtained from its CAESAR submission document [10].

Our 100 Gbit/s Data at Rest Architecture

For the 100 Gbit/s NORX architecture, we make use of eight instances of the G permutation, thereby implementing one full round of the F function within one clock cycle. Figure 4.D.13 shows a simplified block diagram of the top-level design of the NORX architecture developed.

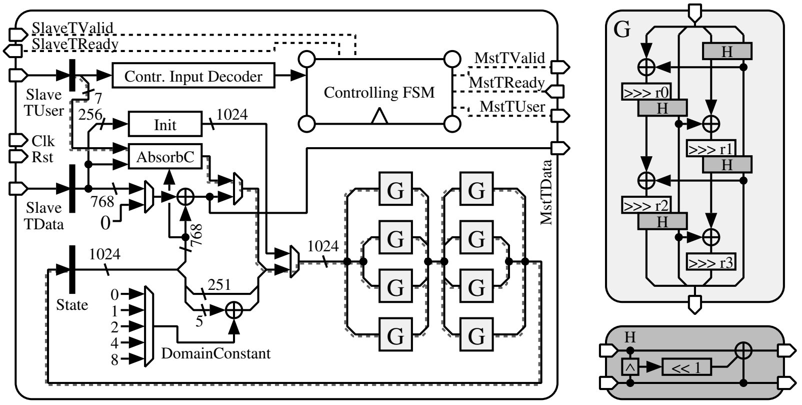


Figure 4.D.13: *Left:* Top-level design of the 100 Gbit/s NORX architecture based on eight G functions; *Right:* Structure of the G function;

The main components besides the logic for the G operations are the FSMs, responsible for controlling the datapath and the interfaces, state and I/O registers, some glue logic, and the *AbsorbC* entity. The latter is required to absorb the ciphertext during decryption mode into the state. The NORX design presented is a byte-oriented architecture, which is why 7 bits are required to determine how many of the 96 input bytes should actually be used for the new rate and how many will be kept from the previous state (i.e., the architecture supports full as well as non-full blocks down to the byte level). The *Init* module computes the initial state based on the 256-bit cipherkey and the 128-bit nonce. Note that, in general, the greater part adding up to the critical path stems from the two successive G functions (highlighted using the dashed red line in Figure 4.D.13). However, due to the high frequency required to achieve the 100 Gbit/s, also the glue logic (incl. the *AbsorbC* function) significantly contributes to the critical path.

4.D.5 Tiaoxin−346

Like many other CAESAR candidates, Tiaoxin−346 [91] is based on the AES round. Its core component is an update function, further

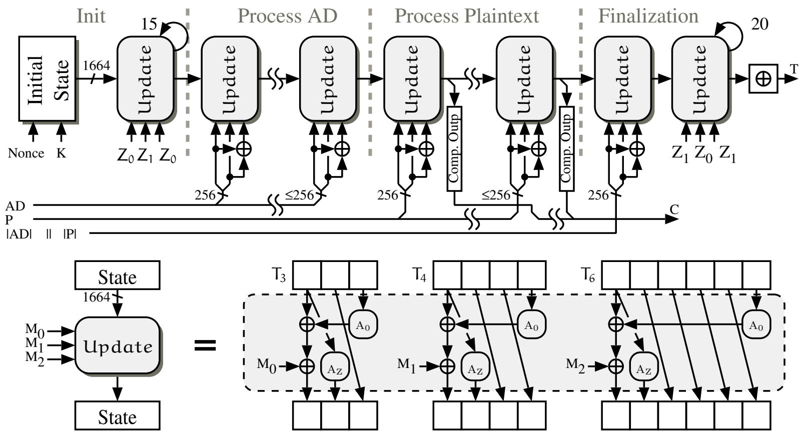


Figure 4.D.14: *Top:* Encryption process of Tiaoxin−346. *Bottom:* Internals of the state-update function; A_0 and A_Z denote a keyless and a keyed AES round using Z_0 as the roundkey.

on denoted by Update, which operates on a 1664-bit-wide state. An overview of the encryption process of Tiaoxin−346 is illustrated in the top image of Figure 4.D.14. Associated and message data are processed in blocks of 256 bit. Moreover, the required AES round calls are executed in parallel (see bottom image of Figure 4.D.14). While only a single call to the Update function is specified for each 256-bit block of incoming data, initialization and finalization need 15 and 20 calls, respectively. Just before the computation of the authentication tag, Tiaoxin−346 processes the lengths of the AD and message data in an additional Update call.

Internally, the state gets split into three different parts, referred to as T_3, T_4, and T_6, each of which consists of three, four, and six 128-bit words. Within Update, the AES round is, on the one hand, used with a zero roundkey (i.e., a keyless instance of the AES round) and, on the other hand, utilized with Z_0 as the roundkey. Z_0 and Z_1 are two constants from SHA-512 [95], being used throughout the algorithm.

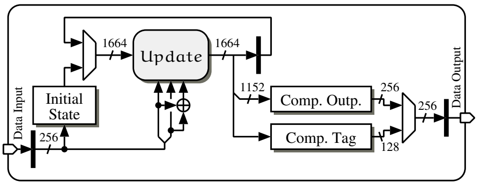


Figure 4.D.15: Simplified top-level datapath of the 100 Gbit/s data at rest Tiaoxin − 346 architecture (control logic and decryption paths are omitted due to readability).

Our 100 Gbit/s Data at Rest Architecture

Figure 4.D.15 shows a simplified version of the top-level architecture of the developed Tiaoxin − 346 design. In general, it consists of a fully-unrolled Update operation and a 1664-bit state register. The update function uses six parallel AES rounds based on Canright S-boxes to process the 256 bit of incoming data. Also the cipherkey and the required nonce are processed via the same data input. Both input as well as output registers are needed to minimize I/O timings. The AES round function employed is basically the same as the one for the GCM-AES reference architecture with the exception that the inputs for the roundkeys are tied to a constant value and thus, the overall AES round operation becomes simpler.

4.E Synthesis Results of CAESAR Candidates

Basically all of the CAESAR candidates investigated in this chapter share the property that their algorithmic structure is based on one or multiple core components. Examples thereof are permutations, state update functions, partial or even complete block ciphers like AES. These components usually get iteratively reused to create the complete AE primitive. As a result, once the technology-specific properties of them are known (i.e., maximum clock frequency and the corresponding area required), performance estimates for diverse ar-

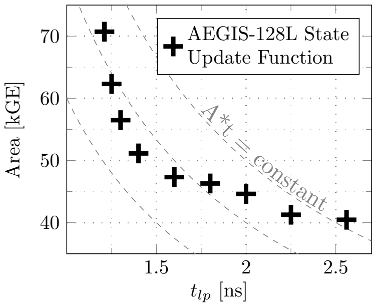


Figure 4.E.1: AEGIS-128L state update function

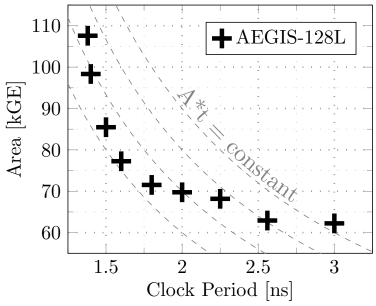


Figure 4.E.2: AEGIS-128L

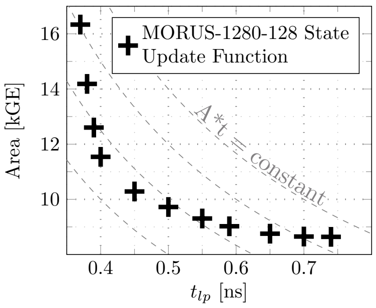


Figure 4.E.3: MORUS-1280-128 state update function

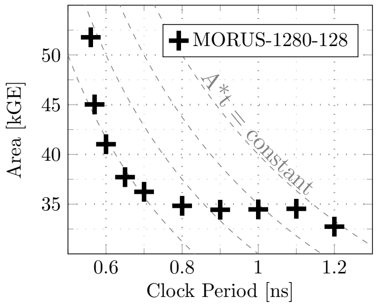


Figure 4.E.4: MORUS-1280-128

chitectures can be derived. Therefore, Figure 4.E.1 to Figure 4.E.10 provide AT plots of these components together with the AT plots of the resulting, completely verified CAESAR candidates. The numbers are based on synthesis results for the 65 nm ASIC technology targeted, using standard cells by UMC under typical case conditions.

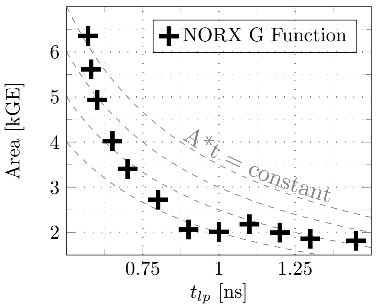


Figure 4.E.5: NORX G function

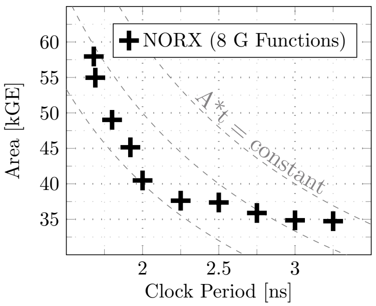


Figure 4.E.6: NORX

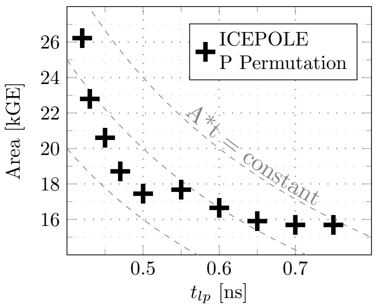


Figure 4.E.7: ICEPOLE P permutation

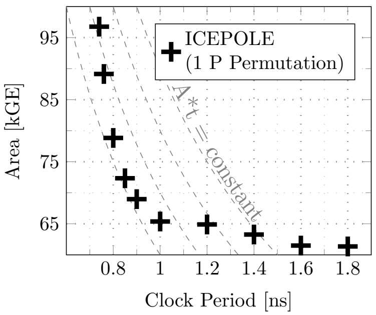


Figure 4.E.8: ICEPOLE

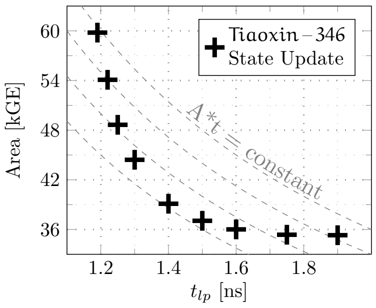


Figure 4.E.9: Tiaoxin − 346 Update operation

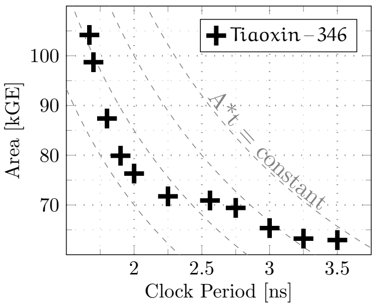


Figure 4.E.10: Tiaoxin − 346

5 Conclusions and Future Directions

To satisfy ambitious market requirements of symmetric encryption devices in terms of throughput and energy efficiency, enterprises commonly rely on hardware-based solutions. As the development and fabrication of these systems is a complex process, several weaknesses exist during the life cycle of a VLSI-based device. Attackers can exploit these vulnerabilities to weaken the device's security and the confidence of their users. We addressed some of these risks and performance aspects as part of this thesis and conclude our results by answering the research questions stated in Section 1.3 as follows:

How dangerous are hardware Trojans for VLSI-based devices and how practical are they?

Hardware Trojans for both ASICs and FPGAs are and will remain a hot research topic. Our contributions to this field are threefold:

Trojan insertion: We have demonstrated that it is *indeed possible to design a small and effective ASIC Trojan with little or no pre-*

vious knowledge about the inner functioning of the target circuit. Interfaces are especially vulnerable parts of an ASIC, since standardized protocols are usually implemented here, which allow a potential attacker to more easily implant a Trojan. With neither insider knowledge about the target design nor time-consuming reverse engineering techniques, attackers need to keep alterations as simple as possible. Our Denial-of-Service (DoS) Trojan was *inserted at the layout level, just prior to the actual fabrication stage using only physical layout modifications.* Such changes are rather risky for the adversary, as he may easily destroy the original design. Nevertheless, we managed to do it just within a couple of working days. We expect that more sophisticated Trojan variants are unlikely to be inserted during the fabrication step into the layout, because significantly more knowledge about the target design would be needed.

Furthermore, we used the open source *RapidSmith tools to insert a Trojan into a placed-and-routed FPGA configuration*, targeting a Xilinx Virtex-II device. This approach provides an easy way to implant malicious circuitry into an existing bitstream without reverse engineering any vendor-specific file formats.

Trojan detection: Compared to related work, we *actually fabricated both genuine as well as Trojan-infected ASIC samples* of a target design and conducted side-channel analyses exclusively from measurement data. Although the implanted DoS Trojan occupied a mere 0.5 % of the total area of the original design, we were able to detect it reliably using the power side channel. In contrast to related experimental work [70], we have confirmed that Principal Component Analysis (PCA), proposed by Agrawal et al. [6], works properly for Trojan detection in physical circuits. Information obtained from three dimensions or less allowed us to identify the three malicious circuits out of a bunch of eight Circuits Under Test (CUTs).

While power was used to detect the ASIC Trojans, *we employed Electromagnetic Radiation (EM) to identify malicious FPGA configurations.* All six different Trojans were detected on the target FPGA albeit only 0.7 % of the original design are occupied by Trojan logic.

Trojan localization: Although we were able to distinguish among genuine and malicious FPGA configurations, we *originally aimed at localizing the Trojan resources in the device.* Unfortunately, we fell short of pinpointing the exact location as EM signatures were not in line with the actual Trojan locations. However, we expect that with a more sensitive near-field EM probe, our results can be improved towards an actual Trojan localization. Interestingly, recently Balasch et al. [11] also used an EM side-channel fingerprinting technique to detect and localize hardware Trojans on an FPGA. They utilized the SASEBO-G board as well and inserted their hardware Trojans after placement and routing of the target design, similar to our approach presented herein. Thanks to a more sophisticated measurement setup, the authors of [11] were able to coarsely locate the Trojan circuits, which confirms our expectations expressed in [106].

In-house configuration of FPGAs lures a large part of customers into believing that it is much safer from a security point of view compared to the mostly unprotected ASIC fabrication process. This is partially true and confirmed by new approaches to ASIC manufacturing, such as *split-fabrication* [59, 60], where wafer processing is distributed across two foundries. First, the lower levels of the fabrication are outsourced to sophisticated but untrusted fabs. This includes all diffusion layers and, possibly, a subset of the metal layers of a CMOS technology. The upper metals are then added by a trusted foundry, completing the functionality of the design. Hence, *split-fabrication can be understood as an approach to achieve what is inherent to the FPGA development process.* Nevertheless, FPGAs suffer from other vulnerabilities due to their reconfigurability that need to be addressed.

Future directions. Our ASIC Trojan analysis comprises only a very small population of chips, which renders general statements difficult. An in-depth study with a larger volume would make the results significantly more meaningful. In addition to that, follow-up investigations must aim at closer collaboration with foundries. This ensures sound contributions regarding which deviations are due to process variations and which are actually due to Trojan influences. Moreover, applying an enhanced version of our EM-based localization approach to ASIC Trojans and comparing the results with those from FPGA analyses would be interesting.

Are state-of-the-art DPA countermeasures of embedded devices ready to withstand sophisticated attack scenarios, and what is the price we have to pay for them?

To answer this question, we presented Zorro, an ASIC designed and manufactured exclusively to serve as an assessment platform for DPA countermeasures. To be best of our knowledge, it is the first chip actually taped-out that hosts a Keccak-based Authenticated Encryption (AE) scheme (SpongeWrap) and allows to enable and disable different masking and hiding countermeasures at will. As DPA usually becomes a huge problem in pervasive applications such as the Internet of Things (IoT), we aimed at low resource consumption during the development of Zorro. Our chip hosts three independent architectures, which exclusively differ in the masking scheme utilized. The *smallest design on* Zorro *uses three shares for the masking and requires approximately 14 kGE* including Design for Testability (DFT) circuitries such as Built-In Self-Tests (BISTs) and scan Flip-Flops (FFs).

Using Correlation Power Analysis (CPA) and Pearson's correlation coefficient as a distinguisher, our experimental results mostly confirm what is known from theory regarding hiding countermeasures and the effort required to detect a significant correlation peak between correct and false key guesses. The only exception was that with hiding countermeasures enabled, the correct key hypothesis showed somewhat higher correlations than what is known from theory. Thus, less power traces were needed to mount a successful attack. Side-Channel Analysis (SCA) attacks based on measurements of the chip indicated that when operating in the unprotected mode of the designs, less than 100 power traces are sufficient to reveal the secret key using standard CPA. As a weak point we targeted the initial θ operation of the Keccak-f permutation. Our target of *withstanding DPA attacks with up to 100 000 power traces was neither achieved with hiding nor masking* countermeasures alone. Even the investigated masking scheme indicated significant correlation peaks for the correct key guess with just 70 000 acquired traces. Although the strength of the countermeasures heavily depend on the investigated algorithm and its implementation, our results indicate that with hiding or masking alone it is unlikely to thwart attacks from sophisticated adversaries.

Future directions. Our practical experiments with Zorro constitute initial findings. The ASIC offers several further possibilities to be analyzed as part of future work. An obvious next step would be a comparison between the Masked Mode (MM) and the Secure Masked Mode (SMM). Furthermore, based on an adequate approach to attack the non-linear χ operation of Keccak, the three different masking schemes on Zorro should be studied.

Can future AE algorithms keep up with today's standards in terms of their hardware efficiency from a VLSI perspective?

We provided the first extensive comparison of CAESAR candidates and a GCM-AES reference architecture for an ASIC technology. More specifically, we targeted the field of high-throughput ASIC designs and compared five second-round algorithms against a Galois/Counter Mode of Operation (GCM) design. A 65 nm CMOS technology by United Microelectronics Corporation (UMC) was used as the target technology and two different use cases were considered with regard to the available input data size:

Data at rest: As a first step, hardware architectures for the GCM-AES reference design and each of the CAESAR candidates investigated were developed, targeting an asymptotic throughput of 100 Gbit/s. While keeping the throughput goal in mind, we tried to minimize the required silicon area for the ASIC designs as far as possible. Area occupation as a primary metric for comparing the algorithms was considered here.

As a second indicator for the performance of the CAESAR candidates, the efficiency in terms of throughput-per-area was used. It turned out that *all AE algorithms investigated outperform GCM-AES by a factor of approximately 2 to 5.* When pushing the operating frequencies of the architectures to their absolute maximum and considering throughput-per-area as the comparison metric, MORUS is even about 15 times as efficient as GCM. Hence, we conclude that the authors of the CAESAR candidates have taken into consideration the asymptotic use case during the design of their algorithms, as all of them *require significantly less silicon area to achieve the throughput goal of 100 Gbit/s.* The simpler design of the ciphers allow straightforward architectures running at high clock frequencies. Moreover, due to the absence of complex building blocks (like the Galois field multiplier required for GCM), the silicon area required can be kept significantly lower compared to GCM.

Data in motion: Assuming very long input messages, as done for the asymptotic throughput goal of the data at rest scenario, is a rather strong assumption. Moreover, there are not many real-world applications for which this assumption actually holds. Hence, in a second step, we utilized Ethernet, TCP, and UDP communications to assess the performance of the architectures developed. We considered typical packet size distributions for each of the protocols and took into account the overhead stemming from protocol and cipher data to compute the throughput for the data in motion scenario. Since most of the candidates spend much more time on initialization and finalization than GCM-AES, their *performance drops significantly compared to the data at rest scenario.* However, as most of the CAESAR algorithm architectures developed can be operated at higher frequencies than required for the asymptotic 100 Gbit/s threshold, yet higher throughputs than with GCM can be achieved. MORUS even exceeds a value of 400 Gbit/s when only transmitting Jumbo frames, but needs to be clocked with a rather high clock frequency of 1.8 GHz.

Comparing cryptographic algorithms in terms of their hardware performance, as targeted for the decision of the second round of the CAESAR competition, is a much trickier task than comparing software implementations. We recommend to:

- First, specify the actual field of application (for instance, low-resource vs. high-performance) and security requirements (like side-channel resistance or expected security level) of the architectures to be developed.

- Second, additional specifications such as the frequency of key changes, the available input data size, or architectural-specific demands (I/O interface, timings, etc.) must be set.
- Eventually, an adequate metric for assessing the performance of the competitors has to be determined.

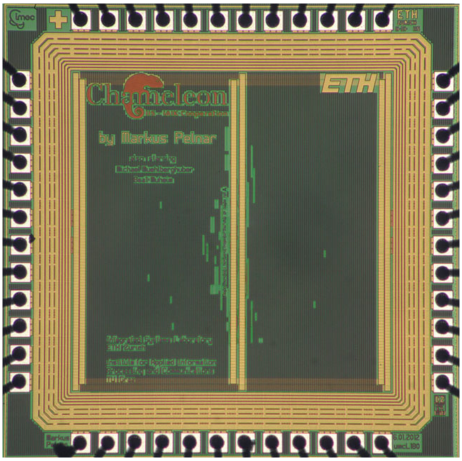
Just comparing the efficiency in terms of throughput-per-area can easily result in misleading analyses as discussed in this thesis. A more practical and sound approach would be to specify the actual protocols (incl. the target throughput) for which the algorithms should be investigated. As for the CAESAR competition, *we suggest to determine two different scenarios, one for low-resource and one for high-speed applications.* For the former, any protocol, typically used in resource-constrained environments (e.g., RFID or smart card systems), can be adopted. For the latter, a similar approach as presented in Chapter 4.4.3 might be appropriate (i.e., Ethernet for throughputs of 100 Gbit/s).

Future directions. The results presented herein comprise one specific scenario, namely the high-speed use case under the asymptotic throughput assumption. Based on this, several topics for future work come into mind. First, as mentioned above, an analysis of candidate architectures developed from scratch for a dedicated communication protocol would be interesting. Second, similar to the integration of the AES-NI instruction set into Intel CPUs, it is quite possible that a permutation will be adopted by processor design houses in the near future. Therefore, an evaluation taking into account just the permutation-based candidates should provide valuable contributions regarding which competitor should be favored.

A
Cryptographic ASICs

Throughout this thesis, a number of cryptographic Application-Specific Integrated Circuits (ASICs) have been developed. The following list summarizes the most relevant ones together with a brief description and their key properties. Additional information about the chips can be obtained from `http://asic.ethz.ch/cg/`.

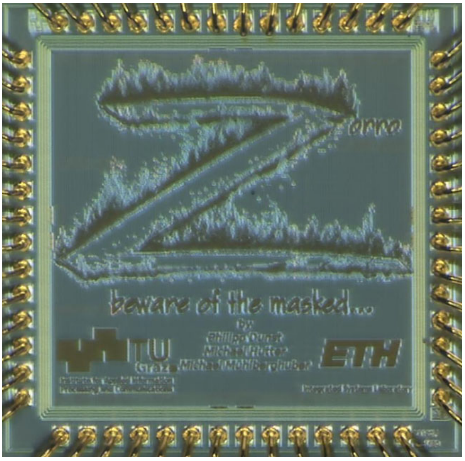
A.1 Chameleon/Chipit



Key Properties	
Tapeout Year	2012
Involved Students	Markus Pelnar, Philipp Dunst
Technology	UMC, 180 nm
Package	QFN56
Dimensions	1525 µm × 1525 µm
Core Area	40 kGE
Max. Frequency	125 MHz

Chameleon hosts three independent architectures. First, an AES implementation [94], targeting applications in the field of resource-constrained environments. Therefore, its main design goal was to keep the silicon area as small as possible. Second, it contains an iterative version of the cryptographic hash function Grøstl [46], which was one of the finalists of the SHA-3 hash competition [93] organized by NIST. And third, an architecture called *GrÆstl*, which combines both AES and Grøstl in a single optimized datapath. While Chipit contains the same circuitry from a functional point of view, we added a DoS hardware Trojan to it, which can be triggered externally.

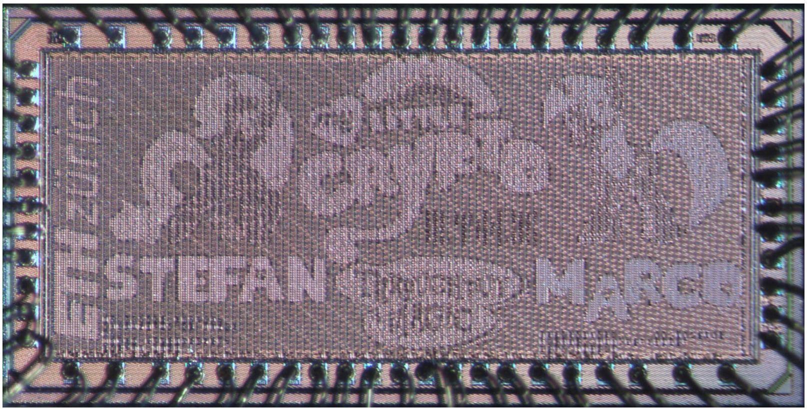
A.2 Zorro



Key Properties	
Tapeout Year	2014
Involved Students	Philipp Dunst
Technology	UMC, 180 nm
Package	QFN56
Dimensions	1525 µm × 1525 µm
Core Area	50 kGE
Max. Frequency	200 MHz

Zorro was designed to serve as an evaluation platform for Differential Power Analysis (DPA) countermeasures. Therefore, it contains three independent Authenticated Encryption (AE) primitives based on the Keccak-*f* permutation, each equipped with *hiding* and *masking* countermeasures. Since hardware architectures, vulnerable to Side-Channel Analysis (SCA) attacks, are usually found in pervasive environments such as Internet of Things (IoT) or Radio-Frequency Identification (RFID) applications, the main design goal was to keep the area requirements of the designs as small as possible. The three designs only differ with regard to the applied masking scheme and all countermeasures can be enabled/disabled at will.

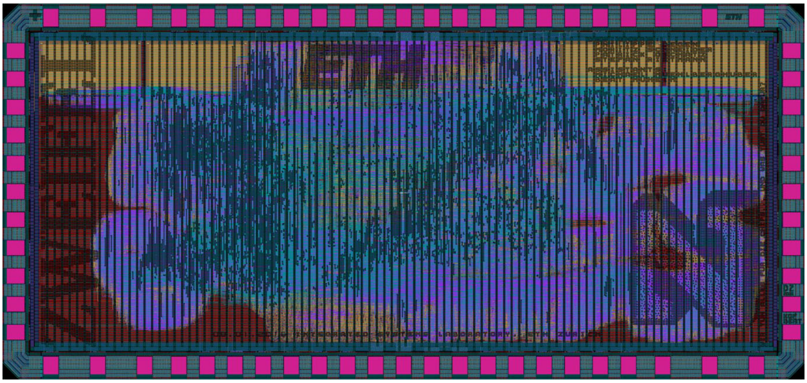
A.3 MLC:TiM



Key Properties	
Tapeout Year	2015
Involved Students	Marco Eppenberger, Stefan Mach, Cyril Arnould
Technology	UMC, 65 nm
Package	QFN56
Dimensions	2626 µm × 1252 µm
Core Area	1.1 MGE
Max. Frequency	405 MHz

In order to evaluate candidates of the CAESAR competition towards their suitability for high-throughput hardware implementations, MLC:TiM contains five competitors and a GCM-AES reference architecture. Their main design goal was to achieve a throughput of 100 Gbit/s on the smallest possible area. All architectures on the chip run independently from each other and can be fed with data from an on-chip RAM or an LFSR generating pseudo-random data sets. The common top-level design implements a serial-to-parallel interface at the input respectively a parallel-to-serial protocol at the output due to the pin constraints of the ASIC. Because of its rather complex original name *My Little Crypto: Throughput is Magic (MLC:TiM)*, the chip got renamed to *Pony* relatively quickly for its daily usage.

A.4 Zweifel



Key Properties	
Tapeout Year	2016
Involved Students	Philippe Degen, Moritz Schneider, Patrick Oschatz, Stefan Rietmann
Technology	UMC, 65 nm
Package	QFN56
Dimensions	2626 µm × 1252 µm
Core Area	1 MGE
Max. Frequency	710 MHz[†]

[†] As of this writing, the chips have not yet returned from fabrication. Hence, these values are the expected numbers obtained from post-layout simulations.

As a follow-up project to MLC:TiM, a second ASIC was initiated to add further CAESAR candidates to the high-throughput analysis. It was called *Zweifel* and hosts the Authenticated Encryption (AE) schemes *Ascon*, *Joltik*, *Minalpher*, and *SCREAM*. Similar to MLC:TiM, the main design goal for the architectures was to achieve a throughput of 100 Gbit/s, targeting the above mentioned 65 nm CMOS technology by UMC. Again a common top-level design was used to feed to competitors with input data, this time with a slightly more sophisticated Built-In Self-Test (BIST) to functionally verify the cryptographic primitives directly on-chip.

Acronyms

AD	Associated Data
AE	Authenticated Encryption
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AES-NI	AES New Instruction Set
AMBA	Advanced Microcontroller Bus Architecture
AN	Association Number
APB	Advanced Peripheral Bus
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
ATPG	Automated Test Pattern Generation
AXI	Advanced eXtensible Interface Bus
BC	Block Cipher
BIL	Bitfile Interpretation Library
BIST	Built-In Self-Test
BRAM	Block RAM
CAESAR	Competition for Authenticated Encryption: Security, Applicability, and Robustness
CBC	Cipher Block Chaining
CD	Cipher Data
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide-Semiconductor
CO	Cipher Overhead
COTS	Commercial Off-The-Shelf
CPA	Correlation Power Analysis
CRC	Cyclic Redundancy Check

CUT	Circuit Under Test
DCM	Digital Clock Manager
DFT	Design for Testability
DoM	Difference of Means
DoS	Denial-of-Service
DPA	Differential Power Analysis
DRP	Dual-Rail Precharge
DSP	Digital Signal Processing
DUT	Device Under Test
EAL	Evaluation Assurance Level
EDA	Electronic Design Automation
EM	Electromagnetic Radiation
FC	Fibre Channel
FCS	Frame Check Sequence
FF	Flip-Flop
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
GCM	Galois/Counter Mode of Operation
GDSII	Graphic Database System II
GE	Gate Equivalent
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HM	Hiding Mode
IBM	International Business Machines Corporation
IC	Integrated Circuit
ICT	Information and Communications Technology
ICV	Integrity Check Value
IEEE	Institute of Electrical and Electronics Engineers
IIS	Integrated Systems Laboratory
IISSI	Integrated Systems Laboratory (IIS) Silicium Integritas
IoT	Internet of Things

IP	Intellectual Property
IP	Internet Protocol
IPG	Interpacket Gap
IV	Initialization Vector
JTAG	Joint Test Action Group
KL	Karhunen-Loève
LAN	Local Area Network
LED	Light-Emitting Diode
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Lookup Table
MAC	Message Authentication Code
MAC	Media Access Control
MM	Masked Mode
NCD	Netlist Circuit Description
NIST	National Institute of Standards and Technology
NM	Normal Mode
OSI	Open Systems Interconnect
PC	Personal Computer
PCA	Principal Component Analysis
PCB	Printed Circuit Board
PD	Protocol Data
PMN	Public Message Number
PN	Packet Number
PO	Protocol Overhead
PRNG	Pseudorandom Number Generator
RAM	Random-Access Memory
RFID	Radio-Frequency Identification
RTL	Register-Transfer Level

SABL	Sense Amplifier Based Logic
SCA	Side-Channel Analysis
SCI	Secure Channel Identifier
SDH	Synchronous Digital Hierarchy
SFD	Start Frame Delimiter
SMM	Secure Masked Mode
SMN	Secure Message Number
SNR	Signal-to-Noise Ratio
SONET	Synchronous Optical Networking
SPA	Simple Power Analysis
SVM	Support Vector Machine
TCI	TAG Control Information
TCP	Transmission Control Protocol
TI	Threshold Implementation
TPM	Trusted Platform Module
TSMC	Taiwan Semiconductor Manufacturing Company
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UMC	United Microelectronics Corporation
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLAN	Virtual Local Area Network
VLSI	Very Large Scale Integration
WDDL	Wave Dynamic Differential Logic
XDL	Xilinx Design Language

Symbols

b	State size of the KECCAK-f permutation
c	Capacity of the KECCAK-f permutation
l	Length of Associated Data (AD) in bits
m	Length of the message in bits, i.e., $m = \|P\| = \|C\|$
r	Rate of the KECCAK-f permutation
t	Length of the authentication tag in bits (usually 128 bits)
ti	Number of time instances due to randomizing countermeasures
A	Associated Data (AD)
A_i	i-th block of AD; $i \in \{0 \ldots \alpha - 1\}$
A^*	Padded AD to reach a multiple of the desired block length
C	Ciphertext data to be checked for integrity and to be decrypted
C_j	j-th block of ciphertext; $j \in \{0 \ldots \beta - 1\}$
C^*	Padded ciphertext to reach a multiple of the desired block length
IV	Initialization vector
K	Cipherkey
M	Message data (may refer to plaintext or ciphertext)
P	Plaintext data to be authenticated and encrypted
P_j	j-th block of plaintext; $j \in \{0 \ldots \beta - 1\}$
P^*	Padded plaintext to reach a multiple of the desired block length
PMN	Public message number
$\boldsymbol{S}_{init}$	Initial value of the KECCAK-f state

S_z	Slice number z of the Keccak-f state
T	Authentication tag
α	Number of AD blocks to be authenticated
β	Number of plaintext respectively ciphertext blocks
τ	Number of authentication tag blocks
$\Gamma_\cdot$	Number of clock cycles required for a hardware architecture to process data item $\cdot$
ρ_c	Correlation coefficient of the correct key guess
Θ	The throughput of a hardware architecture

Operators

0^x	Bit-vector with x zeros
1^x	Bit-vector with x ones
$\cdot\|\|\cdot$	Concatenation
$\oplus$	Bitwise Exclusive-or (XOR)
$\overline{\cdot}$	Bitwise negation (NOT)
$\vee$	Bitwise OR
$\wedge$	Bitwise AND
$\|\cdot\|$	Length of $\cdot$ in bits

Bibliography

[1] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. [Online]. Available: http://competitions.cr.yp.to/caesar.html

[2] Implementation notes: amd64, titan0, crypto_aead. [Online]. Available: http://bench.cr.yp.to/web-impl/amd64-titan0-crypto_aead.html

[3] J. Aarestad, D. Acharyya, R. Rad, and J. Plusquellic, "Detecting Trojans Through Leakage Current Analysis Using Multiple Supply Pad I_{DDQ}s," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 4, pp. 893–904, Dec 2010.

[4] F. Abed, S. Fluhrer, J. Foley, C. Forler, E. List, S. Lucks, D. McGrew, and J. Wenzel. (2015, Aug.) The POET Family of On-Line Authenticated Encryption Schemes. [Online]. Available: http://competitions.cr.yp.to/round2/poetv20.pdf

[5] V. K. Adhikari, S. Jain, and Z.-L. Zhang, "YouTube Traffic Dynamics and Its Interplay with a Tier-1 ISP: An ISP Perspective," in *Proc. of IMC'10*, 2010, pp. 431–443.

[6] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan Detection using IC Fingerprinting," in *Proc. of Security and Privacy*, 2007, pp. 296–310.

[7] E. Andreeva, B. Bilgin, A. Bogdanov, A. Luykx, F. Mendel, B. Mennink, N. Mouha, Q. Wang, and K. Yasuda. (2014, Sep.) PRIMATEs v1.02. [Online]. Available: http://competitions.cr.yp.to/round2/primatesv102.pdf

[8] ARM, “AMBA 4 AXI4-Stream Protocol,” ARM, 2010, version: 1.0. [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html

[9] C. Arnould, “Towards Developing ASIC and FPGA Architectures of High-Throughput CAESAR Candidates,” Master’s thesis, ETH Zurich, Autumn Term 2014.

[10] J.-P. Aumasson, P. Jovanovic, and S. Neves, “NORX V1,” http://competitions.cr.yp.to/round1/norxv1.pdf, Mar. 2014.

[11] J. Balasch, B. Gierlichs, and I. Verbauwhede, “Electromagnetic Circuit Fingerprints for Hardware Trojan Detection,” in *Proc. of EMC’15*, Aug. 2015, pp. 246–251.

[12] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leierson, M. Marson, P. Rohatgi, and S. Saab, “Test Vector Leakage Assessment (TVLA) methodology in practice,” 2013.

[13] G. T. Becker, A. Lakshminarasimhan, L. Lin, S. Srivathsa, V. B. Suresh, and W. Burelson, “Implementing Hardware Trojans: Experiences from a Hardware Trojan Challenge,” in *Proc. of ICCD’11*, 2011, pp. 301–304.

[14] M. Bellare and C. Namprempre, “Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm,” in *Advances in Cryptology – ASIACRYPT 2000*, 2000, vol. 1976, pp. 531–545.

[15] T. Benson, A. Akella, and D. A. Maltz, “Network Traffic Characteristics of Data Centers in the Wild,” in *Proc of. IMC’10*, 2010, pp. 267–280.

[16] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding Data Center Traffic Characteristics,” *SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, Jan. 2010.

[17] F. Benz, A. Seffrin, and S. A. Huss, “BIL: A Tool-Chain for Bitstream Reverse-Engineering,” in *Proc. of FPL’12*, 2012, pp. 735–738.

[18] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, and R. V. Keer. (2012, May) KECCAK Implementation Overview. Version 3.2. [Online]. Available: http://keccak.noekeon.org/Keccak-implementation-3.2.pdf

[19] G. Bertoni, J. Daemen, N. Debande, T.-H. Le, M. Peeters, and G. Van Assche, "Power Analysis of Hardware Implementations Protected with Secret Sharing," Cryptology ePrint Archive, Report 2013/067, 2013.

[20] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The KECCAK sponge function family. [Online]. Available: http://keccak.noekeon.org

[21] ——, "Sponge Functions," in *ECRYPT Hash Workshop*, 2007. [Online]. Available: http://sponge.noekeon.org/SpongeFunctions.pdf

[22] ——, "Building power analysis resistant implementations of Keccak," 2nd SHA-3 Candidate Conference, Aug. 2010. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/BERTONI_KeccakAntiDPA.pdf

[23] ——, "Duplexing the sponge: single-pass authenticated encryption and other applications," 2nd SHA-3 Candidate Conference, Aug. 2010. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/presentations/DAEMEN_SpongeDuplexSantaBarbaraSlides.pdf

[24] ——. (2011, Jan.) Cryptographic sponge functions. [Online]. Available: http://sponge.noekeon.org/CSF-0.1.pdf

[25] ——. (2011, Jan.) The KECCAK reference. [Online]. Available: http://keccak.noekeon.org/Keccak-reference-3.0.pdf

[26] ——, "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications," in *Proc. of SAC'12*, 2012, vol. 7118, pp. 320–337.

[27] ——, “Permutation-based encryption, authentication and authenticated encryption,” Directions in Authenticated Ciphers (DIAC’12), 2012. [Online]. Available: http://keccak.noekeon.org/KeccakDIAC2012.pdf

[28] B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche, “Efficient and First-Order DPA Resistant Implementations of Keccak,” in *Proc. of CARDIS’14*, 2014, pp. 187–199.

[29] N. Borisov, I. Goldberg, and D. Wagner, “Intercepting Mobile Communications: The Insecurity of 802.11,” in *Proc. of MobiCom’01*, 2001, pp. 180–189.

[30] E. Brier, C. Clavier, and F. Olivier, “Correlation power analysis with leakage model,” in *Proc. of CHES’04*, 2004, pp. 16–29.

[31] D. Canright, “A Very Compact S-Box for AES,” in *Proc. of CHES’05*, 2005, vol. 3659, pp. 441–455.

[32] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux, “Password Interception in a SSL/TLS Channel,” in *Proc. of CRYPTO’03*, 2003, vol. 2729, pp. 583–599.

[33] R. B. Cattell, “The Scree Test For The Number Of Factors,” *Multivariate Behavioral Research*, vol. 1, no. 2, pp. 245–276, 1966.

[34] A. Chakraborti, A. Chattopadhyay, M. Hassan, and M. Nandi, “TriviA: A Fast and Secure Authenticated Encryption Scheme,” in *Proc. of CHES’15*, 2015, vol. 9293, pp. 330–353.

[35] R. Chakraborty, S. Narasimhan, and S. Bhunia, “Hardware trojan: Threats and emerging solutions,” in *Proc. of HLDVT’09*, Nov. 2009, pp. 166–171.

[36] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, “Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?” in *Proc. of MICRO’10*, 2010, pp. 225–236.

[37] C. Clavier, J.-S. Coron, and N. Dabbous, “Differential Power Analysis in the Presence of Hardware Countermeasures,” in *Proc. of CHES’00*, 2000, pp. 252–263.

[38] F. Courbon, P. Loubet-Moundi, J. J.-A. Fournier, and A. Tria, “SEMBA: a SEM Based Acquisition technique for fast invasive Hardware Trojan detection,” in *Proc. of ECCTD’15*, Aug. 2015.

[39] J. Crenne, P. Cotret, G. Gogniat, R. Tessier, and J.-P. Diguet, “Efficient key-dependent message authentication in reconfigurable hardware,” in *Proc. of FPT’11*, Dec. 2011, pp. 1–6.

[40] Defense Advanced Research Projects Agency (DARPA). (2007) Trusted Integrated Circuits (TRUST).

[41] Defense Science Board. (2005, Feb.) Defense Science Board Task Force On High Performance Microchip Supply. [Online]. Available: http://www.acq.osd.mil/dsb/reports/ADA435563.pdf

[42] J. P. Degabriele and K. G. Paterson, “On the (In)Security of IPsec in MAC-then-Encrypt Configurations,” in *Proc. of CCS’10*, 2010, pp. 493–504.

[43] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. (2015, Aug.) Ascon v1.1. [Online]. Available: http://competitions.cr.yp.to/round2/asconv11.pdf

[44] M. Dworkin, “Recommendations for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality,” NIST, Tech. Rep., 2004.

[45] ——, “Recommendations for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC,” NIST, Tech. Rep., 2007.

[46] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. (2011, Mar.) Grøstl – a SHA-3 candidate. Submission to NIST (Round 3). [Online]. Available: http://www.groestl.info/Groestl.pdf

[47] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST Non-invasive attack testing workshop*, 2011. [Online]. Available: http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf

[48] H. Groß, E. Wenger, C. Dobraunig, and C. Ehrenhöfer, "Suit up! Made-to-Measure Hardware Implementations of Ascon," Cryptology ePrint Archive, Report 2015/034, 2015.

[49] V. Grosso, G. L. F.-X. Standaert, K. Varici, F. Durvaux, L. Gaspar, and S. Kerckhof, "SCREAM & iSCREAM," http://competitions.cr.yp.to/round1/screamv1.pdf, Mar. 2014.

[50] S. Gueron, "Intel® Advanced Encryption Standard (AES) New Instructions Set," Intel Corporation, Tech. Rep., 2010.

[51] L. Henzen and W. Fichtner, "FPGA Parallel-Pipelined AES-GCM Core for 100G Ethernet Applications," in *Proc. of ESSCIRC'10*, Sep. 2010, pp. 202–205.

[52] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: a first study," *Journal of Cryptographic Engineering*, vol. 1, no. 4, pp. 293–302, 2011.

[53] IEEE, "IEEE Standard for Local and metropolitan area networks - Media Access Control (MAC) Security," *IEEE Std 802.1AE-2006*, Aug. 2006.

[54] ——, "IEEE Standard for Ethernet," *IEEE Std 802.3-2012 (Revision of IEEE Std 802.3-2008)*, Dec. 2012.

[55] ——, "IEEE Standard for Ethernet - SECTION ONE," *IEEE Std 802.3-2012 (Revision of IEEE Std 802.3-2008)*, Dec. 2012.

[56] ——, "IEEE Standard for Ethernet - SECTION SIX," *IEEE Std 802.3-2012 (Revision of IEEE Std 802.3-2008)*, Dec. 2012.

[57] IEEE, “IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks,” *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, Dec. 2014.

[58] IEEE Std 802.11-2007, “IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” LAN/MAN Standards Committee, New York, NY, USA, pp. C1–1184, Jun. 2007.

[59] Intelligence Advanced Research Projects Activity (IARPA). (2011) Trusted Integrated Circuit Program. [Online]. Available: http://www.iarpa.gov/index.php/research-programs/tic

[60] R. Jarvis and M. McIntyre, “Split manufacturing method for advanced semiconductor circuits,” Patent 7,195,931, 2007, US Patent.

[61] Y. Jin and Y. Makris, “Hardware Trojan Detection Using Path Delay Fingerprint,” in *Proc. of HOST’08*, 2008, pp. 51–57.

[62] I. Jolliffe, *Principal Component Analysis.* Springer, 2002.

[63] H. Kaeslin, *Top-Down Digital VLSI Design Vol. 2 (Lecture Notes) - From Gate-Level Circuits to CMOS Fabrication*, Sep. 2015.

[64] A. Karatsuba and Y. Ofman, “Multiplication of Multidigit Numbers on Automata,” in *Soviet Physics Doklady*, vol. 7, 1963, p. 595.

[65] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, “Trustworthy Hardware: Identifying and Classifying Hardware Trojans,” *IEEE Computer*, vol. 43, no. 10, pp. 39–46, Oct. 2010.

[66] E. B. Kavun and T. Yalcin, “A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications,” in *Proc. of RFIDSec’10*, 2010, pp. 258–269.

[67] G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser, "Comprehensive Formal Verification of an OS Microkernel," *Transactions on Computer Systems*, vol. 32, no. 1, pp. 2:1–2:70, Feb. 2014.

[68] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Proc. of CRYPTO' 99*, 1999, pp. 388–397.

[69] H. Krawczyk, "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)," in *Proc. of CRYPTO'01*, 2001, vol. 2139, pp. 310–331.

[70] S. Kutzner, A. Y. Poschmann, and M. Stöttinger, "Hardware Trojan Design and Detection: A Practical Evaluation," in *Proc. of WESS'13*, 2013, pp. 1:1–1:9.

[71] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs," in *Proc. of FPL'11*, 2011, pp. 349–355.

[72] S. Lemsitzer, J. Wolkerstorfer, N. Felber, and M. Braendli, "Multi-gigabit GCM-AES Architecture Optimized for FPGAs," in *Proc. of CHES'07*, 2007, vol. 4727, pp. 227–238.

[73] J. Li and J. Lach, "At-Speed Delay Characterization for IC Authentication and Trojan Horse Detection," in *Proc. of HOST'08*, 2008, pp. 8–14.

[74] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks – Revealing the Secrets of Smart Cards.* Springer, 2007.

[75] S. Mangard, T. Popp, and B. Gammel, "Side-Channel Leakage of Masked CMOS Gates," in *Proc. of CT-RSA'05*, 2005, vol. 3376, pp. 351–365.

[76] D. A. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," *Submission to NIST Modes of Operation Process*, May 2005.

[77] G. E. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, Apr. 1965.

[78] A. Moradi. A Hardware Implementation of POET 2. [Online]. Available: https://www.uni-weimar.de/fileadmin/user/fak/medien/professuren/Mediensicherheit/Research/Publications/poet_v2-hardware.pdf

[79] A. Moradi, A. Barenghi, T. Kasper, and C. Paar, "On the Vulnerability of FPGA Bitstream Encryption Against Power Analysis Attacks: Extracting Keys from Xilinx Virtex-II FPGAs," in *Proc. of CCS'11*, 2011, pp. 111–124.

[80] A. Moradi, D. Oswald, C. Paar, and P. Swierczynski, "Side-channel Attacks on the Bitstream Encryption Mechanism of Altera Stratix II: Facilitating Black-Box Analysis Using Software Reverse-engineering," in *Proc. of FPGA'13*, 2013, pp. 91–100.

[81] A. Moradi and T. Schneider, "Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series," Cryptology ePrint Archive, Report 2016/249, 2016.

[82] P. Morawiecki, K. Gaj, E. Homsirikamol, K. Matusiewicz, J. Pieprzyk, M. Rogawski, M. Srebrny, and M. Wójcik, "ICE-POLE v2," http://competitions.cr.yp.to/round2/icepolev2.pdf, Aug. 2015.

[83] M. Mozaffari-Kermani and A. Reyhani-Masoleh, "Efficient and High-Performance Parallel Hardware Architectures for the AES-GCM," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1165–1178, Aug. 2012.

[84] M. Muehlberghuber. NORX Hardware Reference Implementation. [Online]. Available: https://github.com/norx/norx-hw

[85] M. Muehlberghuber, F. K. Gürkaynak, T. Korak, P. Dunst, and M. Hutter, "Red Team vs. Blue Team Hardware Trojan Analysis: Detection of a Hardware Trojan on an Actual ASIC," in *Proc. of HASP'13*, 2013, pp. 1–8.

[86] M. Muehlberghuber, C. Keller, N. Felber, and C. Pendl, "100 Gbit/s Authenticated Encryption Based on Quantum Key Distribution," in *Proc. of VLSI-Soc'12*, Oct. 2012, pp. 123–128.

[87] M. Muehlberghuber, C. Keller, F. K. Gürkaynak, and N. Felber, “FPGA-Based High-Speed Authenticated Encryption System,” in *From Algorithms to Circuits and System-on-Chip Design*, 2013, vol. 418, pp. 1–20.

[88] M. Muehlberghuber, T. Korak, P. Dunst, and M. Hutter, “Towards Evaluating DPA Countermeasures for KECCAK on a Real ASIC,” in *Proc. of COSADE’15*, 2015, vol. 9064, pp. 222–236.

[89] S. Narasimhan, D. Du, R. S. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy, and S. Bhunia, “Multiple-Parameter Side-Channel Analysis: A Non-Invasive Hardware Trojan Detection Approach,” in *Proc. of HOST’10*, Jun. 2010, pp. 13–18.

[90] I. Nikolić. CAESAR candidates speed comparison. [Online]. Available: http://www1.spms.ntu.edu.sg/~syllab/speed/

[91] ——, “Tiaoxin – 346, VERSION 2.0,” http://competitions.cr.yp.to/round2/tiaoxinv2.pdf, Aug. 2015.

[92] S. Nikova, C. Rechberger, and V. Rijmen, “Threshold Implementations Against Side-Channel Attacks and Glitches,” in *Proc. of ICICS’06*, 2006, vol. 4307, pp. 529–545.

[93] NIST. SHA-3 Cryptographic Hash Algorithm Competition. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/

[94] ——, *Advanced Encryption Standard (AES) (FIPS PUB 197)*, National Institute of Standards and Technology, Nov. 2001.

[95] ——, *Secure Hash Standard (SHS) (FIPS PUB 180-4)*, National Institute of Standards and Technology, Mar. 2012.

[96] K. Paterson and A. Yau, “Cryptography in Theory and Practice: The Case of Encryption in IPsec,” in *Proc. of EUROCRYPT’06*, 2006, vol. 4004, pp. 12–29.

[97] M. Pelnar, M. Muehlberghuber, and M. Hutter, “Putting together What Fits together - GrÆStl,” in *Proc. of CARDIS’12*, 2013, vol. 7771, pp. 173–187.

[98] K. Pentikousis and H. Badr, "Quantifying the deployment of TCP options - a comparative study," *IEEE Communications Letters*, vol. 8, no. 10, pp. 647–649, Oct. 2004.

[99] P. Pessl and M. Hutter, "Pushing the Limits of SHA-3 Hardware Implementations to Fit on RFID," in *Proc. of CHES'13*, 2013, vol. 8086, pp. 126–141.

[100] E. Prouff, M. Rivain, and R. Bevan, "Statistical Analysis of Second Order Differential Power Analysis," *IEEE Transactions on Computers*, vol. 58, no. 6, pp. 799–811, Jun. 2009.

[101] Research Center for Information Security. (2008, Oct.) Side-channel Attack Standard Evaluation Board - SASEBO-G. National Institute of Advanced Industrial Science and Technology. [Online]. Available: http://www.risec.aist.go.jp/project/sasebo/download/SASEBO-G_Spec_Ver1.0_English.pdf

[102] Y. Sasaki, Y. Todo, K. Aoki, Y. Naito, T. Sugawara, Y. Murakami, M. Matsui, and S. Hirose, "Minalpher v1.1," http://competitions.cr.yp.to/round2/minalpherv11.pdf, Aug. 2015.

[103] A. Satoh, T. Sugawara, and T. Aoki, "High-Performance Hardware Architectures for Galois Counter Mode," *IEEE Transactions on Computers*, vol. 58, no. 7, pp. 917–930, Jul. 2009.

[104] R. Schilling, M. Jelinek, M. Ortoff, and T. Unterluggauer, "A low-area ASIC implementation of AEGIS128 – A fast authenticated encryption algorithm," in *Proc. of Austrochip'14*, Oct. 2014, pp. 1–5.

[105] D. Šijačić, B. Yang, and B. Bilgin. (2015, May) Minalpher & PRIMATEs: Overview of Lightweight Hardware Implementation Results. *Google Group:* Cryptographic competitions. [Online]. Available: https://groups.google.com/forum/#!forum/crypto-competitions

[106] O. Söll, T. Korak, M. Muehlberghuber, and M. Hutter, "EM-Based Detection of Hardware Trojans on FPGAs," in *Proc. of HOST'14*, May 2014, pp. 84–87.

[107] E. Stavinov. (2011) Using Xilinx Tools in Command-Line Mode. [Online]. Available: http://outputlogic.com/xcell_using_xilinx_tools/74_xperts_04.pdf

[108] M. M. Tehranipoor, U. Guin, and D. Forte, *Counterfeit Integrated Circuits.* Springer, 2015.

[109] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust.* Springer Science + Business Media, 2012.

[110] K. Tiri, M. Akmal, and I. Verbauwhede, "A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards," in *Proc. of ESSCIRC'02*, Sep. 2002, pp. 403–406.

[111] K. Tiri and I. Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation," in *Proc. of DATE'04*, 2004.

[112] R. Torrance and D. James, "The State-of-the-Art in IC Reverse Engineering," in *Proc. of CHES'09*, 2009, vol. 5747, pp. 363–381.

[113] S. Vaudenay, "Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS..." in *Proc. of EUROCRYPT'02*, 2002, vol. 2332, pp. 534–545.

[114] J. Viega and D. Mcgrew, "The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)," RFC 4106 (Proposed Standard), Internet Engineering Task Force (IETF), Jun. 2005.

[115] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting Malicious Inclusions in Secure Hardware: Challenges and Solutions," in *Proc. of HOST'08*, Jun. 2008, pp. 15–19.

[116] H. Wu and T. Huang, "The Authenticated Cipher MORUS (v1)," http://competitions.cr.yp.to/round2/morusv11.pdf, Aug. 2015.

[117] H. Wu and B. Preneel, “AEGIS: A Fast Authenticated Encryption Algorithm (v1),” http://competitions.cr.yp.to/round1/aegisv1.pdf, Mar. 2014.

[118] ——, “AEGIS: A Fast Authenticated Encryption Algorithm,” in *Proc. of SAC’13*, 2014, vol. 8282, pp. 185–201.

[119] B. Yang, S. Mishra, and R. Karri, “High Speed Architecture for Galois/Counter Mode of Operation (GCM),” 2005. [Online]. Available: https://eprint.iacr.org/2005/146.pdf

[120] C. Zhang, L. Li, J. Xu, and Z. Wang, “High-throughput GCM VLSI architecture for IEEE 802.1ae applications,” in *Proc. of ISCAS’09*, May 2009, pp. 900–903.

[121] G. Zhou, H. Michalik, and L. Hinsenkamp, “Improving Throughput of AES-GCM with Pipelined Karatsuba Multipliers on FPGAs,” in *Proc. of ARC’09*, 2009, vol. 5453, pp. 193–203.

Curriculum Vitae

Michael Mühlberghuber was born in Salzburg, Austria, in 1984. He received a Bachelor's and Master's degree from Graz University of Technology (TU Graz) in 2009 and 2011, respectively. His master thesis was part of a collaboration between the Institute of Applied Information Processing and Communications (IAIK) at TU Graz and the Integrated Systems Laboratory (IIS) at ETH Zurich. In 2011, he joined the IIS as a research assistant. His research interests include hardware Trojans and the development of VLSI architectures of cryptographic primitives, targeting resource-constrained environments and high-performance applications.