HARDWARE SECURITY DESIGN
FROM CIRCUITS TO SYSTEMS

BY

S. T. CHODEN KONIGSMARK

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:

Professor Deming Chen, Co-Chair
Professor Martin D. F. Wong, Co-Chair
Professor William H. Sanders
Professor David M. Nicol

ABSTRACT

The security of hardware implementations is of considerable importance, as even the most secure and carefully analyzed algorithms and protocols can be vulnerable in their hardware realization. For instance, numerous successful attacks have been presented against the Advanced Encryption Standard, which is approved for top secret information by the National Security Agency. There are numerous challenges for hardware security, ranging from critical power and resource constraints in sensor networks to scalability and automation for large Internet of Things (IoT) applications.

The physically unclonable function (PUF) is a promising building block for hardware security, as it exposes a device-unique challenge-response behavior which depends on process variations in fabrication. It can be used in a variety of applications including random number generation, authentication, fingerprinting, and encryption. The primary concerns for PUF are reliability in presence of environmental variations, area and power overhead, and process-dependent randomness of the challenge-response behavior.

Carbon nanotube field-effect transistors (CNFETs) have been shown to have excellent electrical and unique physical characteristics. They are a promising candidate to replace silicon transistors in future very large scale integration (VLSI) designs. We present the Carbon Nanotube PUF (CNPUF), which is the first PUF design that takes advantage of unique CNFET characteristics. CNPUF achieves higher reliability against environmental variations and increases the resistance against modeling attacks. Furthermore, CNPUF has a considerable power and energy reduction in comparison to previous ultra-low power PUF designs of 89.6% and 98%, respectively. Moreover, CNPUF allows a power-security tradeoff in an extended design, which can greatly increase the resilience against modeling attacks.

Despite increasing focus on defenses against physical attacks, consistent security oriented design of embedded systems remains a challenge, as most formalizations and security models are concerned with isolated physical components or a high-level concept. Therefore, we build on existing work on hardware security and provide four contributions to system-oriented physical defense: (i) A system-level security model to overcome the chasm between secure components and requirements of high-level protocols; this enables synergy between component-oriented security formalizations and theoretically proven

protocols. (ii) An analysis of current practices in PUF protocols using the proposed system-level security model; we identify significant issues and expose assumptions that require costly security techniques. (iii) A System-of-PUF (SoP) that utilizes the large PUF design-space to achieve security requirements with minimal resource utilization; SoP requires 64% less gate-equivalent units than recently published schemes. (iv) A multilevel authentication protocol based on SoP which is validated using our system-level security model and which overcomes current vulnerabilities. Furthermore, this protocol offers breach recognition and recovery.

Unpredictability and reliability are core requirements of PUFs: unpredictability implies that an adversary cannot sufficiently predict future responses from previous observations. Reliability is important as it increases the reproducibility of PUF responses and hence allows validation of expected responses. However, advanced machine-learning algorithms have been shown to be a significant threat to the practical validity of PUFs, as they can accurately model PUF behavior. The most effective technique was shown to be the XOR-based combination of multiple PUFs, but as this approach drastically reduces reliability, it does not scale well against software-based machine-learning attacks. We analyze threats to PUF security and propose PolyPUF, a scalable and secure architecture to introduce polymorphic PUF behavior. This architecture significantly increases model-building resistivity while maintaining reliability. An extensive experimental evaluation and comparison demonstrate that the PolyPUF architecture can secure various PUF configurations and is the only evaluated approach to withstand highly complex neural network machine-learning attacks. Furthermore, we show that PolyPUF consumes less energy and has less implementation overhead in comparison to lightweight reference architectures.

Emerging technologies such as the Internet of Things (IoT) heavily rely on hardware security for data and privacy protection. The outsourcing of integrated circuit (IC) fabrication introduces diverse threat vectors with different characteristics, such that the security of each device has unique focal points. Hardware Trojan horses (HTH) are a significant threat for IoT devices as they process security critical information with limited resources. HTH for information leakage are particularly difficult to detect as they have minimal footprint. Moreover, constantly increasing integration complexity requires automatic synthesis to maintain the pace of innovation. We introduce the first high-level synthesis (HLS) flow that produces a threat-targeted and security enhanced hardware design to prevent HTH injection by a malicious foundry. Through analysis of entropy loss and

criticality decay, the presented algorithms implement highly resource-efficient targeted information dispersion. An obfuscation flow is introduced to camouflage the effects of dispersion and reduce the effectiveness of reverse engineering. A new metric for the combined security of the device is proposed, and dispersion and obfuscation are co-optimized to target user-supplied threat parameters under resource constraints. The flow is evaluated on existing HLS benchmarks and a new IoT-specific benchmark, and shows significant resource savings as well as adaptability.

The IoT and cloud computing rely on strong confidence in security of confidential or highly privacy sensitive data. As (differential) power attacks can take advantage of side-channel leakage to expose device-internal secrets, side-channel leakage is a major concern with ongoing research focus. However, countermeasures typically require expert-level security knowledge for efficient application, which limits adaptation in the highly competitive and time-constrained IoT field. We address this need by presenting the first HLS flow with primary focus on side-channel leakage reduction. Minimal security annotation to the high-level C-code is sufficient to perform automatic analysis of security critical operations with corresponding insertion of countermeasures. Additionally, imbalanced branches are detected and corrected. For practicality, the flow can meet both resource and information leakage constraints. The presented flow is extensively evaluated on established HLS benchmarks and a general IoT benchmark. Under identical resource constraints, leakage is reduced between 32% and 72% compared to the baseline. Under leakage target, the constraints are achieved with 31% to 81% less resource overhead.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisers, Prof. Martin D. F. Wong and Prof. Deming Chen, who have been great mentors and always encouraged me to tackle new challenges and think about problems from different angles. I am especially grateful for their support for my request to complete the latter stages of my PhD remotely.

In addition to my advisers, I am very grateful to my doctoral committee, Prof. Bill Sanders and Prof. David Nicol. Meaningful new ideas and research directions were the result of discussions with them; this was incredibly valuable to my thesis.

I have been incredibly lucky to work with Mrs. Leslie K. Hwang, who has been the greatest mentor to me from the start of my graduate studies. This dissertation would not have been possible without our fruitful research discussions and her ongoing support in all aspects of my graduate research and beyond.

I wish to thank my co-workers at Microsoft and Google for creating an enjoyable and creative atmosphere during work hours. I am particularly grateful to Dr. Jeremy Condit, Mr. Dan Delorey, Ms. Danielle Hanks, Mr. Eugene Koblov, Mr. Wade Lamble, Mr. Srivastava Nanduri, Mr. Xuejian Pan, and Mr. Minh Tran.

Additionally, I would like to thank all research group members for the meaningful discussions and their support, especially Mr. Keith Campbell, Dr. Yao Chen, Mr. Ashutosh Dhar, Dr. Yuelin Du, Mr. Yi Liang, Mr. Chen-Hsuan Lin, and Dr. Ting Yu.

Finally, I owe my deepest gratitude to my parents and my wife. My parents have always encouraged me to strive for greater challenges and have made me believe that nothing is impossible. My wife, Paema Konigsmark, has always been supportive of my goals, and has been incredibly patient and understanding during the entire length of my graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background and Motivation

Computer security is of utmost importance to our society due to the constantly increasing reliance on ubiquitous computing. With emerging devices in areas such as health care, wireless sensor networks, wearable devices, and the Internet of Things, privacy is no longer the primary concern, and the confidentiality and authenticity of data and devices has to be guaranteed to avoid physical damage.

Secure hardware is of fundamental importance to computer security, as it provides the foundation on which algorithms and protocols are built. If the hardware components are insecure, any higher-level implementation is inherently vulnerable. For instance, a cryptographic module that implements the Advanced Encryption Standard (AES), which is approved by the U.S. National Security Agency (NSA) for top-secret documents, was successfully attacked through analysis of side-channel information leakage.

One promising building block for hardware security is the physically unclonable function (PUF). It implements a device-unique hardware fingerprint that has applications ranging from authentication to random number generation. As PUFs rely on physical variations to generate a device-specific fingerprint, most proposed PUF designs suffer from reliability concerns. The major weakness of PUFs is their vulnerability against machine-learning attacks.

In addition to the threat of invasive attacks, ongoing outsourcing efforts in integrated circuit manufacturing have increased the threat of information leakage due to hardware Trojan horses. It has been shown that minimal area footprint is required to introduce a hardware Trojan horse which has the capability of leaking internal bits in an AES implementation, which could lead to the full disclosure of the secret key employed in encryption.

The amount of power consumption in a circuit depends primarily on the dynamic switching behavior. When the output of a gate is toggled, there are notable spikes in the

power trace. This correlation between device behavior and power consumption can be exploited in side-channel analysis attacks. Without any modifications to the circuit or insertion of hardware Trojan horses, successful side-channel analysis attacks have been launched against AES implementations. While mitigating defense techniques exist, for example by normalizing the power consumption, these techniques are very costly and require detailed security understanding for efficient application.

1.2 Overview of this Dissertation

In this dissertation, we study the three primary threats to hardware security: invasive physical attacks which can change device behavior after manufacturing; insertion of hardware Trojan horses by a malicious foundry which can leak secret information with minimal footprint; side-channel analysis of power traces to reveal device-internal secrets without any noticeable modification to the device.

We propose the first carbon-nanotube (CNT) based PUF in Chapter 2. It exhibits strong cryptographic characteristics while reducing power consumption compared to conventional designs by taking advantage of the inherent uniqueness of the metal-to-semiconductor ratio of CNTs in a carbon-nanotube field-effect transistor (CNFET).

We introduce a System-of-PUFs (SOP) in Chapter 3. The SoP utilizes the difference among multiple proposed PUF designs to improve the resistance against machine-learning attacks and introduces a new multi-level authentication scheme that allows recovery from attacks.

We introduce the first polymorphous PUF (PolyPUF) in Chapter 4. PolyPUF dynamically changes its runtime behavior to defend against machine-learning attacks while still allowing a trusted third party to perform authentication. Through this polymorphous behavior, it can achieve resistance against machine-learning attacks beyond any existing PUF design or architecture. We evaluate PolyPUF against neural network based machine-learning attacks and demonstrate its strength.

The defense against hardware Trojan horses (HTH) as part of high-level synthesis is presented in Chapter 5. HTH insertion by a malicious foundry is a significant threat due to ongoing outsourcing of integrated circuit manufacturing. Through an HTH, information can be leaked with minimal footprint in area and power consumption, which leads to a very low detection probably in post-manufacturing. The high-level synthesis flow mitigates this risk

by automatically detecting vulnerable circuit areas and increasing the difficulty of HTH insertion by dispersing security critical information.

A high-level synthesis flow to protect against power analysis attacks is introduced in Chapter 6. It has been shown that hardware implementations can leak significant side-channel information which can be extracted by analyzing power traces. For example, successful attacks against the advanced encryption standard were demonstrated. Defending against side-channel analysis is a difficult problem, as power consumption depends on several factors and is highly dependent on the underlying data being processed. Existing defense mechanisms are very resource intensive and require deep design and security understanding for efficient application. The proposed high-level synthesis flow identifies operations with high leakage potential and effectively applies countermeasures to achieve high resilience against side-channel leakage with strongly reduced resource requirements.

CHAPTER 2
CARBON NANOTUBE PUF

2.1 Introduction

Modern life depends heavily on electronics. Not only are companies' valuable and confidential assets stored and managed by technology but also our daily lives are connected with technology. Therefore, our privacy and confidential assets are vulnerable to attacks against the technologies we use. This trend leads to an increased interest in security. In addition to these common security concerns, wireless sensor networks and wearable technology have emerged as trends in new devices and can pose significant security risks for our society. Nodes in a sensor network are typically exposed to the public and can contain or handle sensitive data, e.g. power grid information or military defense mechanisms [1]. Wearable technology is emerging as part of ubiquitous computing and may accumulate as much information as the actual wearer, which represents a threat against privacy. Due to the nature of these new devices, they not only require higher security and privacy, but are also critically limited in circuit area, power, and energy budgets. This trend was already observed in current mobile devices, such as smartphones or tablets, but wearable technology tightens these constraints [2].

Software security typically assumes correctness and security of hardware and can only discover hardware based intrusions on a very limited scale [3]. Hardware security provides the building block for secure devices and aims to reduce hardware vulnerability to imaging, probing and intrusion. It is generally designed to take advantage of each chip's unique physical aspects.

Gassend et al. introduced the concept of silicon based physically unclonable functions (PUFs)[4], which has gained attention as an emerging hardware security technology. It maps a digital input, considered to be a challenge, to a digital output, defined as the response, based on intrinsic physical parameters of the circuit. Therefore, the mapping between the input and output of a PUF is called the challenge-response behavior.

A major advantage of PUF is the fast and simple response generation, but extremely difficult challenge-response prediction and duplication. Although manufactured instances of PUF share an identical design, the manufacturing process introduces unpredictable

variations to the intrinsic physical parameters of the chip. Along with the large size of the challenge-response space, this leads to nearly impossible challenge-response behavior replication [5].

Various PUF designs were studied using different physical parameters of the device; silicon PUF [4], arbiter PUF [6], ring-oscillator PUF (RO-PUF) [7], butterfly PUF [8], clock PUF [9], and low-power current-based PUF [10] are examples of PUF designs taking advantage of very different circuit characteristics. The first silicon PUF, presented by Gassend et al., uses the delay of wires and digital logic devices within one circuit. RO-PUF is also designed to evaluate and compare inherent delay characteristics of wires and transistors, but compares distinct circuits. It is based on ring oscillators and uses the unique oscillation frequency for response generation. Butterfly PUF is based on FPGA-specific physical variations. ClockPUF is designed using the clock skew at the sink of the clock network. Ultra-low power current-based PUF converts analog current variations to unique digital quantities.

Critical characteristics of PUF are reliability and uniqueness. The former is measured as the reverse of the average Hamming distance of a single chip under varying environment conditions and the same challenge. The latter is the average Hamming distance between multiple manufactured instances of the same design, and is desired to be 50%.

Most of the existing PUFs focus on conventional silicon devices, and several of them are not geared towards low power operation. Furthermore, as technology moves forward, silicon devices for low-power, high-speed applications are facing a miniaturization bottleneck. Carbon-based structures, such as carbon nanotubes (CNT), are one of the promising emerging technologies that are considered as possible replacements for current silicon technology. Moreover, CNTs have great potential in flexible or wearable electronics [11].

In this chapter, we present a novel carbon-nanotube based PUF (CNPUF), which uses carbon nanotube field-effect transistors (CNFET). Our contributions and the advantages of CNPUF are as follows:

- A PUF design that takes advantage of its unique CNFET characteristics and actively uses metallic CNTs, which are currently inevitable but typically considered a major issue for digital designs.
- Considerable reduction in footprint by using CNFET-unique properties to reduce the transistor count.

- Extremely low power and energy consumption that is 89.6% and 98% lower than ultra-low power current based PUF [10] at 90 nm.
- Very high reliability against environmental variations and SPICE-accurate experimental evaluation in two different settings.
- An extended design that enables a power-security tradeoff, highly relevant for practical usage scenarios.
- Evaluation of PUF behavior with regard to different CNT technology parameters

The rest of this chapter is structured as follows. In section 2.2, we provide background knowledge and explain CNT behavior and characteristics. In section 2.3, we propose CNPUF and theoretically evaluate it. An extension of CNPUF, which can be used for high security applications, is presented in section 2.4. A SPICE accurate experimental evaluation and comparison is provided in section 2.5. Finally, we summarize our findings and give an outlook in section 2.6. This chapter is based on [12].

2.2 Background

Carbon nanotubes (CNTs) are cylindrical carbon molecules that have superior electrical, mechanical and thermal properties [13]. Thus, CNT technology is considered as one of the potential candidates for future electronics [13]. CNFETs, first introduced by S. Tans et al. [14], are transistors with channels consisting of CNTs instead of bulk silicon. Conventional methods in technology scaling will likely encounter physical limitations and these molecular electronics have attracted much interest as they can lead to further technology scaling.

Regardless of the superior properties, there exist fundamental limitations and obstacles in fabrication of CNTs. Due to intense research studies, yield and performance of CNFETs are fast improving and they will be realized as digital circuits in the near future [15]. Most recently, a first fully CNT based subsystem was presented [16], and a first digital carbon nanotube computer was created [17]. Nevertheless, it is still impossible to guarantee perfect alignment, semiconducting property, and uniform distribution, which lead to performance variations [18]. The major CNT variations [19] are: (i) chirality, which defines the type to metallic or semiconducting; (ii) diameter; (iii) growth density; (iv) alignment; (v) doping concentration.

Some of these variations correlate with one another and all of them result in electrical property variation and can lead to malfunction of digital circuits in the worst case. Particularly the lack of chirality (and thus type) control is a major issue for CNT usage in

digital circuits, as metallic CNTs in transistors lead to direct drain-to-source shorting. Due to the lack of precise chirality control, removal of metallic CNTs became necessary [20]. However, removal of undesired CNTs can lead to further circuit variations and even malfunctioning. For CNPUF design, we inherently consider this CNT specific property and take advantage of the type variation for extremely high efficiency digital secret generation.

The CNT chirality is a pair of indices (n, m) and represents the 2-dimensional wrapping of graphene and determines the type to either metallic or semiconducting [21]. Based on the chirality, CNTs are categorized into three categories. If $n = m$, it is called an *armchair* nanotube. Another structure is *zigzag* nanotube with $m = 0$. Otherwise, it is a *chiral* nanotube. For a given chirality (n, m), a CNT is metallic when it satisfies either of following two cases.

1) $n = m$
2) $n - m = 3N$ for any $N \epsilon \mathbb{N}$

Metallic and semiconducting CNTs have different impacts on different operating modes. When the CNFET is turned off, the metallic CNTs are still conducting, as they lead to a direct gate-source shorting. However, both metallic and semiconducting CNTs contribute when the transistor is on. Hence, the ratio of the semiconducting to metallic CNT is closely related to the on and off current (I_{on}, I_{off}) ratio. By utilizing this CNT specific characteristic, our CNPUF design provides a simple, but unconventional and extremely energy efficient security solution.

2.3 Carbon-Nanotube PUF

In this section, we will explain the CNFET based PUF design. Subsection 2.3.1 gives an overview of the basic design and explains the challenge-response behavior. Then, the main internal block of the CNPUF design, the CNPUF Parallel-Element (CNPUF-PE), will be discussed in subsection 2.3.2. An analysis of area cost is provided in subsection 2.3.3. In subsection 2.3.4, we will elaborate on the design characteristics responsible for providing high reliability, which is experimentally shown in section 2.5. Subsection 2.3.5 shows the complexity of the PUF design and the dimensions that affect the challenge-response behavior, which leads to resistance against modeling attacks.

2.3.1 Basic Design

The design of the CNPUF is shown in Figure 2.1. We define a CNPUF Parallel-Element (CNPUF-PE) as a pair of CNFETs that share the same gate voltage, which is an input to the CNPUF-PE. These inputs form the challenge. Each bit of the challenge is associated with a single CNPUF-PE, by directly providing the input challenge as the gate voltage. A high gate voltage corresponds to logic 1, and a low gate voltage corresponds to logic 0.

Each CNPUF-PE has two distinct states, one for a high input gate voltage and one for a low voltage. These two states differ among all CNPUF-PEs because of the static variations caused by the manufacturing process. A detailed explanation of the variation sources is provided in subsection 2.3.5. The output bit of CNPUF is generated by comparing two currents through a series connection of CNPUF-PEs. In Figure 2.1, when $I_1 > I_2$, the output is 1; otherwise, it is 0.

CNPUF can be used in different configurations to achieve multi-bit responses: The simplest approach is to replicate and parallelize a one-output-bit CNPUF to achieve multiple output bits. While this appears costly, each output bit is truly generated by different physical circuits and thus they are fully independent of each other. An area saving alternative to this approach can be achieved by reusing a one-bit CNPUF with a pseudo random number generator (PRNG) as a challenge translator. However, the output bits are no longer independent and the area saving can be very limited, as the CNPUF can be implemented with a small number of transistors, whereas a multi-bit PRNG can introduce high area overhead.

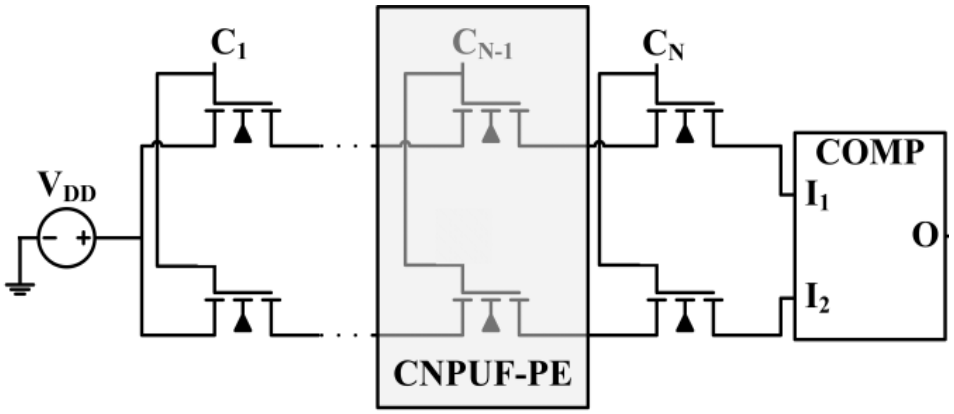

Figure 2.1 Row of the CNPUF design. A series of CNPUF-PE are evaluated by a comparator (COMP) to generate the output bit.

2.3.2 CNPUF Parallel-Element

The CNPUF mainly consists of a serial connection of CNPUF-PEs. Internally, the CNPUF-PE consists of two parallel CNFETs that share the same gate voltage. Each CNFET consists of a large number of semiconducting CNTs and a few metallic CNTs, typically with a metallic-to-semiconducting ratio between 10% and 33% [3]. Due to this difference, a CNPUF-PE has two distinct and nearly independent states for a high gate voltage and a low gate voltage. The current characteristics for a low gate voltage are dominated by the metallic CNTs, as the off-current for semiconducting CNTs is considerably lower than the current for metallic CNTs, even when the number of semiconducting CNTs is large. For a high gate voltage, the semiconducting CNTs dominate the current characteristics due to their much larger number. As CNT technology improves, we expect the ratio of semiconducting to metallic CNTs to increase, and thus the correlation between both states to further reduce.

2.3.3 Area Comparison

In practice, the number of challenge bits is larger than 128 bit. In this case, the area cost per bit of a one-output-bit CNPUF, $A_{CNPUF,bit}$, is approximately the area of a CNPUF-PE and thus two times the area of a transistor, $A_{Transistor}$:

$$A_{CNPUF,bit} \cong 2 * A_{Transistor}$$

This compares favorably to the basic implementation of the Arbiter PUF [1], which requires two multiplexers per input bit (assuming that the challenge bit length is large enough to neglect the Arbiter). Using a transmission gate implementation, the area cost per bit of a basic Arbiter PUF, $A_{A-PUF,bit}$, is

$$A_{A-PUF,bit} \cong 2 * A_{MUX} \cong 8 * A_{Transistor}$$

Here, A_{MUX} is the area of a 2:1 multiplexer (or MUX).

The area advantage of CNPUF is even larger when compared with a ring oscillator PUF (RO-PUF) [2]: A RO-PUF requires at least $2^N - 1$ ring oscillators (ROs) for N input bits. Considering the usage of very small ROs consisting of only three inverters, the cost of a RO-PUF per bit $A_{RO-PUF,bit}$ depends on the challenge length N and is

$$A_{RO-PUF,bit} \cong \frac{(2^N - 1)}{N} * 6 * A_{Transistor}$$

2.3.4 High Reliability

The reliability of PUF designs is typically evaluated with regard to variations in environmental parameters, such as the temperature, and variations in the operating conditions, e.g. the supply voltage. As explained in section 2.2, CNTs have exceptional electrical characteristics and are known to have high stability under environment variations. The reliability of our proposed CNPUF builds on the highly stable characteristics of CNTs, but has two additional features that support reliability:

- Strong impact of physical variations
- Regular design

The physical variations are large and have a strong impact on the PUF challenge-response behavior. Particularly the ratio variations between semiconducting and metallic CNTs have a very strong impact on the circuit behavior. Our design takes advantage of the difficulty of controlling CNT chirality to create a secret by comparing the current through a series connection of CNPUF-PE. Since metallic and semiconducting CNTs are randomly mixed during the fabrication process, the exact ratio of metallic and semiconducting CNTs for each FET is not predictable and can be treated as a random number.

Except for the comparator, our design is very regular, and therefore the dynamic effects in the upper and lower path can average out. Only the comparator has to be specifically designed to be resilient to dynamic variations such as temperature changes or voltage peaks. In comparison, a ring oscillator PUF has MUXes, counters and also a comparator that have to be tuned, because they affect both paths and therefore have to treat these paths equally. This can be difficult in practice, as the variations on different metal layers and in different chip regions can be different, so that the number of circuit elements that require fine tuning has to be minimized.

2.3.5 Resistance against Modeling Attacks

The proposed CNPUF design takes advantage of inherent CNT properties and variations to represent a complex challenge-response behavior to modeling attacks. Some of this is due to the many varying physical factors as described in section 2.2. Even more important is that these varying physical properties can have a different level of impact on the challenge-response behavior. Therefore, a model based attack has to accurately identify and mirror different physical characteristics. Due to an increase in this dimension, CNPUF has a high

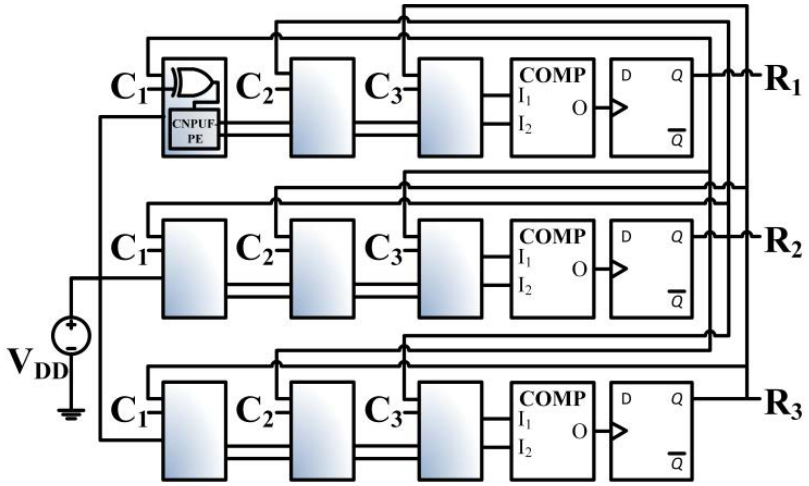

Figure 2.2 Design of the extended CNPUF (ex-CNPUF) to allow a power vs. security tradeoff and more complex challenge-response behavior. All shaded blocks contain the XOR-gate and CNPUF-PE structure shown to the top left.

resistance against such attacks. However, the basic implementation of CNPUF has independent output bits and thus does not take full advantage of the existing circuitry. Furthermore, some applications, such as secret key generation, require very high randomness and must have an even more complex challenge-response-behavior. Therefore, we also present an extended version of CNPUF that introduces a power vs. security tradeoff and has area overhead to create a feedback based design with even higher modeling complexity.

2.4 Extended CNPUF

2.4.1 Extended Design

While the basic CNPUF has many desirable properties, such as low power consumption and minimal area requirements, its static nature does not allow fine-tuning by the designer to specific application needs. Therefore, we also present the extended CNPUF (ex-CNPUF), which is shown in Figure 2.2. As a basic element, the ex-CNPUF contains one basic CNPUF per output bit, shown in Figure 2.1. However, ex-CNPUF buffers each bit response and feeds it back into the CNPUF-PE elements through an XOR-element with the original challenge. This may be repeated a specific number of times for each challenge, depending on the application specific requirements. As a result of this response feedback, the complexity at the bit level increases, which further improves the resistance against modeling

attacks. Furthermore, the output bits will no longer be independent of each other, a feature that drastically increases the design complexity a modeling attack has to consider.

The additional area overhead compared to the basic CNPUF is one XOR gate per CNPUF-PE element. Considering that the CNPUF-PE was designed for minimal area and consists of only two transistors, the area overhead of the extended design is approximately 300%. The area of ex-CNPUF is:

$$A_{ex-CNPUF} \approx 8A(T)$$

Although the area overhead over CNPUF is not negligible, ex-CNPUF still compares favorably to several PUF implementations. It has approximately the same area as a simple arbiter PUF and less area than an RO-PUF. However, the greatest advantage is the power-security tradeoff, which is explained in subsection 2.4.2.

2.4.2 Power-Security Tradeoff

As shown in subsection 2.4.1, ex-CNPUF brings several advantages in flexibility and security, but introduces area overhead relative to the basic CNPUF design. The design of ex-CNPUF is very feasible for high-security applications that require more modeling resistance or higher randomness than what CNPUF can provide. By increasing the number of ex-CNPUF iterations, the complexity and randomness of the response increases. This comes at the cost of higher energy consumption and reduced reliability. The reduced reliability is a result of error propagation within the PUF and the fact that even a single bit-error in the challenge can profoundly change the response, if no error correction schemes are used. Note that the area tradeoff is static and has to be made at design time, but the power tradeoff can be dynamically adjusted. Thus, ex-CNPUF can power a security interface that provides different degrees of resistance for different domains.

2.5 Experimental Evaluation

2.5.1 Simulation Setup

For the experimental evaluation of CNPUF, we simulated the design in HSPICE[1] in a Linux environment. To simulate the CNFETs contained in CNPUF, we employed the Stanford CNFET HSPICE model [22], [23]. For standard logic and comparison purposes, we employed the Predictive Technology Model (PTM)[24]. Our CNT simulation is based on zigzag structure with a nominal chirality of metallic nanotubes of $(n, m) = (10, 0)$ and a nominal chirality of semiconducting nanotubes of $(9, 0)$.

As a solution to the lack of support for metallic CNTs in the HSPICE model, we simulate real CNFETs by splitting them into one ideal metallic and one ideal semiconducting CNFET, as shown in Figure 2.3. The metallic CNFET is modeled by assigning the appropriate chirality and setting the gate voltage to always-on, independent of the challenge. Note that this separation is solely for the purpose of experimental evaluation through simulations; one metallic CNFET and one semiconducting CNFET in the simulation model represent a single transistor. In this regard, our design does not require ideal or pure-semiconducting CNFETs, but instead takes advantage of metallic CNTs.

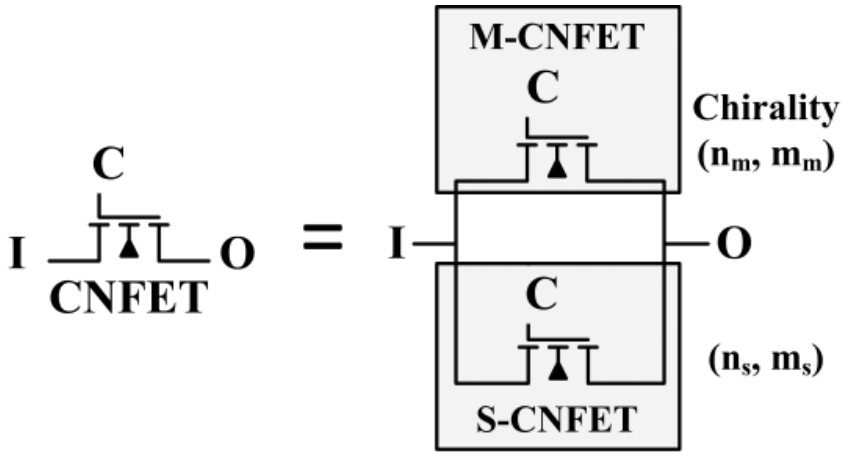

Figure 2.3 Simulation model for the metallic CNTs in the CNFETs contained in CNPUF.

[1] HSPICE Version E-2010.12-SP2 32-BIT

For this experimental evaluation, we implemented an 8-Bit input/output CNPUF and analyzed reliability, inter-chip variability, and power consumption. A small design was chosen, such that a large number of SPICE-accurate simulations can be performed to provide detailed insight into the reliability under environment variations. As a proof of concept, we have also evaluated a larger design of 128-bits and observed in a smaller number of simulations that the behavior is very similar to the 8-bit implementation.

2.5.2 Reliability

Reliability is one of the main criteria for PUF quality and quantifies the capability to repeatedly and consistently produce the same challenge-response behavior. As explained in section 2.2, CNTs generally have a high reliability that is further improved by specific design measures described in section 2.3.4. To experimentally validate the reliability, CNPUF is evaluated under a standard simulation environment that is used in most literature, and a simulation environment with more dynamic variations to simulate real circuit performance.

The standard simulation environment is similar to that employed in several publications [10], [25] and has the following characteristics:

- *Temperature variations*: The temperature of the whole circuit is evaluated at specific temperature points or at random temperature values.
- *Supply voltage variations*: The supply voltage of the whole circuit is varied. This is implemented by having a common random voltage variation with $\sigma_{V,supply}$ at every voltage source (including challenges).

To provide a stronger comparison with implemented PUF circuits, we also propose an extended simulation environment with the following parameters:

- *Dynamic temperature*: In addition to the static temperature variation, we model dynamic temperature variation by adding a different random temperature T_{rand} to the static temperature for each simulation. Therefore, we compare challenges that were acquired at different temperatures.
- *Local voltage variation*: Simulation of dynamic local voltage variation by introducing a local variation $\sigma_{V,local}$ at each voltage source and at each gate in addition to the common voltage variation.

Table 2.1 Experimental results for reliability

Nominal Metal Ratio	Intra Chip Hamming Distance HD_{intra}			
	Basic CNPUF		Ex-CNPUF	
	Std Env	Ext Env	Std Env	Ext Env
10%	0.025	0.040	-	-
20%	0.019	0.034	0.045	0.05
30%	0.012	0.030	-	-
Total Average	0.019	0.035	0.045	0.05

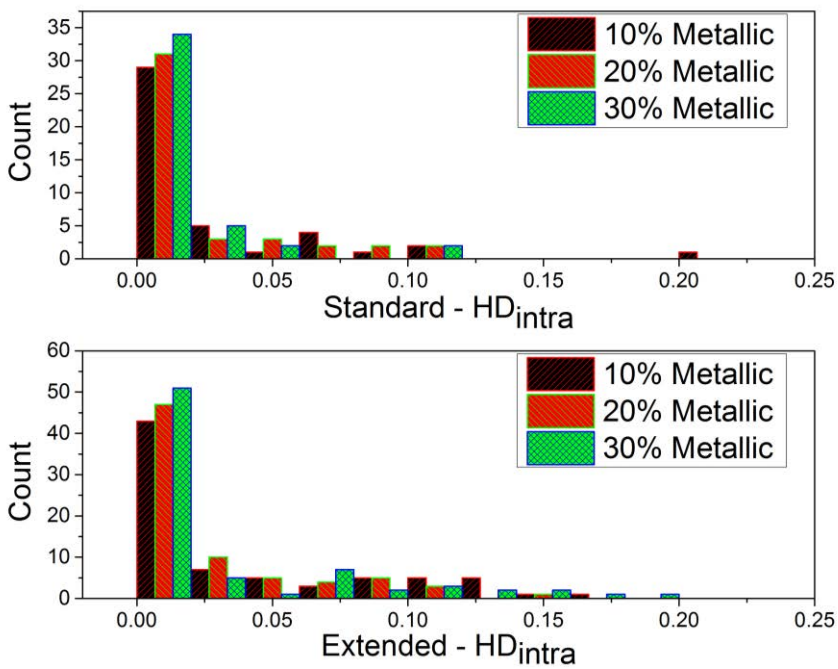

Figure 2.4 Robustness of CNPUF in a standard simulation environment (top) and in an extended simulation environment (bottom).

This is particularly relevant to real physical devices, as there is additional circuitry on chip besides the PUF that can influence the power and temperature with different levels of activity. The experimental data for both simulation environments is shown in Figure 2.4. The detailed averages for each ratio are provided in Table 2.1. The range for these parameters is shown in Table 2.3. For the standard environment, we conducted over 3500 HSPICE simulations and evaluated the intra-chip Hamming distance to:

Table 2.2 Power and energy comparison between CNPUF and ultra -low power current-based PUF [10] at 14nm and 90nm.

Designs	**CNPUF**		**Current based PUF [10]**	
Technology	90nm, 1.2V	14nm, 0.8V	90nm	14nm, 0.8V
Power	15.6μW/bit	1.26μW/bit	150μW/bit	24μW/bit
Delay	43ps	26.5ps	250ps	~5ps
Energy	0.67fJ/bit	0.0334fJ/bit	37.5fJ/bit	0.12fJ/bit

Table 2.3 Simulation parameters for CNPUF.

Parameter	**Range**
Temperature T	$-20° \; to \; 80°C$
Dyn. Temp. T_{rand}	$0° \; to \; 20°C$
Voltage variation	$\mu = 0.8V$ $3\sigma_{supply} = 22.5\%$ $3\sigma_{dynamic} = 7.5\%$
CNT ratio variation	$\mu_{ratio} = \{0.1, 0.2, 0.3\}$ $3\sigma_{phy} = 22.5\%$
Channel length variation	$\mu_{channel} = 14nm$ $3\sigma_{channel} = 22.5\%$

$$HD_{Intra,std} = 1.9\%$$

In the extended simulation environment for accurate comparison against actual circuit implementations, the intra-chip Hamming distance was determined from more than 6000 HSPICE simulations:

$$HD_{Intra,ext} = 3.5\%$$

The reported data was gathered for three different ratios of metallic CNTs to semiconducting CNTs, as the ratio between metallic and semiconducting CNTs can have different nominal values. This ratio is typically between 10% and 33%; therefore, we provide evaluations with 10%, 20%, and 30% ratios and show that CNPUF is viable in the whole range. The graph further shows that a higher metallic ratio actually leads to a slightly higher reliability. This is an effect of increasing dominance of the metallic CNTs, as their share becomes larger. To achieve a well-balanced design, we therefore propose the usage of CNPUF with a nominal metallic ratio of around 20%, which can be achieved without any breakdown in current technology.

Due to simulation complexity, ex-CNPUF was only evaluated at a 20% metallic ratio. As expected, the reliability slightly decreases as the complexity of the design increases. However, the intra-chip Hamming distance is still well below the 10% typically considered a limit for error correction [9], and competitive with other PUF designs

2.5.3 Inter-chip Variability

For security applications, different PUF instantiations require sufficiently different challenge-response behavior, so that the behavior of one PUF instance may not be inferred through ownership of another. Ideally, all pairs of responses from different instances would share 50% of their output bits on average. As a metric for this variability between physical instances, the inter-chip Hamming distance, HD_{inter}, is used.

For the generation of HD_{inter}, we created 10 groups of 10 PUF instances. To each group of PUF, we issued 100 randomly generated challenges. The average HD_{inter} of CNPUF is 49.67% and therefore very close to perfect (50%).

2.5.4 Power Consumption

By using intrinsic CNFET-unique properties, such as the metallic-semiconducting ratio, CNPUF allows secret-key generation at a very low cost. An area comparison was provided in section 2.3.3 and showed that CNPUF requires less logic than other PUF designs. These advantages combine to greatly reduce the power and energy consumption, as shown in Table 2.2. Based on SPICE-accurate simulations, we report that CNPUF achieves the highest power and energy efficiency to the best of our knowledge and reduces the power consumption per bit to $1.26\mu W$ and energy consumption to $0.0334 fJ/bit$. We compare CNPUF with ultra-low power current based PUF [10] at 90nm and at 14nm. According to data provided by the authors, CNPUF reduces power by 89.6% and energy by 98% when implemented in 90nm technology. For comparison purposes, we reimplemented [10] into 14nm technology and conducted power and energy measurement under ideal conditions. At this technology node, CNPUF reduces the power by 94.75% and energy by 72.16%.

Table 2.4 Comparison of HD_{intra} in different simulated PUF designs. Lower percentages mean higher robustness.

CNPUF	ScanPUF[24]	ROPUF[9]	ClockPUF[9]	Current PUF[10]
1.9%	5%	9.51%	5.07%	~3%

Table 2.5 Comparison of HD_{intra} between real PUF circuits and CNPUF under extended environment simulation.

CNPUF	Butterfly PUF [8]	SRAM-PUF [25]	ScanPUF [24]
3.5%	6%	~8%-18%	3.2% (*)

However, it is very likely that the quality of PUF designs that require sub-threshold operation, e.g. [10], will degrade at smaller technology nodes, as susceptibility to environment variations greatly increases.

In Table 2.4, we compare the reliability of CNPUF against other PUF designs with simulated results and show that CNPUF can outperform them. In Table 2.5, the evaluation under extended environment conditions of CNPUF is compared against physical implementations of other PUF designs. Note that the authors only evaluated ScanPUF at a single temperature, which reduces the comparability, as all other designs were evaluated at a wide range of temperatures. The numbers show that in addition to a considerable reduction in area and power consumption that we previously showed, CNPUF can also achieve higher reliability. The inter-chip distance of all PUF designs, including ours, is very comparable and close to the desired 50%.

2.6 Conclusion and Outlook

We presented a PUF design based on intrinsic physical variations of CNTs. It takes advantage of the metallic to semiconducting CNT ratio in CNFETs to increase reliability, while strongly reducing the average power consumption and energy usage per bit. CNPUF was experimentally evaluated with SPICE-accurate simulations and showed strong results for security relevant properties such as reliability and inter-chip distance. Furthermore, we presented and evaluated an extension of CNPUF that allows a power vs. security tradeoff for dynamic usage in high security circuits.

CNPUF and ex-CNPUF provide the future basis for authentication and secret key generation by offering security at a very low area and power cost. This can open the field of PUF for a variety of new applications and is especially relevant for current research areas such as wireless sensor networks or ubiquitous computing.

CHAPTER 3

SYSTEM-OF-PUFS: MULTILEVEL SECURITY FOR EMBEDDED SYSTEMS

3.1 Introduction

With the emergence of ubiquitous computing, the entire society increasingly relies on embedded computing devices in every aspect of life. They enable wireless communication and contactless payments, enhance automobile safety and reliability, and are at the core of the emerging smart-grid. The critical importance of embedded devices drives the growing need for computer security. The emergence of ubiquitous computing has brought new security threats and further increases the importance of security, particularly reliable authentication. Wearable technology and personal medical devices are employed to monitor and augment the functionality of human organs, but an adversary that circumvents authentication protocols can directly impact the physical well-being of users. Moreover, such authentication failures in wireless sensor networks for border control and defense purposes can have hazardous consequences at an international scale. Therefore, the need for continued advances in the domain of computer security is clear.

Software and network security have gained increased attention and are widely perceived to provide the necessary means for secure communication, authentication, and data storage. However, researchers and security professionals have shown that algorithms and protocols that are theoretically proven to be secure are often physically attackable in their implementation. The primary cause for this vulnerability is that implicit high-level assumptions to the hardware, such as information containment and resistivity against physical modifications, are difficult or very costly to achieve. For example, Side-channel information leakage was used to successfully attack AES implementations [26], and secret keys can be extracted from volatile memory long after the device is disconnected from its power source [4]. Moreover, state-of-the-art security and authentication protocols are based on secure storage and usage of secret keys [27]. Therefore, non-volatile memory (NVM) and fuses were used to construct a hardware-based secret key [7]. However, NVM is prone to invasive physical attacks such as focused-ion beam based circuit-edits [28], [29] and non-

invasive imaging attacks [30]. To worsen the security options of embedded systems such as smart-cards and wireless sensors, this new device generation is critically power and resource constrained, allowing only minimal resource allocation for security purposes. Classic cryptographic algorithms are of high complexity and are power intensive, rendering them undesirable for most embedded system applications [30].

Due to the threat of invasive hardware attacks, physically unclonable functions (PUFs) [4] were introduced as a light-weight building block for hardware security. A PUF is a disordered physical system that reacts to an external stimulus or challenge C with a response R, which depends on nanometer-scale intrinsic fluctuations [31]. As this nanometer-scale disorder depends upon unique device-specific properties that originate from random variations during the manufacturing process, the PUF behavior is device-specific. Therefore, two PUF devices with identical (layout) design exhibit different behavior, which defines the unclonability of PUF. Moreover, as the behavior depends upon the exact internal properties of the disordered system, any physical modification or tampering results in modified behavior, which is utilized to achieve tamper resistance. Thus, PUFs do not exhibit the same weaknesses towards non-invasive imaging and invasive attacks. However, PUFs only provide the basic building block for security and have to be incorporated into a system that can participate in a security protocol for tasks such as authentication. Whereas PUFs by themselves have been intensively studied, analyzed, and formalized, there exists no consistent system-level security model that clearly defines and scrutinizes the security of an embedded system. This lack of system-level security model shows in current protocols, which introduce unsecure components into a PUF-based system that can lead to considerable security obstacles and reduced functionality.

This chapter addresses the security demand of embedded systems and introduces a security model and PUF-based authentication protocol. Our unique contributions are:

- A system-level security model for embedded systems with emphasis on invasive and modeling attacks.
- Authentication protocol which operates in multiple levels and alleviates the resource constraints of embedded systems by moving resource intensive components off-chip.
- A heterogeneous System-of-PUF (SoP) that utilizes the large PUF design-space to achieve tamper-resistance and resistivity against modeling attacks without costly active components.

Table 3.1 Comparison of PUF designs in reliability (lower FHD_{intra} is better) and randomness (0.5 for FHD_{inter} is ideal).

PUF designs	**Quality metrics**	
	$\boldsymbol{FHD_{inter}}$	$\boldsymbol{FHD_{intra}}$
ClockPUF [14]	0.503	0.057
SRAM	0.4997	< 0.12
Arbiter PUF	0.51	0.05
4-XOR Arbiter PUF [13]	0.51	0.19
Ring-Oscillator [4]	0.4614	0.0048

- SoP is designed to achieve specific security requirements utilizing the system-level model. It reduces the gate-equivalent cost by 64% compared to an existing design.

The remainder of this chapter is structured as follows. In section 3.2, we review the background of PUF and PUF-based authentication. The main contributions of this chapter begin with section 3.3, where we introduce a system-level security model. In section 3.4, we analyze existing PUF-based protocols and design-techniques. In section 3.5, we propose a SoP with multilevel authentication protocol, followed by a detailed security analysis in section 3.6. An experimental evaluation that shows the feasibility of the protocol is provided in section 3.7. This chapter concludes with a summary in section 3.8. This chapter is based on [32].

3.2 Background

PUFs are characterized by their challenge (input) and response (output) behavior. Typically, a PUF consists of a number of equally designed components that have marginally disparate physical properties due to manufacturing variations. The challenge to a PUF is used to select which PUF components are compared for their physical properties and the response is a bit or bit-string representing the outcome of pairwise comparison of the selected elements.

3.2.1 PUF Quality Metrics and Designs

In this chapter we use inter-chip and intra-chip distances [33] to establish design criteria for PUF. Inter-chip Hamming distance HD_{inter} is a metric for the difference between manufactured PUF instances and thus represents randomness, the usability of static

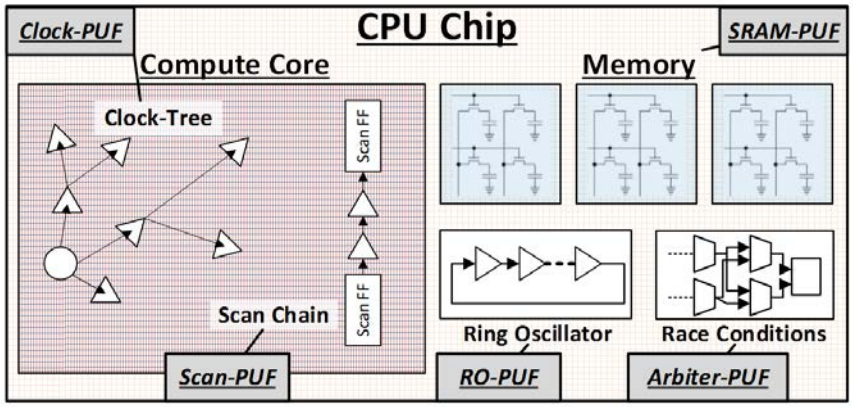

Figure 3.1 Overview of various PUF designs.

variations, e.g. process variations. Ideally, this randomness is 50%, as this would imply that different manufactured instances are uncorrelated. The strict avalanche criterion is another important property by which a single bit-flip in the input leads to a flip of half of the output bits. Intra-chip Hamming distance HD_{intra} represents the variability of the responses of a single PUF instance when issued with the same challenge. Unless $HD_{intra} = 0$, the PUF response contains bit-errors. HD_{intra} is a metric for the robustness of the PUF and lower values indicate higher reliability under environment variations. When averaged over the bit-length, we refer to the fractional Hamming distances FHD_{intra} and FHD_{inter}.

A wide variety of PUF designs have been introduced. Figure 3.1 visualizes some of the components that can be employed as a PUF, and the properties of these designs are summarized in Table 3.1.

SRAM-PUF [34], [35] leverages the device-specific start-up value of a SRAM-cell to provide a device-specific fingerprint. It is a low-cost PUF as it utilizes existing components, and it is suitable for ID or key generation. However, the reliability and uniqueness of SRAM-PUF are typically lower than those of dedicated PUF structures that have been engineered for these properties.

In the Arbiter PUF [7], an output bit is generated from a delay comparison of two equally designed paths. The challenge is used to select a specific path by controlling multiplexers (MUXes) that select between equally designed, but physically different wire segments. Rührmair et al. presented model-building techniques that successfully predicted the behavior of an Arbiter PUF with a small training data-set [36]. *N* Arbiter PUFs can be combined by exclusive-or (XOR) gate connection of the individual response bits (*N*-XOR-Arbiter PUF), for integer *N*. These XOR-Arbiter PUFs increase the difficulty of creating a

model for the PUF, but the XOR combination also linearly reduces the reliability of the PUF [37]. Furthermore, it was shown that a 6-XOR Arbiter PUF is very expensive to model [36].

Ring-oscillator PUFs (RO-PUFs) [7] consist of a number of ring-oscillators that are designed to be identical. Based on the challenge, two MUXes select one ring-oscillator each and their frequency is compared using a counter for each of the oscillators. This comparison determines the output bit of the RO-PUF. RO-PUF has been shown to achieve high reliability, but is comparably power-intensive and slow as many cycles are needed to distinguish the respective oscillator frequencies.

Yao et al. [9] introduced a PUF design based on the clock network of the chip that inherits the stability and inherent reliability of a clock network but introduces enough variations to show randomness. They choose specific sinks in the clock network and branch the clock signal from these sinks and generate a bit from comparison similar to the Arbiter PUF. It should be noted that this PUF design introduces an overhead of approximately 20% on the clock-tree, which is a considerable expense for current technologies due to the complexity of clock networks.

3.2.2 PUF Security Models and Formalization

The quality of PUF research is apparent from the number of security formalizations and models that exist for the PUF as a component. Rührmair et al. [31], [38] have introduced formal security proofs for Strong and Weak PUF designs. Strong PUF is defined to have a very large set of possible challenges with accessible but complex challenge-response interface. Corresponding responses of selected challenges will be paired to identify the correctness, and numerical prediction of a response is strongly prevented. On the other hand, Weak PUF has a small challenge-space. Thus, it does not exhibit the same resistance as Strong PUFs with regard to modeling attacks, but can be utilized to generate a secret key. It is preferred over NVM, as it is based on physical disorder and thus provides inherent tamper resistance.

Based on practical attack scenarios, Rührmair and van Dijk [31] introduced three different PUF models: (1) *stand-alone, good PUF model* that assumes a single isolated protocol execution without malicious intervention such as manipulated hardware; (2) *PUF re-use model* which allows the adversary multiple access to the PUF, but does not consider hardware modifications; (3) *bad PUF model*, which is concerned with scenarios around physically modified PUFs which can be exploited by an adversary.

3.2.3 Helper Circuits

As a basic building block, PUF relies on additional circuitry to enhance or complement its functionality. Although a wide variety of techniques exist, we will only discuss challenge expansion, error correction, and hash-based randomization, due to their importance for authentication protocols.

Most PUF designs only generate one response bit, as the outcome of a random binary variable is evaluated. A resource-intensive approach of extracting multiple response bits such that the response has bit-length l_R is to implement l_R PUFs that are simultaneously stimulated by the identical challenge. For lightweight applications, a pseudo-random number generator (PRNG) can also be used as a challenge expander, e.g. a linear shift feedback register (LFSR) [30], [39]. Based on an initial challenge as the seed, the PRNG will generate a consistent challenge sequence, such that the PUF response with l_R bits is generated by a single PUF structure in l_R time-steps. Thus, this provides for a time-space trade-off, which can be utilized for low-resource embedded systems. It should however be noted that this approach is preferable with a long challenge, such that the probability of encountering the same actual challenges from the PRNG in a short time frame is small. Moreover, the challenge-expander technique is more susceptible to modeling attacks, as it provides less complexity and every bit is generated from the same PUF-structure.

As PUF elements are designed to be equal, their response is volatile and dynamic variations such as temperature or supply voltage variations commonly lead to noise in PUF responses. This noise results in random bit-flips in the response. To maintain correct functionality in the presence of such bit-flips, PUFs can be complemented with error correcting codes (ECC) and fuzzy extractors [40]. Ring-oscillator PUF design employing index-based syndrome encoding (IBS) was shown to have very high reliability characteristics [40]. Further improvements over this exist, which allow a tradeoff between design complexity and error correction [41]. Despite continued research and advances in this domain, these error correcting techniques are very expensive in area and energy compared to the cost of the actual PUF, suggesting that they are less suitable for lightweight applications. This cost overhead is even more pronounced, as error correction leads to certain (predictable) loss of entropy, which in turn requires a larger PUF design.

As described in subsection 3.2.1, randomness is an important property of PUF. When the PUF challenge-response behavior lacks randomness and is easily predictable, it can be complemented by a hash-function that randomizes the PUF responses. On-chip hash

functions are also used to enable authentication without exposing the direct PUF responses [42]. However, it should be noted that on-chip hash functions incur considerable cost in both area and run-time that typically exceed those of the PUF by more than an order of magnitude [30], [43]. Even hash functions that are designed for light-weight hardware implementations incur overhead that exceeds the cost of the PUF itself [44].

3.2.4 PUF-based Authentication Protocols

We use the terminology of [30] and refer to the *Prover* as a device to be authenticated, and the *Verifier* as the trusted party authenticator that judged whether the *Prover* is authentic or not. The authentication protocol determines the interaction between *Prover* and *Verifier*.

A simple authentication protocol based on issuing random challenges with known responses is presented in [7]. Initially, challenge and response pairs are gathered in an enrollment phase. The trusted party can validate the PUF responses against the known responses. To handle man-in-the-middle attacks, it is proposed to only use each challenge once.

Reverse Fuzzy Extractor [39] is a lightweight authentication scheme that attempts to move computationally complex or resource intensive components off the PUF-circuit to the authentication granting authority. It is based on reversing error correction schemes employed to increase PUF reliability.

Another recent approach is Public PUF (PPUF) [45], where detailed physical characteristics of each PUF instance are public, allowing anyone to simulate PUF behavior. A PUF is then verified by not only providing the correct response to a challenge, but doing so in a much shorter time than possible with simulations. As the true response can be simulated before issuing the challenge, no previous CRP storage is required. As PPUF has high latency and power consumption, extensions with the same principle were developed [46]. Due to the computational requirements and detailed device-specific measurements, PPUF is most suitable for small-scale applications.

Slender PUF Protocol is another lightweight protocol [30] that can be used for identification and authentication. The protocol has two main ideas: (i) only substrings of responses are provided, and (ii) the challenges to the PUF are jointly created by both the *Prover* and the *Verifier*. The *Verifier* is assumed to have an ideal model of the PUF, so that a response for any possible challenge can be generated. The substring received from the *Prover* is then used to check whether it indeed is a substring of the real PUF response. The

second idea is that neither party is allowed to solely generate the challenge; thus the challenge comes from a pseudo random number generator (PRNG), and the seed to this PRNG is determined by randomly generated numbers (nonces) from both parties.

Noisy PUF protocol [47], [48] is one of the few works that aim to use multiple PUFs to achieve increased security. They use the term noisy to characterize the inherent variation in PUF responses. This protocol aims to reduce the vulnerability against modeling attacks by modifying the challenge to the main PUF with the aggregated response X from a non-noisy (error corrected) PUF. They assume that the trusted party can create correct models for both PUFs by observing X, and therefore authenticate the device. They assume that the trusted party can then *permanently* disable access to X, such that an attacker cannot observe X and therefore cannot create models. This assumption does not hold under consideration of invasive attacks, and this is identified as a significant issue for PUF security in section 3.4.1.

3.3 System-level Security Model

In this section, we will discuss metrics and security requirements for embedded systems. Based on these requirements, we present mechanisms to achieve security with respect to particular vulnerabilities. We then introduce a practical system-level security model for embedded systems, which defines and classifies components, interactions, and vulnerabilities. With this model, we aim to breach the chasm between component-level formalized security models [31], [38], [49], [50] and their system-level implementations. This system-level security model is important to achieve improved consistency and resource effectiveness in systems and protocols; the current lack of a system-level security model has led to a disparity between systems and components that exposes the system to additional vulnerabilities, as shown in section 3.4.

3.3.1 Threats and Metrics

We discuss several system-level threats that will be considered in the proposed security model. For each of the threats, we define different degrees of security functionality with respect to the threat which is used to characterize specific components in subsection 3.3.3.

Definition 1 (Security Functionality): The security features that are provided by a product or component [51]; A component C has a set of security functionalities $F(C)$. The degree of a particular functionality $f \in F(C)$ is denoted by $F(C)_f$.

3.3.1.1 Invasive Attacks

The threat of invasive attacks, e.g. through circuit edit with de-packaging [28], [29], [52], was one of the driving forces for research in PUF. With an increasing number of attack techniques and defense mechanisms, a system-level characterization is important to enable security based system design. We differentiate between the following three degrees of resistance against invasive attacks, which we refer to as tampering.

Definition 2 (Tamper Evident): A component which is tamper evident (TE) shows physical signs of tampering that can be observed and utilized by a trusted party.

Definition 3 (Tamper-Volatile Functionality): Tamper-volatile functionality (TVR) components change their functional behavior upon invasive modification, which can be exploited to detect and respond to such attacks. An example of this class is PUF, which exhibits highly volatile behavior that changes upon tampering.

Definition 4 (Tamper Adverse): A component which is tamper adverse (TA) increases the difficulty of *performing* an invasive attack by posing an obstacle to the adversary. TA components do not take active measures of recognizing invasive attacks. An example of this class is passive metal mesh, which shields relevant parts of the datapath, and therefore increases the circuit-editing effort by an adversary who wants to monitor the datapath.

Definition 5 (Tamper Susceptible): A component that does not contain any measures to detect or oppose invasive attacks and does not exhibit uncontrollable behavioral changes upon invasive attacks is tamper susceptible (TS).

3.3.1.2 Modeling Attacks

Modeling attacks aim to unveil internal behavioral patterns of a device for prediction and exploitation. A model can be generated from a vast number of input and output observations using machine learning (ML) algorithms. They have been shown to be successful against PUF when both input and output are directly accessible.

Definition 6 (Modeling Resistant): A component is modeling resistant (MR) if it cannot be modeled, as it does not expose its behavior to the outside.

Definition 7 (Modeling Resistant through Infeasibility): When a component exhibits highly complex behavior such that the computational cost of generating a model exceeds the capabilities of an expected adversary, it is modeling resistant through infeasibility (MRI).

Definition 8 (Modeling Susceptible): Exposed input-output (IO) behavior and reasonable computational cost define a modeling susceptible (MS) component.

3.3.1.3 Side-Channel Attacks

Side-channel attacks exploit side-products of confidential computation, memory-access, or other functionality, to indirectly infer secret information. This attack is of particular use when the component that operates on the secret information is not directly accessible. Examples of side-channel attacks include power analysis [26], thermal imaging [53], and photon emission analysis [54]. They have been shown to be particularly useful to extract the secret key from cryptographic processors running secure encryption.

3.3.1.4 Attack Multiplicity

A significant threat to conventional hardware systems is a *break one, break all* (BOBA) principle, in which the successful attack against a single system is applicable to many or all similar systems without major modification. An example of high attack multiplicity is a system that contains a hardware encoded secret key, such that all manufactured systems contain identical secret keys. When one of these systems is attacked and the secret key is extracted, all other systems are exposed as well. A PUF based implementation is an example of providing low attack multiplicity – even when an attacker extracts the challenge-response behavior of one PUF instance, it does not expose the behavior of other instances.

3.3.1.5 Malicious Design and Untrusted Foundry

Both design and manufacturing processes are increasingly collaborative but distributed, and therefore pose multiple threats: (i) modified RTL behavior or additional circuitry can lead to hardware Trojans horses (HTH), which can expose confidential data or disrupt other services; (ii) electronic counterfeiting, overproduction, and IP theft due to outsourced foundry.

3.3.2 Classification

Existing formalized security models as introduced in subsection 3.2.2 are primarily concerned with threat models, attack vectors, and vulnerabilities. Particularly for system-level considerations and for applicability in embedded systems, a security model with component-wise functional characteristics is insufficient. Due to the limited pool of

resources in embedded systems, implementations have to carefully optimize security metrics while minimizing the resource overhead. By enabling characterization and classification of hardware security components, our system-level model allows resource-driven security with intuitive component selection.

Definition 9 (Core Component): A core component C_C is one that provides the foundation of the security functionality; if a requirement r in $R(C_C)$ of system S is not met by a functionality f in $F(C_C)$, the requirements $R(C_C)$ are infringed and therefore the security requirements of the entire system S are not satisfied:

$$\mathrm{f} \neq \mathrm{r} \rightarrow \neg R(S) \text{ for } \exists r \in \mathrm{R}(\mathrm{C_C}), f \in F(C_C)$$

Definition 10 (Periphery): Components which do not directly interact with, affect, or enable security functionality are defined as security periphery. It is in the nature of computing systems to contain such periphery, and this category enables classification of all elements in this security model, even when they are not concerned with security itself.

Definition 11 (Service Component): Unlike modifier components, a service component C_S contributes to usability, quality, reliability, or other security-unrelated functionality of a component C. Component C can be a core component, modifier component, or other service component. When C_S is applied to C, the security functionality of the resulting system is limited by the functionality $F(C_S)$:

$$F(C)_f = \min\big(F(C)_f, F(C_S)_f\big) \text{ for } \forall f \in F(C_S)$$

Definition 12 (Modifier Component): A modifier component C_M applies to a compatible generalized component C which can be a core component, periphery, service component, or other modifier component. The purpose of a modifier component is to change security characteristics. When C_M and C are compatible, $[C_M, C]$ evaluates to true. Similarly, when C_M is applied to C and modifies it, $mod(C_M, C)$ evaluates to true. In this model, the modifier component directly transfers all of its security characteristics to C:

$$[C_M, C] \wedge mod(C_M, C) \rightarrow F(C)_f = F(C_M)_f \text{ for } \forall f \in F(C_M)$$

3.3.3 Security Components

According to the proposed model, the classification of components depends not only on the design, but also on the purpose of the component. Herein, we will demonstrate our classification on a subset of state-of-the-art hardware defense techniques. It will be applied

in section 3.5 to provide a resource optimized security solution for unilateral embedded system authentication.

Core components are those that provide critical security functionality. This classification includes the PUF, which has applications that include secret key generation, unclonability, and Trojan detection. Security features of a PUF are: Tamper volatile functionality, as the physically volatile responses change upon invasive tampering; physical unclonability, as the security functionality is based on actual physical device parameters; low attack multiplicity, due to the device-specific security functionality, successful attack (modeling, side-channel) of a device only breaches the device itself, and not other instances; other core components can be random number generators (RNGs), which provide design and implementation dependent security functionality. Additional potential core components can be dedicated security registers [55], secure RAM implementations, such as oblivious RAM [56], and security processors [57].

Most embedded systems are not primarily concerned with security; therefore most components on such systems are categorized as periphery. This includes all components that do not affect security functionality or components in any way. Examples are arithmetic and signal processing units, (conventional) test circuitry, and memory in so far as it does not contain confidential data.

Service components are those that positively affect usability. This classification applies to many components in PUF protocols, as PUF itself is a fundamental building block that in itself provides limited security functionality. Service components with the purpose of increasing PUF reliability are error correcting codes that employ helper data. As a downside, this component introduces additional resource utilization and information leakage, mandating a larger or more complex PUF implementation. Another, much simpler, service component for PUF is the input/output (PUF-IO) circuitry. Although PUF-IO is of low complexity, bundling it into the PUF itself would not accord with our security model and can generally provide adverse security implications. For protocols that directly expose the challenge and response of the PUF, PUF-IO introduces significant susceptibility to modeling attacks [58].

The purpose of modifier components is to transform the security functionality of other components, and they are applicable at the entire design stack, ranging from the signal routing to sensors. Examples of this classification are: signal routing at low metal layers to shield against semi-invasive and invasive hardware attacks [28]; passive metal meshes to

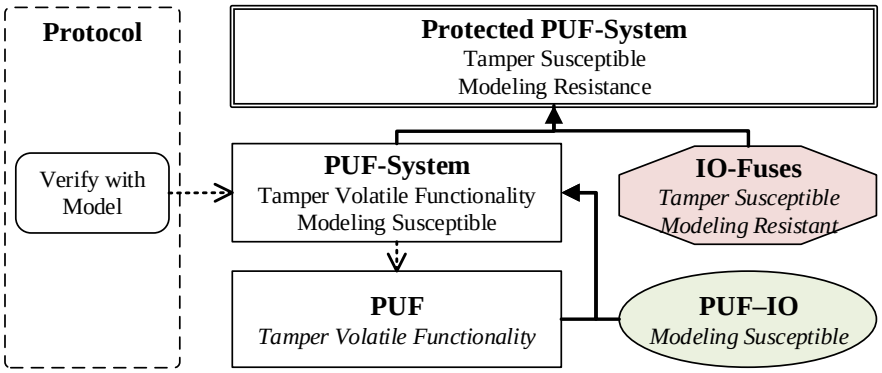

Figure 3.2 Modeling susceptibility (MS) due to PUF-IO is removed using IO-Fuses, which are tamper susceptible (TS) and diminish the tamper volatile functionality (TVF) of PUF.

increase the difficulty of semi-invasive attacks and active metal meshes that carry a signal to prevent invasive attacks [28]; fuses that can be burnt to disable access to certain components after read / write access by the trusted party [30], [59], [60] and that, in the case of PUFs, are often applied to remove the vulnerability created by directly accessible PUF-IO; circuitry that limits the interaction with a component or the number of protocol executions that involve the aforementioned component.

3.4 PUF Security Issues

In this section, we apply the security model introduced in section 3.3 to discuss the security of current protocols and expose vulnerabilities. The issues that we describe are impractical or impossible assumptions in subsection 3.4.1, inconsistent evaluation of tampering in subsection 3.4.2 and storage complexity in subsection 3.4.3. We propose a protocol to overcome these issues and specifically discuss the concerns mentioned here in section 3.6.

3.4.1 Known Model Assumption

Protocols for authentication based on PUFs typically require either a known model of the PUF [30], which is examined in this subsection, or a large set of CRPs [7], discussed in subsection 3.4.3. Protocols such as Slender PUF Protocol assume that the trusted party can compute a true model for the PUF challenge-response behavior, but an adversary cannot. This is justified by suggesting that the PUF circuitry initially exposes the challenges and responses of a PUF, such that machine learning algorithms can be used to learn a model. After model generation, the trusted party will externally disable the direct access and thus

hinder the malicious party from creating a model, which is done using a fuse to disable access, as described in subsection 3.4.1. As model-building attacks might be possible when the full PUF response is accessible to the outside [30], [58], the ability to disable the sensing connection is key to this assumption.

Using our embedded system security model introduced in section 3.3, we will show that this argument is difficult to uphold and has negative consequences for the security of the system. The analysis of this application is represented in Figure 3.2, in which rectangles represent core components, octagons depict modifier components, and ellipses represent service components. The security functionality introduced by a component is depicted using cursive typeface; functionality that is inherited by a system is in standard typeface. Independent of the particular implementation, PUF is considered a core component, and provides inherent tamper resistance due to its volatility. Additionally, PUFs by themselves are susceptible to side-channel attacks [61]. As a service component to directly interact with the PUF, which enables the trusted party to build a model, PUF-IO is inserted. The solid arrows in Figure 3.2 show how the components constitute a larger component. When PUF-IO directly exposes the challenges and responses of a PUF, the PUF behavior can be modeled [58]. Thus, the resulting system is susceptible to modeling attacks. The conventional response to this is to use fuses that disable the direct PUF access, which can be modeled as a modifier component. While this initially defends against modeling attacks as intended, it also introduces a new vulnerability to the PUF system, namely the tamper susceptibility. This is the case, as invasive circuit editing [28], [29] can render the fuses useless and re-enable the direct PUF access. Thereby, tamper resistance as one of the main advantages of PUF is lost. At this point, additional costly techniques that have limited applicability for embedded systems, such as active metal meshes, can be employed to reduce the vulnerability against invasive attacks. However, tamper resistance cannot be restored to the point of inherent volatility that leads to immediate changes upon modification, as is the case for PUF.

This reasoning was based on the assumption of physical fuses, which are burnt to remove access. However, it can be applied to similar other concepts that aim to change physical access to suit the needs of the trusted party. For example, logical disabling of the sensing capability cannot be considered to be secure, as an adversary could acquire the relevant knowledge to re-enable it. Maliciously forcing glitches, e.g. by altering the clock frequency, is also a common approach to overcome logical access restrictions. We identify this as a

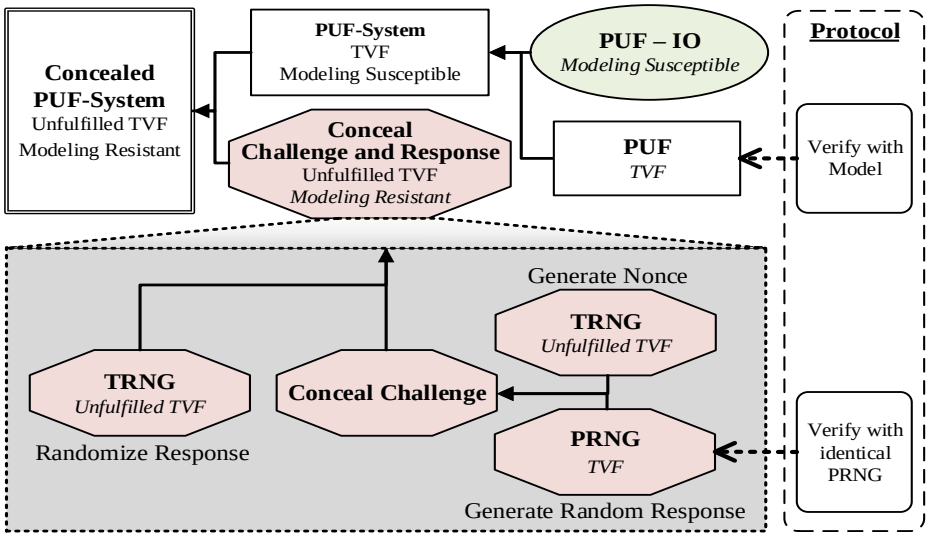

Figure 3.3 System-level security model of slender PUF protocol. Tamper volatile functionality (TVF) of TRNGs is unfulfilled, which introduces susceptibility against invasive attacks.

major deficiency in existing PUF-based security protocols and propose to treat the trusted and malicious party equally. Thus, we require that modeling capabilities of the trusted party lead to the same for the adversary.

3.4.2 Achieving Tamper Resistance

In this subsection, we emphasize the importance of carefully analyzing, characterizing, and utilizing security features with regard to hardware invasive attacks. As outlined in subsection 3.3.1, functional tamper resistance as provided by a PUF has to be utilized by the protocol or usage scenario to achieve consistent resistance against invasive attacks.

Components such as a *true random number generator* (TRNG) or a *pseudo random number generator* (PRNG) have to be treated with particular care and cannot be assumed to work as designed under invasive attacks. A security requirement of correct functionality is difficult to achieve for these components and requires cost-intensive tamper resistant techniques. Whereas a PRNG can be verified to work correctly by simulating or manufacturing a PRNG with same design and seed, this is not possible for a TRNG: by definition, the TRNG exhibits random and unpredictable behavior. Even an implementation based on PUF, which inherently provides TVF, is not tamper resistant, as TVF needs to be fulfilled (verified) by the protocol. The issue is that a PUF-based TRNG will exhibit changed behavior upon invasive attacks, but this change is not directly noticeable. Potential solutions to this are to: (i) model the distribution of the TRNG at enrollment and verify this

at runtime or protocol execution; (ii) periodically verify the randomness of responses at runtime. Thus, using TRNGs when invasive attacks are foreseen requires a more complex protocol with increased resource utilization. This fallacy is demonstrated on the basis of slender PUF protocol [30] in Figure 3.3. Here, a TRNG is employed to generate a random nonce on the PUF system, and another TRNG is applied to select a random substring of the PUF response. The slender PUF protocol applies these as modifier components, meaning that they change the security behavior of other components, namely concealing the input and output to the PUF to thwart modeling attacks. The issue is that the security functionality of these modifier components has to be carefully analyzed in conjunction with the protocol. When typical PUFs are invasively attacked, their challenge-response behavior changes and the protocol that utilizes the PUF, e.g. for authentication or key generation, is not functional anymore, thus enabling tamper resistance. In this protocol, the TRNGs are not core components but merely modifier components; even when implemented with PUF, they can be invasively attacked using e.g. circuit editing to change their behavior. This is possible, as the protocol does not take steps to ensure that these components provide correct functionality. Therefore, both of the TRNGs are susceptible to invasive attacks. As the authors themselves note, an attacker that is able to control both the *Prover* and *Verifier* circuitry would be able to generate a model, which is the case using invasive attacks as we have outlined here. Although this can be mitigated using additional, costly active tamper detecting circuits, this would not be in line with the motivation of slender PUF protocol.

3.4.3 Exponential Storage Need

When protocols do not rely on the *known model assumption* explained previously, they typically operate on a set of initially gathered challenges and responses as part of the PUF enrollment. As there is a finite number of gathered challenges, such protocols require that each gathered challenge is issued only once, as in [7]. The purpose of this is to thwart replay attacks in a scenario where the attacker has eavesdropped into legitimate authentication and thereby knows all valid responses to previously posed challenges. This approach provides an effective hindrance against this form of protocol, but it also exposes the protocol to denial-of-service (DoS) attacks: A malicious third party can query the *Verifier* until the stored CRP set is exhausted. This could be mitigated by storing a huge amount of CRPs at the *Verifier* side, which ultimately causes a data storage problem. To withstand a DoS attack, the *Verifier* would be required to store large numbers of long CRP strings and additional

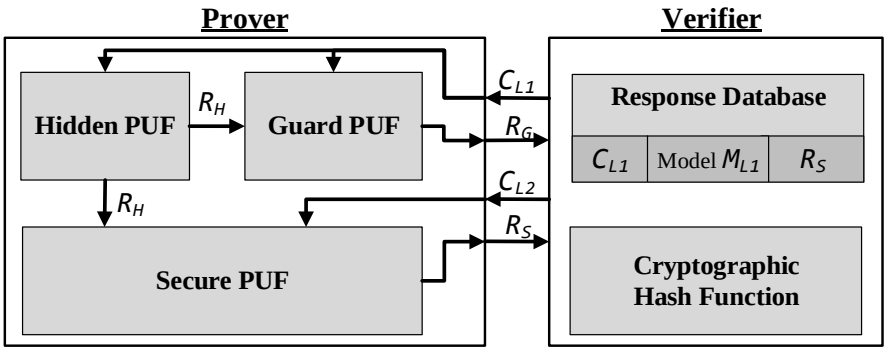

Figure 3.4 Components of the lightweight System of PUF.

synchronization bits. Whereas the *known model assumption* poses a security vulnerability that can lead to a compromised system, using a (limited) number of stored challenge-response pairs only exposes the protocol to DoS attacks that reduce usability and applicability. In section 3.5, a system is introduced that does not assume a one-sided model, but is able to avoid the problems outlined here while operating in multiple levels.

3.5 System of PUFs

A major limitation that can be observed in current protocols is the usage of only one type of PUF design [7], [30]. However, there exist a wide range of competing PUF designs that provide complementary features, particularly with regard to reliability and resource overhead, as outlined in subsection 3.2.1. Only very few authentication protocols employ a combination of multiple PUFs [47], [48], [62], and these protocols do not take advantage of the trade-off enabled by the large PUF design-space. Whereas multiple research works compare and contrast existing PUF designs, we are not aware of any work that attempts to combine different designs to take advantage of unique characteristics.

To deal with the various issues discussed in section 3.3, we propose a novel SoP that utilizes the design-space offered by continuous PUF research and uses multiple levels of challenge-response interrogation for authentication.

3.5.1 Security Requirements

Before introducing System-of-PUF (SoP) and the multilevel authentication protocol, we state the following security requirements that were considered in the design of SoP and the protocol. These requirements are typical for light-weight embedded system applications. To

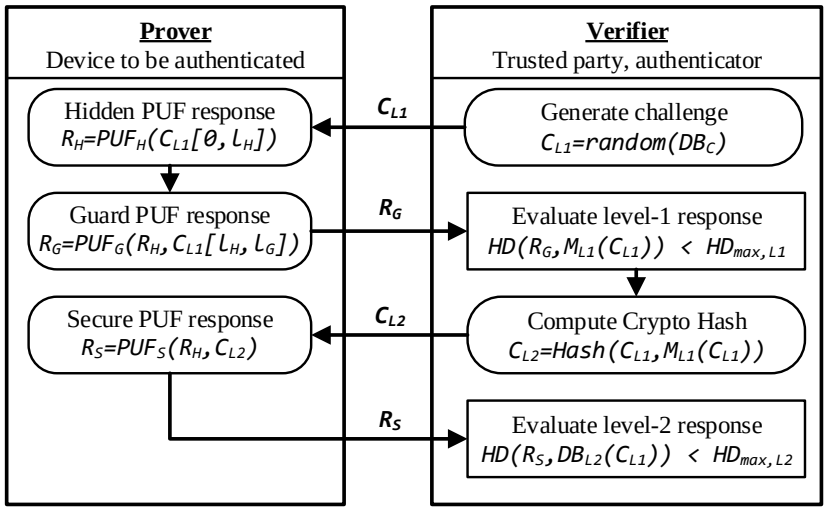

Figure 3.5 Proposed multilevel authentication protocol between the *Prover* and the *Verifier*.

achieve further requirements such as resistance against side-channel attacks, the protocol can be expanded with existing techniques [53].

Requirement 1: Unilateral authentication between a trusted party (*Verifier*) and a hardware device (*Prover*). This is a typical scenario for embedded system applications, such as smart-cards [63], sensor networks [64], and RFIDs [65].

Requirement 2: Resistive against invasive attacks, such as de-packaging and circuit-edits. Invasive attacks should render the *Prover* device inoperative and at the least change its behavior such that it cannot successfully participate in the authentication protocol.

Requirement 3: Resistive against modeling attacks; neither adversary nor trusted party should be unable to generate a model for the internal behavior of the *Prover*, particularly PUFs.

3.5.2 Multilevel Authentication

To provide secure unilateral authentication, the proposed protocol operates in multiple levels with a system that consists of three different PUFs, as shown in Figure 3.4: (i) Hidden PUF is a reliable PUF with challenge-length l_H that does not expose its response R_H to the outside and thus is *hidden*. It limits the exposure of Guard PUF and Secure PUF inputs to the outside, and thereby increases the difficulty of modeling attacks of both level-1 and level-2 responses. (ii) Together with the Hidden PUF, Guard PUF provides the first level of authentication with a challenge-length of l_G. The response of the Guard PUF R_G is exposed, but the input is not. Although of reasonable complexity, we assume this PUF to be modeled

by the trusted party and thus also any attacking party. It not only acts as a guarding stage before the Secure PUF, but also indirectly propagates errors of the Hidden PUF to the *Verifier* and thus reduces the critical level-2 false-negative rate. (iii) Secure PUF is the secure backbone of SoP and is impossible to model within reasonable time and computational complexity, which is denoted as modeling resistivity through infeasibility (MRI) as defined in subsection 3.3.1.2. This PUF has a challenge-length of l_S, and both the challenge- and response-space should be large such that MRI is achieved. The specifics of the authentication protocol are depicted in Figure 3.5. Initially, the *Verifier* chooses a challenge C_{L1} of bit-length l_G, equal to the challenge-length of the Guard PUF. This challenge is chosen randomly and can thus fulfill the role of a nonce to thwart replay attacks. As this challenge is randomly chosen, it is very unlikely that the same challenge will be issued twice, thereby eliminating replay attacks. Then, the challenge is issued to the *Prover*. Here, the Hidden PUF will generate the response R_H, which is internally connected to the Guard PUF and Secure PUF inputs. From R_H and C_{L1}, the level-1 response R_G is generated. The output of the Hidden PUF is directly connected to the Guard PUF and the Secure PUF; it is not available to the outside. After accepting the level-1 response through a Hamming distance check against the model M_{L1}, the *Verifier* will send the level-2 challenge C_{l2}. It is generated from a secure hash function from the initial challenge and R_G, and has a length of $l_{Hash} = l_S - l_H$. The *Prover* then generates the level-2 response R_S from the initial Hidden PUF response R_h and C_{L2}. Finally, the *Verifier* will verify the level-2 response R_S by calculating the Hamming distance against the true response stored in the database $DB(C_{L1})$.

At first glance, employing the Guard PUF and Hidden PUF as an outer layer to the Secure PUF can seem unnecessary, as the Secure PUF itself is already assumed to be of significant complexity to thwart modeling attacks. Additionally, it may seem counter-productive to secure the Guard PUF through the Hidden PUF when the trusted party aims to generate a level-1 model. However, there are three reasons why this approach is taken in SoP: (i) increasing the difficulty of generating a level-1 model requires prolonged interrogation by an adversary, thereby eliminating many attack scenarios, e.g. ATM operation; (ii) as only serious and malicious attacks arrive at the second level, SoP supports breach recognition that cannot be triggered by accidental misuse or without prolonged interrogation of the *Prover*; (iii) although the Secure PUF by itself is resistant against modeling attacks, we increase this resistivity using the outer layer, and thereby either enable

Table 3.2 PUF design criteria and example implementation

PUF	Criteria	Examples
Low cost Hidden PUF	High reliability	RO PUF [4], Error corrected PUF [17]
High cost Hidden PUF	High reliability Reconfigurable	Recyclable PUF [42]
Guard PUF	Low cost	Arbiter PUF [4]
Secure PUF	High security	XOR-Arbiter PUF [7], lightweight-PUF [20], interleaved Arbiter PUF [14]

long-term security (in presence of increasing compute capability) or the usage of a smaller PUF to achieve the same degree of security.

A database of (previously gathered) responses is employed, as we do not make the *known model assumption* as discussed in subsection 3.4.1. However, SoP is also not vulnerable to the *exponential storage* issue outlined in section 3.4.3, as our multilevel approach allows storing very few of the lengthy level-2 responses without falling susceptible to DoS attacks.

3.5.3 Breach Recognition and Recovery

As explained in subsection 3.5.2, the Guard PUF may be modeled by an adversary. Note that although this is possible, it will come with a considerable computational cost and high runtime, as the PUF is not fully exposed to the outside. We define a security breach as a situation in which an adversary has gathered significant information on the challenge-response behavior. In this multilevel scheme, a security breach can be recognized: When a false *Prover* P' repeatedly replies with the correct level-1 response $R'_G = P'(C_{l1})$ but with incorrect level-2 responses, the protocol can directly infer that the adversary has either generated a model, or gathered a large set of challenge-responses. If an adversary purposefully responds to a level-1 challenge with an incorrect response, he will not receive a correct level-2 challenge and therefore will be unable to infer the actual level-2 behavior. Due to the multilevel nature of SoP, it is still possible to recognize the true PUF: Until all stored level-2 challenges are exhausted, the true *Prover* may successfully authenticate, at which point the Breach recovery may be initiated.

One of the major advantages of the SoP is that it may be used with different PUF designs. For higher cost applications with a larger resource budget, the Guard PUF can be designed to be reconfigurable. This can effectively reset the level-1 PUF behavior. As the correct

Prover can be authenticated even after a level-1 breach, the advances of an adversary can thus be diminished.

3.5.4 Design-space Utilization

In combination with the wide range of existing PUF designs, our proposed SoP and multilevel authentication protocol enable a variety of design choices adequate for different application scenarios. In ultra-low cost applications such as RFID and wireless-sensor networks, secure authentication is an important requirement, but the on-chip resources are minimal. For such applications, a minimum implementation of SoP that does not implement breach recovery and thus only requires simple PUFs is possible. Instead of parallelizing PUFs, output bits can be created with challenge expanders through LFSRs.

In addition to the degree of freedom previously described, each of the three PUFs has to be chosen with great care to their specific criteria, as shown in Table 3.2. The high reliability of the Hidden PUF is one of the main requirements, as any bit-error in R_H will automatically falsify the Guard PUF and Secure PUF responses due to the avalanche criterion. The high resilience to modeling attacks of the Secure PUF is also a criterion that is fundamental to the security of the proposed authentication scheme. This clear distinction simplifies the design task and reduces the cost, as highly secure designs typically are unreliable and have to be correct with highly complex fuzzy extractors as described in subsection 3.2.3.

3.6 Security Considerations

In this section, we discuss the security of SoP under the proposed model and with regard to the previously identified prevalent PUF security issues. As described in section 3.5, the first level of SoP may be modeled by an adversary. Accordingly, we will show that security of the authentication protocol is not contingent upon secrecy of the first-level challenge-response model.

3.6.1 System-level Security Analysis

Our analysis of SoP and the multilevel authentication protocol with regard to the system-level security model we introduced in section 3.3 is shown in Figure 3.6. The most pronounced difference between our protocol and prior work is our approach to handling Requirement 3, resistivity against modeling attacks. The two techniques to thwart modeling attacks in SoP are: (i) conceal PUF I/O by separating input to the Prover from the output,

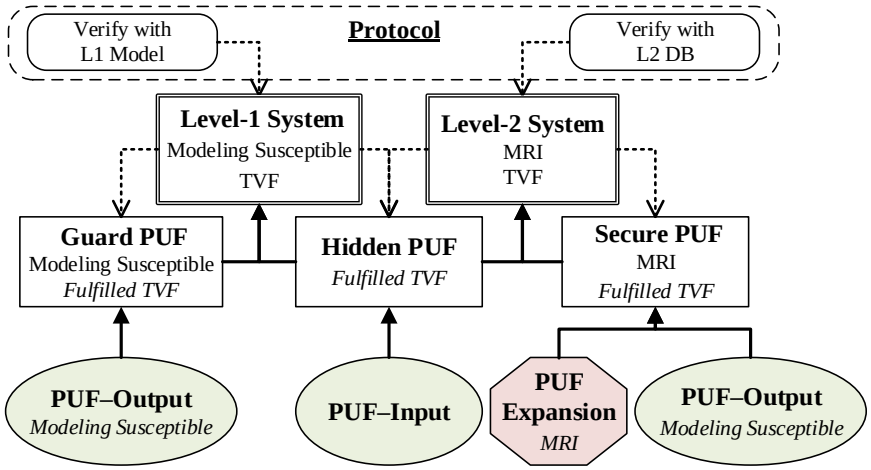

Figure 3.6 System-level security model of SoP. Due to the size of the Secure PUF, the level-2 behavior is modeling resistant through infeasibility (MRI). The system exhibits tamper-volatile functionality (TVF).

such that none of the PUFs expose both their input and their output; (ii) employ a secure PUF with large CRP-space to achieve modeling resistivity through infeasibility (MRI), as previous research has shown that PUF size can be increased to increase the cost of modeling to the point of infeasibility.

Resistivity against invasive attacks as per *Requirement 2* is achieved through exclusive use of PUF. Note that the protocol is designed such that the inherent TVF of PUF is fulfilled by comparing the level-1 and level-2 responses with respective database entries. Tampering of the Hidden PUF changes the response R_H and therefore the input challenges to both the Guard and Secure PUF, thereby changing the level-1 and level-2 responses. Tampering with the Guard PUF or the Secure PUF changes their behavior, which changes the level-1 or level-2 responses, respectively.

3.6.2 Attack Scenarios

In the following, we will discuss several possible scenarios, which include those that would lead to successful masquerading of a malicious party. We will show that in practice, our proposed protocol is not vulnerable against replay attacks and is resilient against denial-of-service attacks that would disable other protocols with limited stored CRP sets. We first consider a random guessing attack, and then the attacks of an eavesdropping attacker.

3.6.2.1 Random Guessing Attack

The attacker has to correctly guess the level-1 response with l_G bits and the level-2 response with l_S bits. False-positive authentication is the case when an attacker successfully guesses both of these responses correctly. The probability for a randomly guessed false positive authentication P_{RGFP} is:

$$P_{RGFP} = \left(\sum_{i=0}^{HD_{max,l1}-1} \binom{l_G}{i} 0.5^{l_G} \right) \left(\sum_{i=0}^{HD_{max,l2}-1} \binom{l_S}{i} 0.5^{l_S} \right)$$

For the lightweight system evaluated in section 3.7, this probability evaluates to $P_{RGFP} = 1.8 * 10^{-8}$

3.6.2.2 Strong Knowledge Attack

We consider the attack-scenario that the malicious party was in physical possession of the true *Prover*. Although difficult, it is possible that the malicious party generated a model for the level-1 behavior (from Hidden PUF input to Guard PUF output). When a malicious *Prover* tries to authenticate itself to a trusted *Verifier*, it will therefore correctly respond to the initial *level-1 challenge* C_{l1}, and will receive the corresponding *level-2 challenge* C_{l2}. However, it is numerically impossible that the attacker was able to generate a valid model for the level-2 behavior due to modeling resistivity through infeasibility (MRI) of the Secure PUF, as outlined in subsection 3.5.2. Nonetheless, it is possible that the attacker gathered a large amount of CRPs including C_{l2} and *level-2 responses* R_S by eavesdropping into valid protocol executions. Consider the attacker to have obtained k_{att} CRPs. For each CRP, he has to store C_{l1} of length l_G, C_{l2}, and R_S which both have length $l_S - l_H$, for each CRP. Considering only this data, this requires the attacker to store $D_{attacker} = (l_G + 2l_S - l_H)\frac{bit}{CRP}$. The trusted party stores only C_{l1} and R_S for each CRP, which results in storage requirement of only $D_{trusted} = (l_G + l_S)\frac{bit}{CRP}$. As $l_G \ll l_S - l_H$, the trusted party uses only half as much memory as the malicious party. Furthermore, the malicious party has to store an exponential number of CRPs to achieve realistic authentication probabilities. The probability for a successful knowledge attack P_{KA} is:

$$P_{KA} = \frac{k_{att}}{2^{l_S}} + \left(1 - \frac{k_{att}}{2^{l_S}}\right) P_{RGFP}$$

In our scenario, this means that an attacker with a model for the level-1 behavior will require $k_{att} = 2 * 10^{17}$ stored level-2 responses to achieve a false-positive rate of 1%. This

will require storage of $D_{attacker} = 2 * 10^{17} * 144b \approx 3.4 * 10^{9} GB$. Although the attacker and the trusted party have only a factor of 2 difference for a single CRP, the trusted party is not required to know a large subset of the challenge-space, as he chooses the challenge.

This demonstrates the efficiency of our multilevel authentication and shows that SoP acts as a force-multiplier that supports the trusted party in authentication by drastically reducing off-chip memory requirements and on-chip resources.

3.7 Experimental Evaluation

3.7.1 Overview

In the experimental evaluation, we simulated the lightweight SoP as described in section 3.5. Our simulation environment was a C++ implementation of a synthetic PUF similar to that employed in previous research [58]. Our implementation considered the differences in reliability and randomness between different designs due to process variations and environmental variations. The specific configuration of our evaluated SoP consists of synthetic implementations for three selected PUF designs according to the criteria in Table 3.2 and the characteristics in Table 3.1:

- Hidden PUF: 16-Bit input RO-PUF
- Guard PUF: 32-Bit Arbiter PUF
- Secure PUF: 64-Bit 4-XOR Arbiter PUF

3.7.2 Gate-level Cost Comparison

To provide an estimate of the area overhead incurred by authentication protocols based on PUF, we performed a consistent gate-level evaluation. For this comparison, we evaluate the main security components, as the control logic introduces negligible overhead and should be comparable for all protocols.

Table 3.3 Gate equivalent (GE) cost of proposed SoP.

Component	Explanation	GE units
Hidden PUF	16-Bit RO-PUF	145
Guard PUF	32-Bit Arbiter PUF	130
Secure PUF	64-Bit 4-XOR PUF	1032
Challenge Expanders	16-Bit + 32-Bit + 64-Bit LFSRs	460
Total		1767

Table 3.4 Gate equivalent (GE) cost of Reverse Fuzzy Extractor.

Component	Explanation	GE units
PUF	64-bit 4-XOR PUF	1032
Challenge Expander	255-Bit LFSR	1024
Syndrome Generator	234-Bit LFSR	940
SPONGE Hash	256-Bit light-weight Hash	1950
Total		4946

In Table 3.3, the cost of our proposed lightweight SoP is given as 1767 gate equivalent units (GEs). Due to the entropy loss inflicted by the Syndrome generator, the Reverse Fuzzy Extractor requires longer responses and additionally employs a Hash function, leading to a total cost of 4946 GEs as shown in Table 3.4. Therefore, our lightweight SoP reduces the gate count by 64%.

This emphasizes the low-cost characteristic of SoP. The proposed protocol implements unilateral authentication as part of the requirements introduced in subsection 3.5.1. Reverse fuzzy extractor additionally provides mutual authentication. We note that unilateral authentication is sufficient and often required for many light-weight embedded systems, such as NFC, RFID, and sensor networks.

3.7.3 Reliability despite Error Propagation

As explained in Section 3.5, we chose the Hidden PUF as an implementation of a RO-PUF, as it is the most reliable design available from the comparison in Table 3.1. The Hamming distances of each PUF component are shown in Figure 3.7, and the propagated

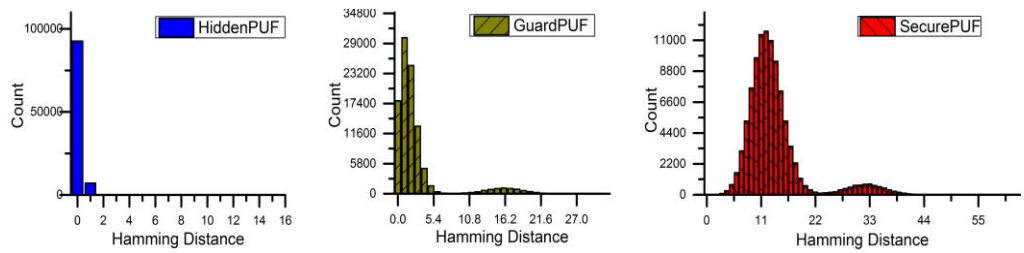

Figure 3.7 Intra-Chip Hamming distances of each of PUF component. We observe that an error in the Hidden PUF (left) will propagate and lead to a large error in the Guard PUF (center) and the Secure PUF (right) due to the strict avalanche criterion.

error from the Hidden PUF can be seen in the Guard PUF and Secure PUF around $HD_G = 16$ and $HD_S = 32$, respectively. This shows that even a minor error in the Hidden PUF leads to a large error with $FHD_{intra} = 0.5$ for Guard PUF and Secure PUF. The reason for this is the strict avalanche criterion, which requires that even a single bit-flip on the input should, on average, lead to a bit-flip for half of the output bits. Figure 3.7 also shows why we selected $HD_{max,l1} = 5$ and $HD_{max,l2} = 21$ as parameters for the protocol: The real responses of the Guard PUF have a Hamming distance of 5 or less to the ideal response. Similarly, the correct responses of the Secure PUF have a Hamming distance of 21 or less.

3.7.4 Authentication Error

For authentication, the false-positive and false-negative rates are an important quality metric, as they represent the amount of authentication attempts that were falsely accepted or rejected, respectively. In our experiments, $P_{FN,l1} = 7.8\%$ of the level-1 responses had an error that exceeded the tolerance of $HD_{max,l1}$ and were thus falsely rejected. The cause of this lies in the strict avalanche criterion, and the series connection between the Hidden PUF and the Guard PUF. Thus, the protocol behaves as intended and rejects bit-errors in the Hidden PUF already at the first level. With an adequately chosen tolerance $HD_{max,l2}$ at the second level and the Hidden PUF errors already filtered at the first level, only $P_{FN,l2} = 0.257\%$ of the level-2 responses were incorrectly rejected.

3.8 Conclusion

This chapter contributes a new system-level security model that bridges the chasm between application-level security analysis and design of secure hardware, and models for isolated components. From this model, we analyzed and explained several hardware security requirements using existing protocols, and showed that they cannot be fulfilled without extensive cost. We presented a multilevel authentication protocol which is verified using the system-level security model and which takes advantage of a combination of different PUF-designs to minimize resource allocation. SoP does not require expensive error-correction, as high reliability designs are employed where required. Furthermore, the need for latency and power intensive hash functions on the PUF circuit is replaced by a combination of strong PUFs and *off-chip* cryptographic hash. With breach recognition and recovery, new security features are introduced and shown to increase the attack-difficulty while enhancing reusability. A low-cost implementation of SoP was shown to reduce the area by 64% in a gate-level comparison. This low resource allocation and high flexibility allows SoP to provide a security solution tailored for ubiquitous computing devices.

CHAPTER 4
POLYPUF: PHYSICALLY SECURE SELF-DIVERGENCE

4.1 Introduction

Hardware security is increasingly recognized as an important research area for current and future devices. Security features are required for all modern communication and computing devices, particularly for verification of authenticity and data confidentiality. Diverse hardware-based threats such as hardware Trojan horses (HTHs), reverse engineering, physical de-packaging and modification, machine-learning, and side-channel attacks not only lead to billion dollar losses in counterfeits [66], but also challenge the capabilities of existing security techniques.

Hardware security is of particular value for emerging mobile applications such as wireless sensor networks, RFID chips, and smart cards. For these devices, conventional security techniques exceed power and footprint limitations. For instance, even widely used public-key cryptography techniques have high computational cost that can exceed the capabilities of such devices or can strain mobility by depleting their battery. Moreover, conventional defenses against physical attacks such as metal-meshes or tamper sensing through signal carrying wires require constant power supply and are therefore infeasible when low cost is a requirement.

PUFs are promising security primitives as they are based on intrinsic nano-scale manufacturing variations, are lightweight, and provide resistivity against physical attacks. It is a binary mapping that represents the unique IC fingerprint by accumulating and reflecting the manufacturing process variations that went into each specific device. The input and output of this function are referred to as challenge and response respectively. This function is a $\{0,1\}^m \rightarrow \{0,1\}^n$ mapping with m challenge bits and n response bits, which we refer to as an $m \times n$ PUF in this chapter. A wide range of variation sources are used to generate this fingerprint, for example the unique constellation of carbon nanotubes [12] in carbon nanotube transistors. Due to the utilization of manufacturing variations, the exhibited challenge-response behavior is device unique and is not physically reproducible by

remanufacturing. Furthermore, the secret of the PUF is the internal physical structure and therefore adversaries cannot easily extract it. This dependence of the PUF behavior on the exact physical parameters provides the PUF with a volatility which implies destruction of the secret on invasive physical attacks.

Although the PUF behavior is primarily determined by manufacturing variations, it is also influenced by environment variations including temperature, pressure, EM-waves, and quantum fluctuations [67] that can deteriorate PUF reliability. This lack of reliability manifests as noise in the challenge-response behavior and is characterized as the per-bit error rate in the responses when the same challenge is repeatedly issued. A common application is authentication, where a trusted party proves its authenticity by demonstrating ownership of the PUF.

The two primary concerns for the widespread viability of PUF-based security are reliability and resistivity against machine-learning. As previously stated, the volatility of a PUF is a benefit against invasive attacks, but it also introduces bit-errors as a disadvantage. The volatile nature leads to small changes in the PUF response due to environment variations. With reduced reliability, the obstacles in other areas increase; for instance, costly error-correction may be required, which in turn requires increased PUF sizes due to entropy loss. The issue of machine-learning resistance is important to guarantee that adversaries cannot create a model for the PUF. If an adversary successfully creates a model, it would fatally defeat any security application of the PUF; for instance, in token-based authentication, an adversary with an accurate model can impersonate PUF ownership and thus achieve false acceptance. Reliability and resistance against machine-learning are particularly difficult to achieve, as the techniques to implement them have contradictory effects. On the one hand, a common approach to increase the machine-learning resistance is to combine the responses of multiple PUFs. However, this also has a multiplying effect on the volatility induced bit-error rate and thus reduces reliability. On the other hand, reliability can be increased by implementing error-correction, for instance through repetition codes. However, this introduces information leakage and thus decreases the resistance against model-building attacks. An additional major disadvantage of the reliability-reducing Strong PUFs is that the increased error rate forces longer responses, which in turn require transmission of longer bit strings and can have a significant impact on energy consumption [68]. As reliability and simultaneous model-building resistance are the primary concerns of PUF, they are the focus of this chapter.

The problem with resistance against machine-learning is fundamentally due to two issues: complexity and determinism. A highly complex behavioral pattern is very difficult to learn through machine-learning, as more training data and computational resources are needed. The difficulty with increasing complexity in a PUF is that it typically has a detrimental effect on reliability. For machine-learning algorithms, deterministic behavior is the ideal training target, as the pattern to be learned can be accurately specified. As determinism is reduced, for instance due to noise, learning algorithms require more computational resources and training data to identify the underlying pattern. As such, reducing determinism is a viable approach to counter machine-learning techniques. For PUF, however, high determinism is required so that the response to any given challenge can be compared to a known correct response. In this chapter, we present the first PUF architecture with intentional non-determinism and allow the PUF to change randomly between multiple behavioral patterns.

The unique contributions made in this chapter are:

- The first PUF architecture to achieve unpredictable non-deterministic polymorphic challenge-response behavior.
- The first PUF architecture to demonstrate strong and scalable machine-learning resistance without detrimental effect on reliability and with wide applicability.
- A consistent security solution by evaluating and eliminating other threat vectors such as an unprotected random number generator.
- Quantitative evaluation of neural network attacks on our proposed PolyPUF architecture against reference architectures with various PUF configurations and training sets of up to one million examples.

This chapter is structured as follows. In section 4.2, we introduce relevant background and discuss related work. In section 4.3, we introduce PolyPUF and describe the polymorphic behavior it exhibits. In section 4.4, the practicality and possible applications of PolyPUF are discussed. Section 4.5 discusses various security concerns with existing work and clarifies the security advantages of PolyPUF. We provide an analysis of possible attacks in section 4.6. In section 4.7, we present an extensive evaluation of PolyPUF and existing techniques. We conclude in section 4.8. This chapter is based on [69]

4.2 Background

4.2.1 Notation

We denote binary vectors in bold lowercase characters. As such, we refer to a PUF challenge as $\boldsymbol{c}$ and to the response as $\boldsymbol{r}$. Multiple binary vectors are differentiated through use of a subscript, e.g. challenges $|\boldsymbol{c}_1|$ and $|\boldsymbol{c}_2|$. We denote random binary vectors as $\boldsymbol{x}$. The length of the vector is indicated through their absolute value, e.g. challenge length $|\boldsymbol{c}|$. The individual bits are referenced through round brackets, e.g. $\boldsymbol{c}(i)$ where $i \in [0, |\boldsymbol{c}|]$. Sets are denoted through uppercase letters, e.g. A. Due to their significance for this chapter, we denote bit-error rates with ϵ. The Hamming distance between two bit-vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is denoted by $\Delta_{\boldsymbol{a},\boldsymbol{b}}$. In larger equations where this subscript notation is not suitable, we also use $\Delta_{\boldsymbol{a},\boldsymbol{b}} = HD(\boldsymbol{a}, \boldsymbol{b})$ interchangeably.

4.2.2 Statistical PUF Behavior

In cryptography, confusion and diffusion are important properties of a security primitive. Confusion describes the complexity of the relation between the secret key and the cipher text, and diffusion describes the complexity between plain text and cipher text. In the context of evaluating a PUF, diffusion can be described as the complexity of the relation between the input (challenge) and output (response) of the PUF.

With low diffusion, knowledge of the response to one challenge $\boldsymbol{c}_i$ implicitly provides information on similar other challenges $\boldsymbol{c}_j$ where the distance $\Delta_{\boldsymbol{c}_i,\boldsymbol{c}_j}$ is smaller than a threshold d_{th}. It follows that machine-learning algorithms can extract significant information on the challenge-response behavior from few CRPs. Moreover, weak diffusion also enables a modified form of repetition attacks, where an adversary re-uses a previously observed response $\boldsymbol{r}_i = PUF(\boldsymbol{c}_i)$ for a new challenge $\boldsymbol{c}_j$ where $\boldsymbol{c}_i \neq \boldsymbol{c}_j$ and $\Delta_{\boldsymbol{c}_i,\boldsymbol{c}_j} < d_{th}$ with hopes that it will be accepted instead of $\boldsymbol{r}_j = PUF(\boldsymbol{c}_j)$. Here, $\Delta_{\boldsymbol{r}_i,\boldsymbol{r}_j}$ is small, as the originating challenges are close to each other. Due to bit-error mandated authentication thresholds, the response will be accepted when $\Delta_{\boldsymbol{r}_i,\boldsymbol{r}_j}$ is small. For the average response in a large set of CRPs, this generalizes to:

$$\Delta_{\boldsymbol{c}_i,\boldsymbol{c}_0} < \Delta_{\boldsymbol{c}_j,\boldsymbol{c}_0} \rightarrow \Delta_{\boldsymbol{r}_i,\boldsymbol{r}_0} < \Delta_{\boldsymbol{r}_j,\boldsymbol{r}_0} \tag{4.1}$$

In contrast, high diffusion implies internal complexity and provides high resistance against machine-learning attacks.

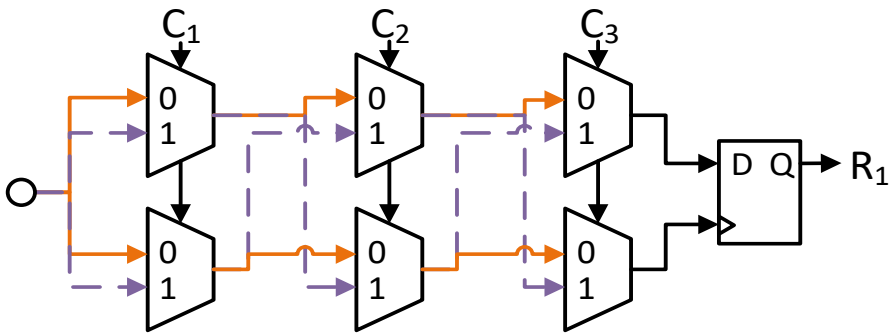

Figure 4.1. An Arbiter PUF with three input bits and one output bit, 3x1 Arbiter PUF.

The avalanche criterion is another desirable property requiring that a small change in the input changes the output significantly. The strict avalanche criterion requires that any input bit flip leads to a flip in each output bit with a probability of 50%, which implies strong randomization and therefore difficult input prediction.

4.2.3 Strong and Weak PUFs

PUFs are divided into two main categories, Weak PUFs and Strong PUFs. Weak PUFs allow a small number of challenges; in some cases only a single challenge can be issued. Their most common application is the generation of secret keys that can be used for cryptography. An example of this category is the SRAM PUF, which employs the start-up state of SRAM cells to generate a response. Strong PUFs must have a large CRP space such that it is unreasonable that an adversary can obtain a large share of all possible CRPs. Furthermore, its behavior must be unpredictable for an adversary and must provide tamper resistance [70]. Whereas the requirement of a large challenge response space is easily accomplished, unpredictability is an ongoing concern as discussed before.

Strong PUFs are subdivided into a variety of different designs that exploit physical variations through delay, frequency, temperature, and aging. The Arbiter PUF was one of the first proposed silicon-based PUFs and is shown in Figure 4.1. The challenge determines the signal path through a chain of multiplexers, and the response is set to logic 1 (0) when the signal through the upper path is faster (slower) than through the lower path. As the challenge determines the actual paths that the signals take, the response directly depends on the challenge and a large CRP-space is possible. The simplicity of the Arbiter PUF is one of its main weaknesses, and it is considered one of the weakest PUFs under model-building attacks as shown in section 4.2.5 and can easily be predicted [71].

To counter the predictability of the simple Arbiter PUF, the XOR Arbiter PUF was presented by Suh and Devadas [7]. It increases the internal complexity by combining the response of multiple Arbiter PUFs in an XOR operation. Despite its simplicity, this approach notably increases the difficulty of model-building attacks. However, two key factors limit the scalability of this approach: First, a k-XOR Arbiter PUF uses k different PUFs, and hence requires a linearly increasing number of PUFs. Second, the error rate ϵ of a simple Arbiter PUF increases to $\epsilon_{K-XOR} = 1 - (1 - \epsilon)^k$. Therefore, the error rate scales linearly for small K. In a 28 nm FPGA implementation, an error rate of up to 0.317 was demonstrated for a 4-XOR Arbiter PUF, drastically limiting its applicability [72].

A high error rate decreases the trusted party's ability to differentiate a true PUF from a counterfeit because the authentication protocol has to allow a threshold so that the probability of false rejection of an authentic response is low. Previous research has shown that k-XOR Arbiter PUFs are effective against model-building attacks when $k \geq 6$ [36], [71]. However, this number of Arbiter PUFs degrades the error rate; therefore, the effective defense against model building remains an unsolved problem.

4.2.4 Machine-Learning Techniques

4.2.4.1 Artificial Neural Networks

Artificial neural networks (ANNs) were initially designed after biological neural networks and are employed for tasks such as natural language processing and computer vision [73]. Each artificial neuron has inputs with corresponding weights and produces an output by applying a non-linear activation function to the sum of weighted inputs. The activation function affects the number of neurons that are needed for complex computations and the computational cost of simulating a neural network. Typically, a sigmoidal function is used as it can be normalized to produce stable outputs and is easily derivable which is useful for updating the input weights [73]. Supervised learning is the process of training the network with known training labels.

A feed-forward network of neurons consists of an input layer, a problem-specific number of hidden layers, and an output layer without cycles. The number of layers in the neural network specifies the depth. The number of neurons in the input layer and output layer are constrained by the problem, in the case of PUFs to the challenge and response lengths, respectively. The number of hidden layers and the number of neurons in each hidden layer

are derived heuristically. Increasing the number of neurons in the hidden layer allows modeling of higher complexity patterns, but can lead to overfitting and increases the computation time.

Backpropagation with gradient descent is a common learning algorithm for ANNs. For each weight $w_{i,j}$, the corresponding impact on the error function E is derived from the chain rule, where $w_{i,j}$ is the weight between neuron i and neuron j. With this derivative and a problem-dependent learning factor ϵ, the weight is updated from iteration t to t+1 to minimize the error function $w_{i,j}(t+1) = w_{i,j}(t) - \epsilon \frac{\delta E}{\delta w_{i,j}}(t)$.

The downside of backpropagation learning is that the weight update is dependent on the gradient, which has a small magnitude due to requirements on the activation function. Therefore, backpropagation can converge slowly. In resilient backpropagation (RPROP) [74] training, the change in weights does not directly depend on the gradient. Instead, the gradient only determines the direction of the weight update, and an individual update value $\Delta_{i,j}$ determines the magnitude of the weight update. This allows RPROP to converge much faster than backpropagation. In relevance to the evaluation in section 4.7, an epoch is a single pass through the entire training set including early evaluation with a verification set. Therefore, a limitation to epochs is a more sensible termination criterion than pure runtime for the context of learning under an ANN.

In this chapter, a feed-forward artificial neural network is trained with a hyperbolic tangent sigmoid activation function. This function achieves the desired stabilizing behavior of the sigmoid function but is one of the most common activation functions because it ranges from [-1,1] and therefore allows negative valued outputs.

4.2.4.2 Pattern Complexity and Model-Building Resistivity

The complexity of the pattern to be learned mandates the difficulty that a machine-learning algorithm faces in creating a model for it. A class of sets C is said to shatter a set A when the power set $P(A) = \{U \cap A | U \in C\}$, meaning that each subset of A can be expressed as an intersection of A and a subset of C [75]. The Vapnik-Chervonenkis (VC) dimension is a measure of the capacity of a classification algorithm: it is the cardinality of the largest subset that the algorithm can shatter. Therefore, the VC dimension provides insight into the complexity that the learning algorithm can represent. It also follows that a pattern that requires a learning algorithm with high VC dimension has a high pattern complexity.

For artificial neural networks with sigmoidal activation function and fixed depth, the VC dimension is contained between the lower bound $\Omega\left(\omega log(\omega)\right)$ and upper bound $O(\omega^4)$, where ω is the number of programmable parameters [75]. For a neural network with a single hidden network, the number of programmable parameters is the sum of input neurons and hidden neurons. It follows that a neural network with more hidden neurons can characterize patterns that are more complex. Furthermore, when a pattern requires a larger amount of hidden neurons to be learned, it follows that this pattern has higher internal complexity. This is an important consideration for the experimental characterization of model-building resistivity in section 4.7.

4.2.5 Security Threats

4.2.5.1 Model-Based Token Impersonation

Highly successful modeling attacks on Arbiter PUFs, RO-PUFs, feed-forward Arbiter PUFs, simple PUFs, and XOR-Arbiter PUFs were demonstrated by Ruhrmair et al. [36] on synthetic PUFs. These attacks were based on logistic regression using RPROP gradient descent and evolutionary strategies. More recently, the authors expanded their results to include FPGA and ASIC implementations and showed results resembling those of their synthetic implementation [71]. In relevance to this contribution is that they broke a 128-bit and 64-bit Arbiter PUF in mere seconds, which leads to the assumption that this is indeed one of the weakest PUFs with regard to modeling attacks.

4.2.5.2 Side-Channel Information Extraction

It is in the nature of reasonably complex physical devices to leak information on the operation that is being performed, which is indirectly observable as a side-channel leakage through measurements of power, temperature, and other parameters. Side-channel attacks passively exploit this to extrapolate confidential information. This is true even for cryptographic modules such as the Advanced Encryption Standard (AES), which is approved by the U.S. National Security Agency (NSA) for top-secret documents. Researchers have demonstrated that a side-channel unaware implementation of 128-bit AES can be attacked to reveal the entire secret key with only 8,000 measurements [76]. One technique against side-channel leakage is power randomization to reduce cross-correlation between power trace and performed operation [53]. Another approach is reducing leaked

information by normalizing the consumed power in logic gates and interconnects [76]. Additionally, obfuscation is a technique to increase the difficulty of understanding and reverse-engineering hardware [53], [77], which limits the applicability of side-channel attacks as design and internals are hidden.

4.2.5.3 Physical Access and Tampering

Physical security is one of the root causes for the invention of the PUF. Conventionally, confidential information such as a secret key for encryption/decryption is stored in on-chip non-volatile memory (NVM), as this information has to be preserved even when the device is powered down. However, this form of storage is vulnerable, as well-equipped adversaries can de-package the chip and physically access and read the contents of the NVM [29], [78]. Moreover, adversaries can physically tamper with circuitry, for example utilizing focused ion beams, to modify or disable components [28]. Conventional hardware security approaches such as tamper detection, metal meshes, and similar techniques are expensive in power and area and therefore are not applicable for lightweight devices [79]. Furthermore, semi-invasive attacks such as optical fault induction [80] allow adversaries to change individual bits in microcontroller memory by illumination. Therefore, the volatility that PUF provides is one of its strongest characteristics: when physically tampered with, the behavior changes and thus the internal secret is destroyed. In section 4.5 we further discuss that PUF by itself is not a safeguard against invasive attacks for the entire device.

4.2.6 PUF Architectures and Protocols

4.2.6.1 Reliance on Strong PUFs

The promise of a Strong PUF is to provide a large challenge response space that is infeasible to model with state-of-the-art machine-learning techniques. A number of PUF-based protocols expose both the challenge and response and hence rely on this intrinsic modeling resistivity. However, no PUF design for a Strong PUF achieves scalability and resistivity against model-building attacks without sacrificing reliability. This remains the most common issue among all protocols studied by Delvaux et al. [79]. For these reasons, an architecture that enables any PUF design to become a true Strong PUF has significant merit and benefits a wide range of existing protocols. In section 4.3, we propose PolyPUF that achieves these goals.

4.2.6.2 Reverse Fuzzy Extractor

As previously described, most PUF-based authentication scenarios rely on a Strong PUF that can resist model-building attacks yet achieves high reliability. The Reverse Fuzzy Extractor (RFE) [81] attempts to avoid this requirement by hiding the actual PUF response: During authentication, the verifier issues a challenge $\boldsymbol{c}$ with random nonce x_{nonce}, which is an arbitrary random bit-string for one-time usage. The PUF device then generates the actual response $\boldsymbol{r}_{puf}$, which contains perturbations due to environment variations, and the corresponding helper data d_h. To hide the actual response $\boldsymbol{r}_{puf}$, the PUF device then releases a hash h_{PUF} that contains the response. Using helper data d_h and the true response $\boldsymbol{r}$, the verifier can construct $\boldsymbol{r}'$ which should match $\boldsymbol{r}_{puf}$ if the PUF is authentic. The trusted party can then indirectly compare $\boldsymbol{r}'$ and $\boldsymbol{r}_{puf}$ by evaluating $hash(\boldsymbol{r}') = h_{PUF}$.

Releasing helper data always leads to a loss of entropy [79], [82], and despite the measures taken the RFE, Delvaux et al. [79] demonstrated that an adversary can selectively issue challenges to solve a system of linear equations that characterizes the helper data leakage.

Furthermore, we emphasize that the entropy loss due to helper data leakage requires a significantly larger PUF challenge and response lengths. This in turn requires a larger challenge expander and a larger hash function [32].

4.2.6.3 Slender PUF Protocol

The slender PUF protocol attempts to invalidate machine-learning-based attacks by exposing only a random substring of the response. The challenge to the PUF is determined by combining cryptographic nonces from the prover and verifier through a linear feedback shift register (LFSR). These nonces are generated from true random number generators (TRNGs). This challenge is available to both the prover and verifier, and cannot be fully controlled by one party. While it was the first protocol to efficiently introduce noise into the PUF response, it is limited by the requirement that it can only be applied on a true Strong PUF that meets the avalanche criterion. Such a Strong PUF can be approached with a k-XOR Arbiter PUF, but this leads to a significant increase in the error rate. In [83], the authors demonstrate that the substring has to consist of 1250 bits to achieve an acceptable false rejection rate of 1% with a 4-XOR Arbiter PUF. Additionally, the usage of a random nonce on the prover side probabilistically enables an adversary to select the nonce such that the resulting challenge has a small Hamming distance to a known challenge.

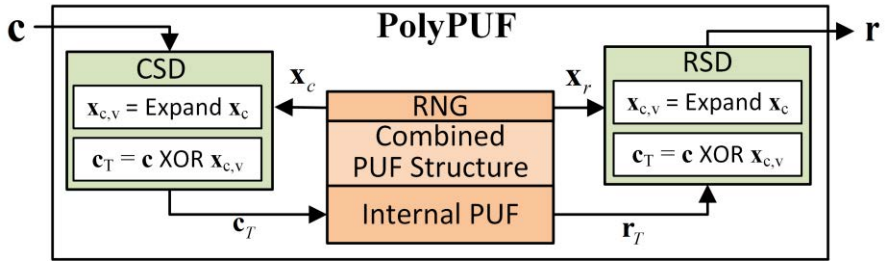

Figure 4.2 PolyPUF with challenge self-divergence (CSD), response self-divergence (RSD), and internal PUF structure.

4.2.6.4 Noise Bifurcation PUF Architecture

Yu et al. [72] described a Noise Bifurcation PUF (NBPUF) architecture for PUFs that increases the noise for an adversary, reducing their ability to perform machine-learning attacks, without increasing the noise observed by the trusted party [72]. In their architecture, out of every d bits in the response, $d-1$ bits are randomly discarded. Therefore, this architecture requires that the response be pre-expanded by a factor of d. In authentication, only those bits that have deterministic behavior are considered, meaning that all d bits have the same value. Due to these discarded bits during authentication, the response has to be pre-expanded by a factor of $d \cdot 2^{d-1}$ and therefore increases exponentially. The evaluation is performed on synthetic PUFs and demonstrates that machine-learning does not converge with a dataset of 1 million CRPs when this architecture is applied to a 5-XOR and 6-XOR Arbiter PUF.

4.3 PolyPUF Architecture

The PolyPUF challenge-response behavior can take many different shapes, and randomly changes between them. As the shape changes randomly, an adversary cannot learn it using model-building attacks. However, a trusted party can use secret knowledge to verify the authenticity of responses despite the random behavioral changes. The ultimate goal of PolyPUF is to decouple the observed response $\boldsymbol{r}$ from the issued challenge $\boldsymbol{c}$ so that model building becomes impossible, while maintaining reliable challenge-response behavior. For this purpose, the challenge-response mapping is truly random for each individual output bit, and therefore goes beyond the complexity achieved in the NBPUF. Therefore, PolyPUF enables true Strong PUFs that can withstand model-building attacks. The architecture is shown in Figure 4.2 and is described in detail in the following subsections.

4.3.1 Random Number Generation

Many PUF architectures and protocols rely on TRNGs as a core component, but do not provide sufficient measures to safeguard them against physical attacks [30], [79]. Defending such components for large random numbers is difficult and expensive, as the complexity of defense measures to achieve physical security increases with the component's footprint.

The PolyPUF architecture overcomes these concerns by utilizing a very small TRNG that can be derived directly from the internal PUF and hence introduces minimal resource overhead and security concerns. During enrollment of the PUF, the trusted party programs a randomization challenge $\boldsymbol{c}_x$ into the PUF, which was observed to have low reliability. One can derive the random bit-vector $\boldsymbol{x}$ by XOR-reducing the PUF response $\boldsymbol{r} = PUF(\boldsymbol{c}_x)$ such that

$$\boldsymbol{x}(i) = \bigoplus_{j\,=\,0\,\ldots\,|\boldsymbol{x}|\,-\,1} \boldsymbol{r}_i(i \bmod |\boldsymbol{x}| + j \cdot |\boldsymbol{x}|) \tag{4.2}$$

This approach is physically secure, as the TRNG cannot be modified without changing the actual behavior of the internal PUF, which would render it useless. Furthermore, the XOR gates required for the TRNG can be embedded into most existing PUF designs.

The theoretical basis for this approach of generating small random numbers lies in the entropy maximizing nature of the XOR operation used in equation (4.2). The bit-vector can be approximated as a series of independent random bits $\boldsymbol{r}(i)$ with a bias, such that $E[\boldsymbol{r}(i)] = \mu$. This is a conservative approach for entropy estimation, as the entropy of the XOR of two random variables is at least the entropy of the individual random variables. Under these considerations, the bias of each bit of the random seed vector is:

$$E[\boldsymbol{x}_i] = \frac{1}{2} + (-2)^{\left\lfloor \frac{|\boldsymbol{r}|}{|\boldsymbol{x}|} \right\rfloor - 1} \left(\mu - \frac{1}{2}\right)^{\left\lfloor \frac{|\boldsymbol{r}|}{|\boldsymbol{x}|} \right\rfloor} \tag{4.3}$$

If the randomization challenge $\boldsymbol{c}_x$ is improperly selected to have an error rate of only 10%, which is worse than an average challenge in the Arbiter PUF [83], the response bias would be $\beta = 1 - \epsilon = 0.9$. Even in this situation, a 64-bit response can generate a 3-bit random seed vector with an expected value of $E[\boldsymbol{x}_i] = 0.505$ according to equation (4.3), which is very close to the ideal 0.5. As discussed, it is critical that the random seed is short, and the expected value increases to $E[\boldsymbol{x}_i] = 0.66$ for a random 12-bit seed.

When higher randomness is desired, e.g. because of small $|\boldsymbol{r}|$ or large $|\boldsymbol{x}|$, this process can be repeated and the random numbers of each iteration can be XOR-combined.

4.3.2 Challenge Self-Divergence

Challenge self-divergence achieves challenge-response diffusion by diverging the challenge, which is issued to the device from the true challenge, which is processed through the internal PUF. Therefore, the true challenge is concealed from any outside party and is only observed and known within the security device itself.

First, the challenge divergence seed $\boldsymbol{x}_c$ is generated as a short true random number by the PUF and its components as described in subsection 4.3.1. The apparent reasons for a short $\boldsymbol{x}_c$ lie in reduced cost and facilitated PUF-based implementation, but it also has a profound effect on the verifiability and physical security of the PUF. We will show below that $|\boldsymbol{x}_c|$ can be a very small value.

Second, $\boldsymbol{x}_c$ is maximally expanded into divergence vector $\boldsymbol{x}_{c,v}$ by repetition to match the challenge length of the PUF, such that $|\boldsymbol{x}_{c,v}| = |\boldsymbol{c}|$. Finally, the true challenge is derived as $\boldsymbol{c}_T = \boldsymbol{c} \oplus \boldsymbol{x}_{c,v}$. This implies that each issued challenge is transformed into one of $2^{|\boldsymbol{x}_c|}$ possible true challenges.

The XOR operation is critical, as it combines the original challenge divergence seed and the original challenge with maximum entropy, as every output bit depends on both input bits. Moreover, this form of self-divergence performs a uniform action across all original challenge bits and is based on uniform XOR operations for random number generation. Therefore, it provides a strong foundation for resistivity against side-channel attacks and can be optimized for side-channel leakage minimization at the layout level.

For PolyPUF, $|\boldsymbol{x}_c| = 2$ is a viable selection to achieve the desired polymorphic behavior because it sufficiently diverges the challenge. Further increase of this would allow more possible challenges, but also requires a larger random seed and more computation on the server side, which are not desirable.

4.3.3 Response Self-Divergence

Challenge self-divergence only hides the challenge and therefore does not provide true polymorphic behavior – knowing the true response of the PUF can be a starting point for an advanced machine-learning exploit. Two problems remain. First, additional decoupling is needed to achieve sufficient challenge-response diffusion without further increase of $|\boldsymbol{x}_c|$, which would have the aforementioned detrimental effects. Second, it does not provide any improvement to the bias that is typically observed in PUF behavior.

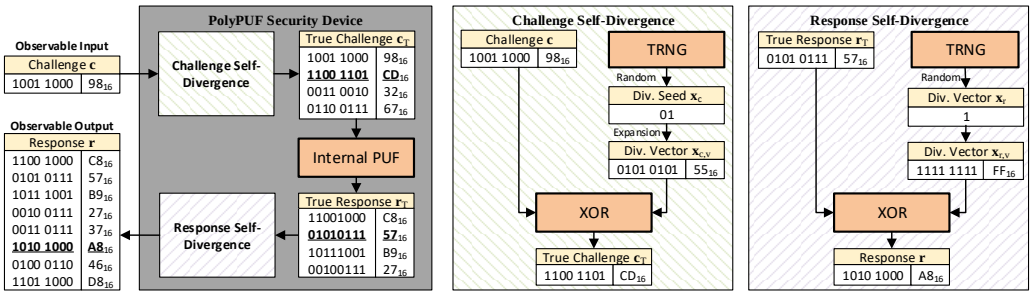

Figure 4.3 Example of polymorphic behavior of PolyPUF with $|x_c| = 2$ and $|x_r| = 1$. The left side shows the overall processing steps in PolyPUF. A third party may observe any of the responses in R, as the actually observed response is non-deterministically generated through challenge and response self-divergence, which are shown in the center and right boxes, respectively.

To overcome both problems, we present a response self-divergence scheme that complements challenge self-divergence for truly polymorphic behavior. For this purpose, we have investigated response self-divergence through a shuffling approach as well as an XOR approach similar to the one described in section 4.3.2. In both cases, the true response $\boldsymbol{r}_T$ is divided into groups $\boldsymbol{g}_i$. Based on the same mechanism employed in section 4.3.1, a small response divergence seed vector $\boldsymbol{x}_r$ is generated. For the XOR approach, each group $\boldsymbol{g}_i$ is XORed with a response divergence seed $\boldsymbol{x}_r$. In the shuffling approach, the bits in the divergence seed determine whether consecutive groups are exchanged. The investigation showed that XOR performed much better, and the evaluation can be simplified to the following example: Consider a case where shuffling is used with $|\boldsymbol{x}_r| = 1$, $|\boldsymbol{r}_t| = 2$, and $|g_i| = 1$. The bias of the response in this example is $E[r(i)] = P(\neg x_r(i))E[r_T(i)] + P(x_r(i))E[r_T(i+1)]$. Therefore, bias is very possible and not drastically reduced. For the case of XOR, as we outlined in subsection 4.3.1, the entropy of the response is as good as the entropy of the divergence seed.

Due to response self-divergence and as a side-effect of polymorphism, PolyPUF achieves the strict avalanche criterion, as an individual challenge bit-flip leads to a bit-flip in the output with an average probability that can arbitrarily approach 0.5 based on design requirements and length of the response self-divergence seed.

4.3.4 Polymorphism

Together, the challenge and response divergence grant polymorphic behavior, as the challenge-response behavior changes randomly. This behavior is illustrated in Figure 4.3 for a very small PolyPUF with $|\boldsymbol{x}_c| = 2$ and $|\boldsymbol{x}_r| = 1$. In this example, a single challenge

has eight possible responses, and PolyPUF will unpredictably issue one of these. Given a response to challenge $\boldsymbol{c}$, it is practically impossible to infer which true challenge $\boldsymbol{c}_T$ was actually evaluated by the internal PUF, because a large number of equally probable combinations of true challenge and true response exist. Even when a large CRP set is gathered, the true challenges and responses cannot be derived. In fact, this polymorphism is not restricted to adversaries, and the trusted party faces the same non-determinism in PUF behavior. However, with the model of the internal PUF, the trusted party can explore the range of possible responses and thereby decide on the authenticity of the received response, as will be shown in section 4.4.

We further explore the advantages of PolyPUF by discussing an alternative implementation where multiple actual PUFs are interchangeably used. The cost of implementing multiple PUFs is not negligible, but may be bearable in all but ultra-lightweight applications. However, this alternative approach also has the following downsides: If multiple internal PUFs were used, either (i) all of them would compute the response and only one PUF would actually issue the response to the requestor, or (ii) only one PUF is selected to compute and issue the response while the others remain inactive. Approach (i) has a considerable power overhead, and potentially reduces reliability due to cross talk between PUF instances, which requires careful design work. Approach (ii) leaks significant side-channel information, as each PUF is unique and therefore exhibits a different power profile. Moreover, actually implementing multiple PUFs, independent of approaches (i) and (ii), introduces the problem that advanced machine-learning techniques such as ANN discussed in section 4.2.4 could perform space separation and hence cluster the PUFs and identify their individual challenge-response behaviors. PolyPUF does not suffer from any of these weaknesses, as all responses are issued by the identical internal PUF. Hence, the polymorphism originates in non-deterministic self-divergence instead of space expansion and provides stronger security.

4.4 PolyPUF Application

Conceptually, PolyPUF was designed to provide a strong foundation against all threat vectors identified in the introduction. The resistivity against modeling attacks arises from the polymorphic behavior described in section 4.3.4. The minimization of side-channel information leakage originates in the design optimizations to achieve said polymorphism. In the challenge self-divergence, the initially generated random number is small to have a

small range of possible challenges that the trusted party has to explore during authentication. Additionally, the proposed algorithm for performing challenge self-divergence takes information leakage into account. For instance, consider a scheme to derive the true challenge by a summation of the original challenge with a random bit-vector $\boldsymbol{x}$, which provides a range of $[\boldsymbol{c}, \boldsymbol{c} + |\boldsymbol{x}|]$ consecutive challenges. However, this approach (i) leaks more side-channel information as summation leaks far more side-channel information than a simple XOR operation, and (ii) as the possible challenges lie close to one another, knowledge of one CRP allows inference of other similar challenges due to the weak diffusion of the internal PUF, as reflected in equation (4.1).

4.4.1 Wide Applicability

As an architecture, PolyPUF has the unique advantage that it can be applied to almost every PUF design that allows a large challenge-response space and can turn it into a true Strong PUF with model-building resistivity. Due to challenge and response self-divergence, PolyPUF does not pose limiting requirements on bias, complexity, or reliability of the internal PUF. Even when the internal PUF exhibits biased behavior and does not achieve diffusion or meet the avalanche criterion, PolyPUF will exhibit high diffusion and meet the avalanche criterion.

The source of this advantage lies in the polymorphic behavior specified in section 4.3.4; the model-building resistivity is grounded in this polymorphic behavior instead of characteristics of the internal PUF. We furthermore argue that PolyPUF is even applicable to the weakest known PUF designs where existing architectures such as the NBPUF do not provide sufficient improvement. We experimentally show that PolyPUF is indeed capable of this by evaluating it with a variety of Arbiter PUFs in section 4.7.

4.4.2 Reliability

Two of the major benefits of PolyPUF are the reliability and scalability this architecture achieves. Existing approaches to achieve a Strong PUF rely on combination of the responses of multiple PUFs to increase the challenge-response complexity. This, however, decreases the reliability of the resulting PUF, as an error in any of the individual PUFs leads to an error in the resulting PUF. For instance, the XOR-Arbiter PUF has an error rate that increases almost linearly with the number of contributing PUFs. A 4-XOR Arbiter PUF, which is not sufficient to achieve model-building resistivity, was shown to have an error

rate of more than 30% [72]. In contrast to this, PolyPUF does not negatively affect reliability at all, as no error-magnifying combination of multiple PUFs is implemented. This means that PolyPUF can achieve the reliability of any single PUF instance that it is applied on.

4.4.3 Authentication Protocol

We consider parametric authentication, the most common scenario for PUF [7], [30], [72]. Here, the trusted party generates a true model for the internal PUF in an enrollment phase with access to the internal PUF. Afterward, any outside access to the internal PUF is physically deleted, e.g. through fuses.

Accurate authentication is possible despite the challenge and response self-divergence due to several considerations in the PolyPUF specification. The challenge and response self-divergence seeds $\boldsymbol{x}_c$ and $\boldsymbol{x}_r$ were specified as small bit-vectors, which allows the trusted party to computationally explore all options. By exploring the previously described operations and querying the secret model, the trusted party can find all $|S_{CRP}| = 2^{|\boldsymbol{x}_c|+|\boldsymbol{x}_r|}$ possible responses. Although this equation is exponential, we emphasize that: (i) the experimental evaluation shows that $|\boldsymbol{x}_c| = 2$ is sufficient to thwart the strongest known machine-learning techniques; (ii) $|\boldsymbol{x}_r|$ can be specified by the trusted party to balance computational cost on the server side with machine-learning resistivity in the PUF; (iii) evaluating a known and established PUF model typically consumes a minimal amount of time; (iv) whereas size and energy cost of the PUF are critical, the computational requirements to the server are much more bearable.

Finally, the response of the PUF is authenticated if it is part of the set of possible responses. When bit-errors are considered, a trusted party can iteratively compute the candidate response $\boldsymbol{r}_c = \underset{\boldsymbol{r}_i \epsilon S_{CRP}}{\arg\min} |\boldsymbol{r}_i - \boldsymbol{r}|$. In the simplest form, $\boldsymbol{r}$ can be accepted if $|\boldsymbol{r}_c - \boldsymbol{r}| < t_\epsilon$, where t_ϵ is a scenario-specific authentication threshold. We emphasize that a small threshold should be sufficient, as PolyPUF is the only known architecture to increase modeling resistance without negatively affecting reliability.

4.5 Security Considerations

4.5.1 Pitfalls of Challenge Expansion

Challenge expansion is a technique typically employed for lightweight PUFs. Challenge expansion implies that the application or protocol requires $|\boldsymbol{r}| > 1$ output bits from the PUF. For example, in an encryption scenario, a secret key of more than 128 bits is desired to increase the time consumption of brute-force attacks. Similarly, in authentication where the PUF response is used for authentication, a large output length is desirable to minimize the probability of a random guess achieving false acceptance.

However, implementing a large number of PUFs can be expensive in the power and circuit area. Therefore, a small number of actual PUFs with $|\boldsymbol{r}_{puf}| < |\boldsymbol{r}|$ output bits is expanded to a length of $|\boldsymbol{r}|$ by challenge expansion. In a typical implementation, only a single PUF with $|\boldsymbol{r}_{puf}| = 1$ is implemented. The desired number of output bits $|\boldsymbol{r}|$ is sequentially generated through $|\boldsymbol{r}|$ challenges that are produced by a pseudo-random number generator (PRNG). As the original challenge can be used as a seed, the responses remain consistent across multiple queries. Whereas this is highly cost efficient and maximally reuses each PUF instance, several security and practicality concerns exist.

First, challenge expansion itself is not physically secure. One of the main advantages of PUF is its volatility that limits the success of invasive physical attacks. However, the challenge expansion circuitry remains physically attackable and thus requires extensive conventional defenses. This diminishes some of the cost savings introduced by PUF and reduces the range of viable applications. Second, implementing a large pseudo-random number generator or cryptographically secure hash function to perform challenge expansion is expensive in itself [32]. As some PUFs are very lightweight, the difference between a challenge-expanded PUF and a PUF that actually consists of multiple parallel elements may be insignificant compared to the security advantages. Additionally, challenge expansion decreases the diffusion described in section 4.2, as all response bits originate in the identical PUF. Furthermore, it is common that a single PUF exhibits only weak diffusion and strong bias, as the nano-scale intrinsic variations cannot be controlled. Therefore, relying on a single PUF increases the likelihood of a device that is unusable from a security perspective. Besides, designs that employ a single PUF with a challenge expander expose n_{out} responses

of the PUF in a single CRP, hence allowing any adversary to gather large per-PUF CRP sets.

For these reasons, the internal PUF is proposed to be implemented with multiple individual PUF elements. In the experimental evaluation, the internal PUF is comprised of individual PUF elements for each configuration without any challenge expansion.

4.5.2 Reliance on True Random Numbers

Multiple PUF designs and architectures involve the utilization of TRNGs without providing guidelines for a secure design for this element. Particularly in this scenario involving PUFs, all relevant components have to achieve a certain degree of resistivity against invasive attacks described in section 4.2.5. Without this consideration, an adversary may tamper, disable, or guide the generation of random numbers and hence compromise security.

To illustrate this shortcoming, the adversary could modify the nonce generator in the NBPUF architecture to a fixed bit-vector if insufficient countermeasures are implemented. Then, the adversary has full control of the challenge generation and can exploit this by repeatedly issuing the same challenges.

To utilize true random numbers without exposing a vulnerability or requiring extensive conventional security measures, PolyPUF requires a very small TRNG. Furthermore, an implementation that re-uses the internal PUF for inherent security against invasive attacks was outlined in section 4.3.1.

4.5.3 Entropy Oblivious Design

We challenge the reliance on error-correction or high-acceptance thresholds for PUF-based authentication. Koeberl et al. have recently performed an extensive entropy and error-correction analysis and shown that min-entropy is often over-estimated [82]. Additionally, they demonstrated that correcting an error of 15% and PUF min-entropy of 15% requires a PUF response length that is more than 15 times the size of the desired entropy. This enormously increases the cost of PUF and diminishes the lightweight characteristic that is one of PUF's strongest features.

Similarly, relying on a higher authentication threshold instead of error correction also introduces new problems. For one, the length of the PUF has to be increased so that the probability of random guessing remains small. Furthermore, due to noise in the PUF

response, it is very difficult to discern the noisy PUF from an emulated PUF that takes advantage of PUF bias. Delvaux et al. [79] have made similar conclusions and have shown that existing PUF authentication protocols have insufficient security and practicality. These results imply that employing XOR, feedback, and feed-forward-based architectures to increase modeling resistivity provides insufficient improvements and introduces other difficulties.

4.6 Attack Analysis

4.6.1 Random Guessing

The simplest and least effective approach to impersonate a PUF is to respond to a challenge with a random response. In the following, the threshold t_ϵ refers to the maximum Hamming distance to the correct response for which the received response can be accepted as authentic.

For PolyPUF, the probability of a random guessing attack is slightly increased, as multiple responses are possible for any given challenge. In the following, we consider the 64x64 PolyPUF with $|\boldsymbol{x}_c| = 2$ and $|\boldsymbol{x}_r| = 3$. Therefore, the probability of a false acceptance in this ideal scenario without consideration of bit-errors due to environment variations equals:

$$P_{FA,ideal} = 2^{|\boldsymbol{x}_c|+|\boldsymbol{x}_r|-|\boldsymbol{r}|} = 1.73 \cdot 10^{-18}$$

Under consideration of bit-errors that are typical due to PUF volatility, the probability for false acceptance increases. The trusted party can compute all possible responses, identify the most likely correct response r_{ideal}, and accept the provided response if $\Delta_{\boldsymbol{r}_{ideal},\boldsymbol{r}} \leq t_\epsilon$. Recall that PolyPUF can be assumed to be unbiased due to the response self-divergence, therefore the bias is $\beta = 0.5$. The false acceptance probability under consideration of bit-errors equals:

$$P_{FA,real} = 2^{|\mathbf{x}_\mathrm{c}|+|\mathbf{x}_\mathrm{r}|} \sum_{i=0}^{t_\epsilon} \binom{|\boldsymbol{x}_r|}{i} \beta^i (1-\beta)^{|\boldsymbol{x}_r|-i}$$

Assuming a 10%-bit error rate in the internal Arbiter PUF, one may set the threshold to $t_\epsilon = 12$ and achieve a false acceptance probability of $P_{FA,real} = 7.3 \cdot 10^{-6}$.

Note that this is only a factor of 32 larger than the false acceptance probability when a simple Arbiter PUF is used, without PolyPUF. Especially given larger PUF sizes as used in practice, e.g. $|\boldsymbol{x}_r| = 256$, this increase in false acceptance probability is diminishing.

The probability of false rejection for the proposed protocol is almost negligibly larger than the probability for the case that a simple Arbiter PUF is authenticated. The bit-error rate is denoted by ϵ and is not increased by PolyPUF. The probability for correct acceptance of the Arbiter PUF is:

$$P_{CA,Arb} = \sum_{i=0}^{t_\epsilon} \binom{|\boldsymbol{x}_r|}{i} \epsilon^i (1-\epsilon)^{|\boldsymbol{x}_r|-i}$$

Similarly, the probability of correct acceptance for PolyPUF can be derived by considering that a larger number of responses are acceptable:

$$P_{CA} = P_{CA,Arb} + \left(2^{|\boldsymbol{x}_c|+|\boldsymbol{x}_r|} - 1\right) \sum_{i=0}^{t_\epsilon} \binom{|\boldsymbol{x}_r|}{i} \beta^i (1-\beta)^{|\boldsymbol{x}_r|-i}$$

Finally, the probability of false rejection is $P_{FR} = 1 - P_{CA}$ and can be quantified as $P_{FR} = 0.01$ for this example implementation.

4.6.2 Direct Machine Learning

The direct approach of machine-learning on a large CRP set is bound to fail against PolyPUF, as these CRPs are virtually guaranteed to be derived from multiple different PolyPUF instantiations. As the exact nature of the instantiation is unknown, it is impossible to derive the relation between responses of different challenges across different instantiations. The experimental results for this are shown in section 4.7.2.

4.6.3 Brute-force Machine Learning

In this subsection, we discuss the cost of performing a brute-force attack on PolyPUF by gathering all possible responses for a given number of challenges, and then training a model for each possible combination of responses. Let us assume that the internal PUF is considered to be accurately learned if the number of known CRP pairs reaches k_{CRP}. Thus, the attacker aims to gather k_{CRP} challenge response pairs of a single instantiation of PolyPUF. Given challenge seed vector length $|\boldsymbol{x}_c|$ and response seed vector length $|\boldsymbol{x}_R|$, the number of PolyPUF instantiations is $r_{limit} = 2^{|\boldsymbol{x}_c|+|\boldsymbol{x}_r|}$. For a given challenge $\boldsymbol{c}_i$, the

probability that the attacker has observed all r_{limit} instantiations after issuing this challenge n_{trial} times is:

$$P_i = \sum_{k=0}^{r_{limit}-1} (-1)^k \binom{r_{limit}}{k} \left(\frac{r_{limit} - k}{r_{limit}}\right)^{n_{trial}}$$

For the example implementation with $|\boldsymbol{x}_c| = 2$ and $|\boldsymbol{x}_r| = 3$ it follows that $r_{limit} = 32$. To achieve a probability $P_i = 99\%$ of having observed all possible instantiations, the adversary has to issue the same challenge $N_C = 260$ times. Thus, the adversary is required to perform $260 \cdot K_{CRP}$ authentications with the PUF device to gather a sufficiently large dataset.

Once this dataset is established, the adversary has to train one model for each permutation of CRPs in the dataset. Thus, the number of models to be trained is $n_{models} = r_{limit}^{k_{crp}}$. This brute-force approach guarantees that one of the models was trained on a pure CRP set that corresponds to a single instantiation. However, the number of models that need to be learned increases exponentially, and therefore this is not a feasible attack. Even when a simple Arbiter PUF is used, $K_{CRP} = 5000$ and $N_{CR} = 32$ require training of approximately $5.6 \cdot 10^{7525}$ models.

4.6.4 Cross Inference Attack

There are two possible attack vectors for the adversary: Attempt to learn the internal PUF by gathering true challenge and true response pairs, or attack one shape of PolyPUF by gathering a CRP set that corresponds to a single shape.

The adversary cannot identify the actual values for the divergence seeds, but he or she can characterize several of them relative to an assumed initial value of x_{c0} and x_{r0} from the initial CRP, which we denote by $CRP_0 = \{c_0, x_{c0}, x_{r0}\}$. The adversary can enumerate all $S_c = 2^{|x_c|}$ possible true challenges and trivially characterize one response with regard to the initial seeds: $CRP_i = \{c_0 \oplus k_{c,i}, x_{c0} \oplus k_{c,i}, x_{r0}\}$. This can then be expanded so that all possible responses for these challenges are well characterized with regard to x_{c0} and x_{r0} by enumerating the possible response self-divergence operations: $CRP_{ij} = \{c_0 \oplus k_{c,i}, x_{c0} \oplus k_{c,i}, x_{r0} \oplus k_{r,j}\}$. Then, the set of understood CRPs is $S_{CRP} = 2^{|x_c|+|x_r|}$. However, it is impossible to learn and explicitly characterize a CRP that corresponds to a challenge not found in S_c. Therefore, the number of CRPs that can deterministically be clustered is limited

Algorithm 4.1 Targeted Machine-Learning Attack

Input:
c_{limit}– limit for number of total challenges to issue
t_{limit} – limit for repetitions of a single challenge
r_{limit} – number of possible PolyPUF instantiations
Output:
S_{CRP}- set of selected challenge-response pairs

Current challenge c=GENERATERANDOMCHALLENGE()
Selected Challenge-Response Pairs $S_{CRP} = \{\}$
Number of issued challenges $n_c = 0$
While($n_c < c_{limit}$)
    Previous challenge $c_{prev} = c$
  RANDOMIZE(c)
  Observed Responses $O_r = \{\}$
  Trial number $n_t = 0$
  While($n_t < t_{limit}$ And $|O_r| < r_{limit}$)
    r=PPUF(c)
    $O_r = O_r \cup r$
    $n_t = n_t + 1,\ n_c = n_c + 1$
  r_S = SELECTRESPONSE($O_r, S_{CRP}, c, c_{prev}$)
  $S_{CRP} = S_{CRP} \cup (c, r_s)$

Simplified Method: SELECTRESPONSE ($R_O, S_{CRP}, c, c_{prev}$)
Selected response $r = \underset{r_i \in O_r}{\operatorname{argmin}} |r_i - S_{CRP}[c_{prev}]|$

Expensive Method: SELECTRESPONSE ($R_O, S_{CRP}, c, c_{prev}$)
Cost[r_i] = 0 for $r_i \in O_r$
For c_i where $HD(c_i - c) \leq 1, c_i \in S_{CRP,keys}$
  For $r_i \in O_r$
    Cost[r_i] = Cost[r_i] + $HD(r_i - S_{CRP}[c_i])$
Selected response $r = \underset{r_i \in O_r}{\operatorname{argmin}}\, Cost[r_i]$

to $|S_{CRP}|$. While infinitely many of these clusters can be created, they are all characterized with regard to a cluster-specific assumed x_{c0} and x_{r0} as reference point and therefore cannot contribute to a coherent model. This can be proven by considering the set of possible challenge self-divergence seeds $S_{x,c}$. It is notable that every possible bit-vector with length $|x_c|$ is contained in this set, thus $|S_{x,c}| = 2^{|x_c|}$. If challenge c_i was derived through seed $x_{c_i} = x_{c0} XOR\ k_i$, then any seed of derivable challenge c_{i+1} can be reduced to $x_{c_i+1} = x_{c_i} \oplus k_{i+1} = x_{c_0} \oplus (k_i \oplus k_{i+1})$. As all possible divergence seeds with length $|x_c|$ were explored, it must be that $(k_i \oplus k_{i+1}) \in S_{x,c}$.

4.6.5 Targeted Model-Building

Considering that the intended application of PolyPUF is to strengthen an internal PUF with weak machine-learning resistance, an attack may exploit the weak statistical properties of this internal PUF. These weak statistical properties imply that the avalanche criterion is not met, and that one CRP reveals information on arithmetically close other challenges as shown in equation (4.1). An adversary may attempt to exploit this behavior to identify those CRPs that are of the same PolyPUF instantiation. If the adversary can gather a large set of CRPs that correspond to the same PolyPUF instantiation, then he or she can perform simple machine-learning on it and should achieve results similar to those of directly attacking the internal PUF.

In Algorithm 4.1, we outline an approach to gather a set S_{CRP} of challenge-response pairs that have a better than average probability of belonging to the same PolyPUF instantiation. The attacker first issues a random challenge and remembers the response it receives. It then repeatedly selects new challenges that have a Hamming distance of one to the previous challenge. For this purpose, the function RANDOMIZE performs a random bit-flip and ensures that the resulting challenge has not been processed yet. For each of these challenges, the algorithms attempt to perform a full response-space exploration by re-issuing the same challenge until either (i) the maximum amount of distinct responses $r_{limit} = 2^{|x_c|+|x_r|}$ have been observed, or (ii) a desired limit of PUF transaction c_{limit} has been reached. At that point, it chooses the most probably response.

The algorithm repeatedly selects new challenges with Hamming distance of one to each other, so that a larger section of the challenge space can get explored while maintaining a short distance between consecutive challenges, so that the statistical weakness of the internal PUF is maximally exploited.

Although this approach appears to be a promising attack as it exploits the weak statistical nature of the internal PUF, two disadvantages have to be specified: (i) It has to be emphasized that equation (4.1) only applies to the average of a large set of CRPs, and certainly does not apply to every CRP. With this attack approach, there will be incorrect selections, which will lead to propagation errors. (ii) Any PUF without error correction will experience a certain number of bit errors, which will also propagate through the response selection and lead to a larger count of incorrect selections.

Even though it is not successful, it should be noted that this attack requires multiple orders of magnitude more challenge-response interactions with PolyPUF compared to a

Table 4.1 Model-building error rates for multiple PUF configurations with varying number of hidden neurons and one million CRPs training set size. The most effective number of neurons is bolded and used for the following evaluation steps.

Configuration	Size	Neurons	Error Rate
Simple Arbiter	32x32	10	0.042%
		20	0.041%
		30	0.042%
	64x64	10	0.049%
		20	0.049%
		30	0.052%
2-XOR Arbiter	32x32	30	0.82%
		40	0.78%
		50	0.82%
	64x64	40	0.78%
		50	0.78%
		60	0.79%
4-XOR Arbiter	32x32	50	8.99%
		60	8.97%
		70	8.84%
	64x64	60	9.14%
		70	8.98%
		80	9.02%

direct attack against the internal PUF. The experimental results of this attack are presented in subsection 4.7.4.

4.7 Experimental Evaluation

We evaluated a synthetic implementation of PolyPUF under machine-learning attacks and compare results to our implementation of the NBPUF architecture, described in section 4.2.6.4. Both architectures are designed to increase the modeling resistivity, hence we evaluate with the weakest known internal PUF from section 4.2, the simple Arbiter PUF. For PolyPUF, $|\boldsymbol{x}_c| = 2$ and $|\boldsymbol{x}_r| = 3$. For the NBPUF, we select $d = 2$ as proposed by the authors. We note that for the evaluation of it, we allow four times the number of output bits compared to PolyPUF, as the architecture requires omitting three-fourths of output bits.

4.7.1 Machine-Learning Setup and Preparation

The ANN was trained with RPROP as specified in section 4.2.4 using Matlab. The termination criteria were set to be 1000 epochs, a performance gradient of less than 10^{-5}, or six iterations with decreasing validation performance.

We challenge the practice of performing machine-learning attacks on a single PUF output bit and emphasize that the following are based on training of all output bits. This has two reasons: First, PUF is based on intrinsic physical variations and when observing a single PUF output bit, the results can be skewed and may not be representative. Second, most machine-learning algorithms use a random initialization vector; therefore, a single response bit evaluation increases the evaluation dependence on this initialization.

The capability of ANNs can be controlled through the number of neurons in the hidden layer. Although there are heuristics, there is no analytical solution to derive the optimal size of the hidden layer for a practical problem such as building a PUF model. Therefore, we experimentally evaluated each of the internal PUFs with varying number of neurons in the hidden layer to find the least error rate configuration. These experiments simultaneously provided the baseline for the following experiments, and also provided insight into the current weakness of PUF scalability. The results of this evaluation are shown in Table 4.1. The optimal number of neurons was experimentally derived for each configuration and size of the Arbiter PUF, and is highlighted in bold font in the table.

As discussed in section 4.2.4.1, the number of neurons together with the error rate provides insight into the pattern complexity. These experiments emphasize a key problem with PUF: the CRP space is easily expanded, but the pattern complexity and resistivity against model-building attacks do not scale accordingly. Moving from a PUF size of 32x32 to 64x64 has almost negligible impact on the complexity of all three PUF configurations, which can be observed in the number of neurons in the hidden layer and the error rate.

After deriving the optimal number of neurons for each internal PUF, we performed experiments on PolyPUF and NBPUF with this optimal ANN configuration.

Table 4.2 Comparison of a basic PUF architecture, NBPUF, and PolyPUF architecture, each with multiple different internal Arbiter PUFs. For model-building error rate, closer to 50% is better, as it characterizes modeling resistivity. For random guessing probability P_{rand}, lower is better.

Architecture	**32x32 Internal PUF**				
	Model-Building Error Rate				$\boldsymbol{P_{FA,ideal}}$
	5k CRPs	50k CRPs	500k CRPs	1M CRPs	
Basic	0.66%	0.15%	0.053%	0.041%	2^{-32}
	6.81%	1.05%	0.77%	0.78%	
	38.81%	12.68%	8.94%	8.84%	
NBPUF	2.44%	0.048%	0.001%	0%	2^{-8}
	15.67%	0.92%	0.16%	0.13%	
	40.78%	14.05%	5.07%	4.57%	
PolyPUF	50.1%	49.97%	50%	50%	2^{-27}
	50.01%	49.97%	49.99%	50.01%	
	50.1%	49.96%	49.99%	49.95%	

Architecture	**64x64 Internal PUF**				
	Model-Building Error Rate				$\boldsymbol{P_{FA,ideal}}$
	5k CRPs	50k CRPs	500k CRPs	1M CRPs	
Basic	1.08%	0.2%	0.062%	0.049%	2^{-64}
	19.35%	1.43%	0.88%	0.78%	
	45.4%	18.51%	9.54%	8.98%	
NBPUF	7.28%	0.2%	0.003%	0%	2^{-16}
	27.05%	3.31%	0.23%	0.17%	
	45.74%	26.92%	5.84%	4.89%	
PolyPUF	49.99%	49.98%	50%	50.01%	2^{-59}
	50.16%	49.98%	50%	49.97%	
	51.97%	50%	49.99%	50.01%	

4.7.2 Resistance Against Malicious Model-Building

A comparison of ANN-based model-building against various configurations of a basic PUF architecture, the NBPUF, and PolyPUF is shown in Table 4.2. For each architecture, this table evaluates the model-building error against three internal PUFs: An Arbiter PUF which is expected to have the least model-building resistivity, a 2-XOR Arbiter PUF, and a 4-XOR Arbiter PUF with the highest model-building resistivity. To illustrate the increasing modeling accuracy with a larger training size, this table provides the error rates for training set sizes between 5k and 1M CRPs. The table shows that PolyPUF is the only architecture that can withstand model-building attacks even when 1 million CRPs are trained. Error rates in italic font are less than 5% and denote attack scenarios under which the PUF is considered to have been broken.

As the results are consistent and for brevity, we only discuss the results for 5k and 1M CRPs. For the same reasons, we only discuss the simple Arbiter and 4-XOR Arbiter PUFs.

The probabilities for successful guessing attacks for various PUF configurations are shown in the last column of Table 4.2. As previously described, the polymorphic nature of PolyPUF leads to a small increase in the success probability of such an attack, but it remains clearly lower than that of the NBPUF.

Three clear patterns can be observed from this table: (i) the model-building error rate and hence resistance increases with a more complex internal PUF; (ii) increasing the training set size reduces the prediction error rate; (iii) scaling the PUF by increasing the challenge-length from 32 bits to 64 bits has minimal impact. These results support the motivation that a new approach to model-building resistance is needed.

For a 32x32 Arbiter PUF as internal PUF, the basic architecture and NBPUF architecture are learned to error rates of only 0.66% and 2.44% respectively with only 5k CRPs in the training set. This implies that both of these architectures are considered to have been broken. As the training set size is increased, the error rate only decreases. In contrast, even with a significantly larger training set of 1M CRPs, the error rate against PolyPUF is 50%; therefore, the model-building resistance of PolyPUF exceeds that of the reference architectures by multiple orders of magnitude.

For the very complex 4-XOR Arbiter PUF as internal PUF, the baseline architectures perform much better. For the case of 5k CRPs, the 32x32 basic architecture and NBPUF achieve an error rate of 38.81% and 40.78%. The model-building attack against PolyPUF had an error rate of 50.1%, which is clearly higher, but all of these architectures are considered to have resisted this model-building attack. As the intensity of the attack is increased by increasing the training set size to 1M CRPs, PolyPUF shows its advantages. Whereas the basic architecture and the NBPUF have error rates of 8.84% and 4.57% respectively and thus were sufficiently modeled, PolyPUF has an error rate of 49.95% and thus remained resistant against this attack.

Another observation from Table 4.2 is the weakness of the NBPUF compared to the basic architecture. Although the error rate is larger for the NBPUF for smaller training sets, it is half of the error rate of the basic architecture under a training set of one million CRPs. This suggests that a large training set allows an ANN to train the NBPUF very efficiently.

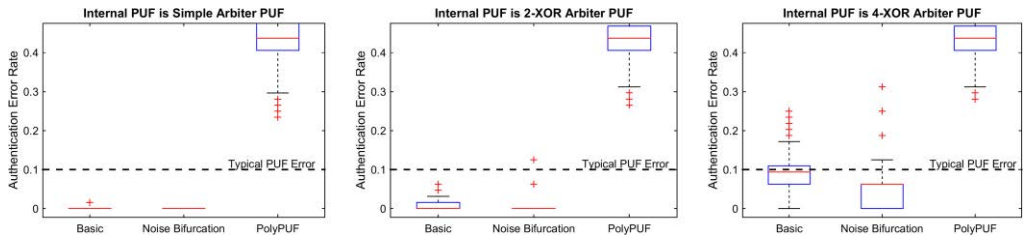

Figure 4.4 Comparison of five thousand malicious ANN authentication attempts. Depicted is the error rate of malicious authentication attempts, where higher is better. From left to right, the internal PUF is a simple Arbiter PUF, 2-XOR Arbiter PUF, and 4-XOR Arbiter PUF. Only PolyPUF has a consistent threshold to the illustrated typical PUF error rate.

Table 4.3 Results of the simple and improved targeted model-building attacks that attempt to exploit the statistical weakness of the internal PUF.

Metric	Simple	Improved
ANN error rate	49.44%	51.5%
Mean $HD(x_c)$	1.04	0.94
Mean $HD(x_r)$	1.56	1.35
Mean total HD	2.6	2.29
Correct selection	1.68%	5.81%

We propose that such a large training set is sufficient to identify the patterns in the PUF challenge-response behavior to identify those bits that randomly flip, and those that are consistent. Whereas random bits are impossible to learn, the consistent bits are learned with a much higher accuracy, as they remain consistent across multiple challenges. As the NBPUF only utilizes the consistent bits and discards the random bits in the evaluation, a well-trained ANN can model it to high accuracy.

Overall, the results in this table support our claim that PolyPUF is able to drastically increase the model-building resistivity of PUFs and that the polymorphic behavior is able to confuse machine-learning algorithms even when very large training data is employed.

4.7.3 Model-Building Authentication Attack

Figure 4.4 illustrates the per-bit error rate of the learned models for the basic architecture, NBPUF, and PolyPUF. For each of these architectures, the model for the 64x64 PUF which was trained with 1M CRPs from Table 4.2 was evaluated for 5k malicious authentication attempts. For authentication, the Hamming distance between malicious authentication attempts and noisy responses of a true PUF is significant to determine a viable acceptance

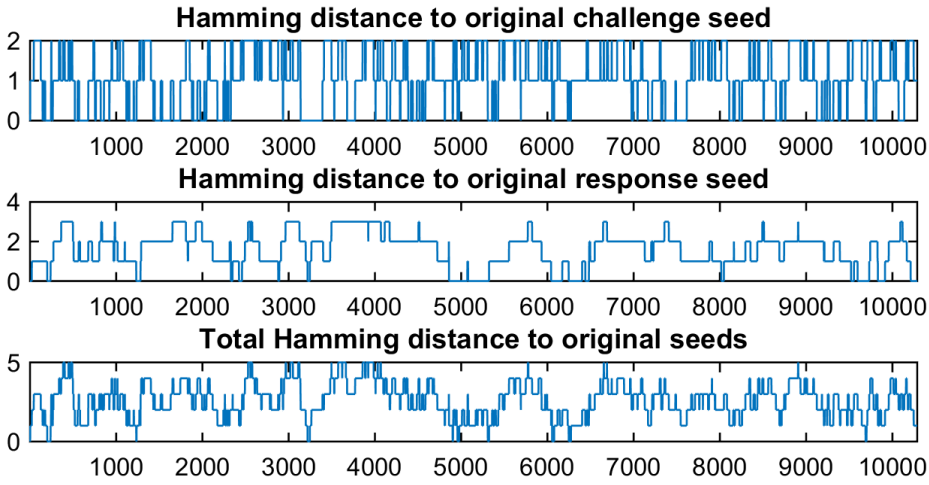

Figure 4.5 Hamming distances between the original seed value and the seed values in the selected responses in the simplified targeted model-building attack.

threshold. These figures emphasize that PolyPUF is the only architecture that maintains a consistent distance to a typical PUF bit-error rate, which is approximated to 10%.

4.7.4 Targeted Model-Building Experiment

The outcome of the targeted model-building attack is summarized in Table 4.3. The result of the simplified method of targeted model-building is an error rate of 49.44% and 50.1% for 1M and 100M total issued challenges. These total challenges correspond to 10,289 and 1,029,167 gathered challenges, respectively. These results demonstrate that this targeted model-building attack does not increase the success probability of an adversary, as PolyPUF behavior remains a virtual blackbox. The underlying reason for this successful defense is shown in Figure 4.5. The Hamming distances between the seed value for the initially selected response and subsequently selected responses are shown. Clearly, the fundamental idea behind the attack holds, and multiple consecutive challenges that correspond to the same divergence seed values are found. However, these results also demonstrate that the overall Hamming distance behaves almost like white noise, and therefore the attacker is learning multiple different instances of PolyPUF and cannot infer a clear model. Since it is impossible for the adversary to consider only those responses that have same seed values, the attack is bound to fail. The visualization of the Hamming distances in the more expensive selection approach is very similar and thus omitted for brevity.

4.7.5 Implementation Cost

PolyPUF offers multiple hardware implementation cost improvements. First, PolyPUF solves the requirement of error correction or cryptographic hashing for authentication and thereby removes two heavyweight but common components in existing PUF protocols [32], [79]. Second, due to the self-divergent approach, PolyPUF does not need a dedicated random number generator and can instead take advantage of the internal PUF itself. This reduces the number of hardware units as well as requiring less usage of conventional security techniques to prevent invasive attacks.

With the drastically increased strength against machine-learning, the internal PUF does not need to be duplicated multiple times. In contrast, existing protocols such as the Slender PUF protocol [30], [83] and the NBPUF [72] require a Strong PUF that meets the avalanche criterion. In the following evaluation, we approximate a Strong PUF with a 4-XOR Arbiter PUF.

The following is a quantitative analysis of the implementation overhead of PolyPUF compared to other PUF architectures for a Xilinx Virtex XC5VJX58T, which was chosen for comparability with existing literature [30].

To achieve comparability among multiple architectures, the implementations are driven by the requirements that (i) the internal PUF generates longer responses through challenge expansion and that (ii) a full response is generated as part of the PUF structure, rather than streaming individual bits of the PUF response. Each of these architectures is applied on a PUF with 64 input bits. As with the previous evaluation, we consider PolyPUF with 64 output bits and a simple Arbiter PUF as the internal PUF. As both the Slender PUF and the NBPUF require a Strong PUF, they are implemented with 4-XOR Arbiter PUFs, which have a higher error rate. The results of Rostami et al. [83] suggest an error rate of 13.2% for a simple Arbiter PUF and 43.2% for a 4-XOR Arbiter PUF.

The high error rate of the required internal PUF requires that the NBPUF transmits a very long response. According to the formulas provided by Yu et al. [72], even selecting a response length of 400 bits and a threshold of 133 bits leads to a false rejection rate and false acceptance rate of 11.9% and 7.3% respectively, both of which are far inferior to the statistical properties that PolyPUF achieves. For conservative comparison, we evaluate against this configuration, although it provides less security than PolyPUF.

Table 4.4 Energy cost comparison of lightweight PUF architectures based on a comparison of individual operations. These operations reflect a generation and transmission of one full response.

Operation	Slender	NBPUF	PolyPUF
PUF	4 * 400	4 * 800	3 * 64
LFSR	400 to 800	800	64
TRNG	64 + 5	64 + 800	N/A
2-XOR	64	64	3 * 64
4-XOR	400	800	N/A
Transmit	64b + 400b	128b + 800b	64b

Table 4.5 Implementation cost comparison of the primary security components of lightweight PUF architectures measured in look-up tables (LUT).

Component	Slender	NBPUF	PolyPUF
PUF	4*128	4*128	128
LFSR	10	10	10
TRNG	128	128	N/A
$\boldsymbol{x_c}/\boldsymbol{x_r}$	N/A	N/A	14
CSD/RSD	N/A	N/A	61
Total	650	650	213

The false acceptance and false rejection rates for the Slender PUF [30] can be reduced to the same equations as the NBPUF architecture for the case that $L_{sub} = L$, which is beneficial for evaluation of these statistical properties.

Our implementations of Slender PUF and NBPUF use the same approach for combining the prover and verifier nonce through an XOR operation. As a further energy optimization in the NBPUF, we assume that the PUF response is not generated for the bit that is discarded. Similarly, we assume that PUF responses that are skipped during substring selection in the Slender PUF are not generated to save energy.

The energy cost comparison in Table 4.4 shows the number of operations that each PUF architecture requires generating a full response. We do not further quantify the energy cost as it is highly dependent on implementation, platform (e.g. FPGA, ASIC), and means of data transmission. However, we would like to emphasize the clear trend that PolyPUF requires the least energy for both generation and transmission of a response. A single PUF operation is significantly more expensive than an XOR operation, as the signal has to travel

through 64 stages. Furthermore, we note that most protocols do not consider the cost of data transmission, although it may require most of the available energy budget. Particularly for wireless sensor networks and similar mobile applications, data transmission can be the primary source of energy consumption [68]. In contrast to PolyPUF, both reference architectures transmit longer responses and exchange nonces. PolyPUF requires 93.1% less transmission than NBPUF and 82.6% less than the Slender PUF. As Wander et al. [68] found that transmission of a single bit is equivalent to roughly 2090 clock cycles of execution on the microcontroller under test, it is clear that the reduction in response length leads to significant energy savings.

The hardware implementation cost of PolyPUF in comparison to the reference architectures is shown in Table 4.5. To achieve a conservative comparison, we evaluated only the security relevant components and disregarded control logic. Similar to PolyPUF, both reference architectures are highly efficient and do not require error-correction or cryptographic hash functions. Therefore, the primary cost reduction is achieved because these reference architectures require usage of a Strong PUF, which is here implemented as a 4-XOR Arbiter PUF. This PUF is roughly four times as expensive as the simple Arbiter PUF that PolyPUF employs. Additionally, PolyPUF employs the internal PUF for random number generation and has a small overhead for generation of x_c and x_r in contrast to the TRNG that the reference architectures require. As they operate in sequence and can therefore re-use the same hardware, challenge and response self-divergence are lightweight as well. Note that response generation in PolyPUF will be much faster, as this comparison assumed sequential generation of response-bits. To achieve a similar throughput to PolyPUF, the reference architectures would require more parallel PUF components.

4.8 Conclusion

The primary challenges for PUF are their reliability under environmental variations, and their resistivity against advanced machine-learning-based model-building attacks. Existing techniques to increase PUF model-building resistivity are not scalable due to their detrimental effect on PUF reliability.

We proposed PolyPUF, a widely applicable PUF architecture that employs challenge and response self-divergence to provide polymorphous PUF behavior. This changes the challenge-response behavior to be non-deterministic and unpredictable, while still being verifiable in an authentication scenario. In an extensive evaluation, this polymorphic

behavior was shown to provide strong resistivity against model-building attacks and was the only architecture to withstand an ANN model trained with one million CRPs by increasing the model-building resistance by more than an order of magnitude. Moreover, PolyPUF achieves model-building resistance without negatively affecting the reliability of the PUF device, which uniquely qualifies it for practical scenarios.

As part of our experimental evaluation, it was shown that neural networks with large training size overcome deterministic noise such as that induced by the NBPUF architecture. Therefore, truly random behavior such as that exhibited by PolyPUF is a necessity.

We have further demonstrated that PolyPUF introduces less hardware overhead than reference architectures, and reduces the energy cost of generating a PUF response. Additionally, PolyPUF requires transmission of much smaller responses, which can provide significant energy savings.

Although existing work has shown that synthetic PUFs behave very closely to silicon or FPGA implementations, the strength of PolyPUF should be evaluated in a silicon or FPGA implementation in future work. This will also enable exploration of side-channel leakage and optimized designs to counter this.

CHAPTER 5
HIGH-LEVEL SYNTHESIS FOR HARDWARE TROJAN HORSE DEFENSE

5.1 Introduction

The Internet of Things (IoT) is the next step towards pervasive and ubiquitous computing and has the potential to drastically change the society through constant recording, processing, and communication of data. Due to the need for light-weight, secure, and reliable communication, new protocols and use cases for the IoT are explored [84]. With the lightweight requirements of IoT, it is important to provide the flexibility of establishing where security enhancements are required, and to what extent these enhancements should be performed. For instance, an encrypted video stream that is transmitted by a smart video recording system is already protected in its transmission through the encryption. At the hardware level, the security enhancements can be focused on protecting the cipher key for encryption and decryption against any form of information leakage, rather than guaranteeing that the encoding of the video stream is leakage-free. Although the video footage may be considered confidential, it is easier to detect its leakage due to the size of the corresponding transmissions.

Security and trust in integrated circuits (ICs) remain an ongoing concern, as a security breach at the hardware-level exposes even provably secure algorithms and protocols to vulnerabilities. Among the threats that hardware security faces, hardware Trojans have emerged as one of the major security concerns due to the economically incentivized increased outsourcing of IC fabrication to third parties that cannot be fully trusted [85], [86]. Such Trojans are not only found in consumer grade electronics, but can exist in mission-critical military equipment as well. Recently, a backdoor was discovered in a military grade FPGA that was implemented in the silicon of the chip itself and could be used to extract secrets and even reprogram the device [87].

Hardware Trojans can be inserted in different stages of the design and fabrication process, and are characterized by the trigger, which activates the Trojan operation, and by the payload, which is the malicious deviation from the intended system behavior. Trojans

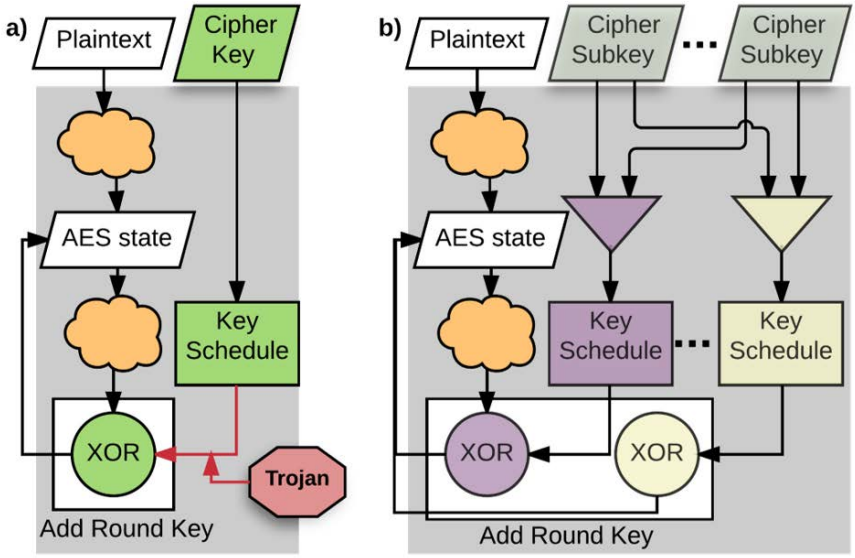

Figure 5.1. a) Simplified example of a hardware Trojan for indirect leakage of the cipher key in AES, b) dispersed cipher key to prevent hardware Trojan insertion.

are triggered by one or more rarely switching nets or a sequential combination of them, which makes activation during testing highly difficult, especially due to the increasing density of integration. The payload can have a destructive impact, e.g. modifying signals or deteriorating the circuit, or can have the purpose of leaking confidential information. Hardware Trojans which leak confidential information as their sole payload have been characterized as especially dangerous, as they minimally change the overall system behavior [88]. For instance, a hardware Trojan was shown to be capable of inferring and leaking the secret key in an advanced encryption standard (AES) circuit implementation without directly probing it [89]. Even secure storage of the cipher key, e.g. in a physically unclonable function (PUF) which is highly volatile against physical modification or probing, would not prevent this indirect information leakage. An example Trojan which indirectly leaks the cipher key by tapping into the net containing the round key in the 'Add Round Key' phase of AES is shown in Figure 5.1.a). Ideally, the confidential information, in this case the cipher key, is dispersed over multiple operations as shown in Figure 5.1.b). As the information contained in the cipher key and all dependent instructions and values such as the round key is dispersed through multiple paths, the device does not expose a single point of vulnerability anymore and, therefore, is much less likely to be successfully infiltrated by a hardware Trojan. Although Trojans inserted in the design or manufacturing stage should ideally be detectable in pre- or post-silicon verification and testing respectively, the

complexity of state-of-the-art ICs make such exhaustive tests infeasible, leaving the need for alternative detection and prevention solutions.

As hardware Trojans that are inserted by a malicious manufacturer and indirectly leak security critical information are extremely difficult to detect after insertion, this chapter focuses on an HLS flow to prevent Trojan insertion and strongly increase detection probability where full prevention is not possible. This problem is especially significant for emerging devices in the IoT space, as they have to meet the highest security standards to gain consumer confidence and defense approval, but have to rely on external foundries due to their size and economic reasons.

An earlier version of this chapter appeared in [90], where we introduced a high-level synthesis flow for prevention of hardware Trojan insertion by an untrusted manufacturer. In this chapter, we extend the contributions of [90] through the following unique contributions:

- A threat-targeted high-level synthesis flow against Trojan insertion that can be adapted for a range of threat scenarios.
- A flexible metric to model the security against Trojan insertion attacks with the goal of information extraction.
- A targeted obfuscation scheme to camouflage the effect of information dispersion and mislead reverse engineering attempts.
- New experimental results that emphasize the proposed synthesis flow's ability to target specific threat parameters by achieve security metrics that exceed the baseline by a factor of at least 8.37 under constant resource constraints.

This chapter is structured as follows. Section 5.2 describes relevant background and related work. The threat model and goals of the adversary are described in section 5.3. Section 5.4 describes threat-targeted high-level synthesis, and introduces information dispersion and obfuscation as inherent parts of the synthesis flow. In section 5.5, we present an experimental evaluation. The chapter is concluded in section 5.6. This chapter was partially published in [90].

5.2 Background

5.2.1 Related Work

Design techniques for hardware Trojan defense can be categorized into three areas – mitigating against destructive Trojans through redundancy [91], improving detection

probability of Trojans [92], and increasing the insertion difficulty [93], e.g. by reducing rarely switching nets that are ideal candidates to trigger a Trojan payload. Synthesis flows to increase system reliability and avoid destructive Trojans by utilizing multiple different third party IPs (3PIPs) with the same functionality to detect deviation are studied in [91], [94]. Ben Hammouda et al. [95] have developed a technique to use ANSI-C assertions in HLS to automatically generate on-chip monitors (OCM) for verification of hardware accelerators, which has the primary application of increasing reliability. Trojans inserted in the manufacturing stage can be detected by analyzing the path-delay fingerprints [96], issuing targeted test patterns based on likely Trojan insertion points [97], or analyzing the side-channel emissions of a device-under-test [85], to name several recently proposed techniques. Although the contribution of detection is significant as it can deter Trojan insertion, it is not capable of actually preventing it. Multiple works have approached the problem of filling unused circuit area that could otherwise be exploited for Trojan insertion by a malicious manufacturer to increase insertion difficulty [98]. Xiao and Tehranipoor [92] have proposed built-in self-authentication (BISA), which employs functional filler cells that contribute to a digital signature. A secondary technique to prevent Trojan insertion is the minimization of the adversary's ability to identify nets that can be used to trigger the Trojan.

In [99], the authors describe efficient obfuscation through insertion of small obfuscation cells in conjunction with a PUF generated response to derive chip-dependent licenses. Chakraborty and Bhunia [100] describe the application of obfuscation against the insertion of Trojans by forcing the device to operate in a normal and an obfuscated mode, thereby expanding the reachable state space and hiding signal probabilities.

In contrast to existing work, we propose the first security optimized HLS flow that increases the difficulty of hardware Trojan insertion by a malicious foundry and therefore helps to prevent it. The proposed synthesis flow hides security critical information by dispersing it on the circuit, such that the malicious foundry would be required to insert a large number of Trojans in different circuit locations. As actual circuits are very large and contain an ever increasing number of nets, hiding the security critical information is a more feasible approach than attempting to minimize the adversary's ability to find suitable trigger signals by targeting rare switching nets. The proposed work can be combined with existing research in side-channel resistance and OCMs to achieve complete security through synthesis.

5.2.2 Need for Obfuscation

As described in the previous subsection, there exist a range of sophisticated and diverse defense mechanisms. However, the potential threat vectors to ICs are similarly diverse, as further described in section 5.3.3. Thus, it is very possible that the adversary gains precise understanding of the underlying circuit before attempting to insert Trojans. Moreover, a malicious party generally requires detailed understanding of the design as well as a detailed analysis of the control flow graph and state transition function prior to Trojan infiltration. The adversary needs to understand (i) which nets contain the desired secret information, and (ii) what the switching probabilities of local nets are, to identify a suitable rarely switching trigger for the Trojan. Therefore, obfuscation is an important technique to increase the difficulty of Trojan insertion by hiding the actual circuit behavior and switching probabilities. Additionally, the information dispersion flow that was previously introduced in [90] cannot achieve its full strength if the adversary can easily derive the full control-flow graph, as further described in section 5.4.4. However, conventional obfuscation flows do not integrate well with the proposed high-level synthesis flow, as the proposed information dispersion attempts to maximally utilize available resources. Moreover, such flows do not inherently consider the effects of information dispersion and thus will introduce obfuscation where it is ineffective and has no practical benefits. To mitigate this risk and further increase the security of the synthesized design, this chapter introduces obfuscation in co-optimization with information dispersion and is embedded in the synthesis.

5.2.3 Vulnerability Characteristics

The vulnerability of a circuit-level design to hardware Trojan insertion for information leakage by a malicious foundry can be characterized with three metrics: (i) the amount of available circuit area that can be used to insert Trojan payload, which includes area that is only artificially filled with dummy cells; (ii) the availability of rare-switching nets that can be used for Trojan activation; (iii) the availability of nets carrying the desired security critical signal. Whereas (ii) has been studied in the past, we present a fully automated flow in this chapter to simultaneously minimize (i) and (iii) such that the risk of Trojan insertion is drastically reduced.

5.3 Adversary Objective and Threat Model

The objective of the proposed flow is to defend the synthesized design against Trojans inserted by a malicious manufacturer. This defense is primarily concerned with Trojan resistance, i.e. increasing the required effort and resources that an adversary has to invest to inject a Trojan into the design. A secondary objective is the facilitation of Trojan detection if prevention was not successful. Therefore, the threat model targeted in this chapter is different from those commonly studied for hardware security [101]. The majority of proposed Trojan defenses are geared towards detecting them after insertion, for instance through comparison with a golden design, performing side-channel analysis, or targeted functional testing [102]. The prevention of manufacturer inserted Trojans that leak confidential information is a particularly difficult problem, as such Trojans do not noticeably change circuit signals. Due to the need to account for manufacturer and environment variations, runtime detection of such leakage Trojans is rarely possible.

5.3.1 Attack Goal

The adversary intends to insert Trojans into the device with the objectives of: (i) revealing confidential information which is not externally observable; (ii) revealing this information based on a deterministic (though not necessarily external) trigger signal; (iii) hiding this information leakage from detection by the trusted party. The leakage channel for this intentional information leakage is not further specified, though modulation of a wireless transmission as demonstrated in [103] is representative of the threat.

5.3.2 Adversary Capabilities

The malicious foundry is capable of inserting a Trojan into the layout provided by the trusted party, which includes tapping into existing signal routing for both the trigger and the payload of the Trojan. Due to the engineering difficulty and cost, as well as the increased probability of detection in testing, the adversary is assumed to be incapable of performing significant modifications to existing placement and routing in the layout, with the exception of the removal of transistors that are clearly detected as redundant (e.g. dummy cells). In line with the attack goal of avoiding detection, the adversary further has only limited routing capability.

Moreover, in addition to knowledge of the layout, the motivated adversary is assumed to have gained full access to the high-level design description of the fabricated hardware.

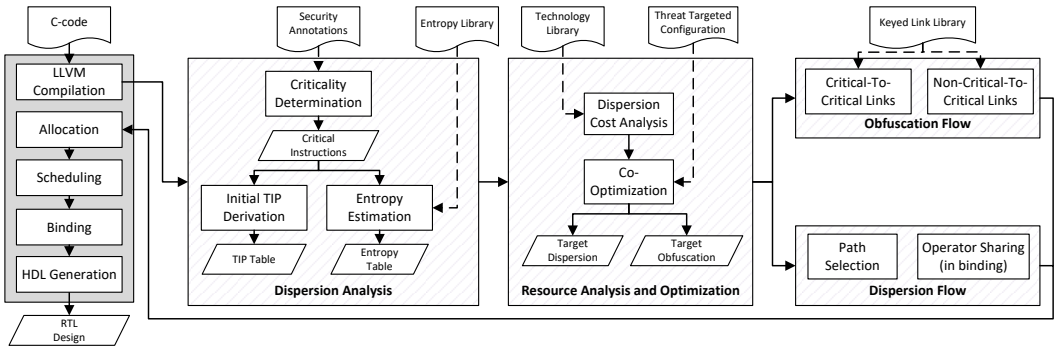

Figure 5.2 The threat-targeted high-level synthesis flow extends typical high-level synthesis steps with a dispersion analysis, resource analysis with optimization of security parameters, and obfuscation as well as dispersion flows.

As motivated earlier, this is possible through knowledge transfer, IP theft, or reverse engineering. For high-security applications in the military or for the widely connected IoT, security may not depend on the secrecy of the design. Thus, further scrambling of the circuit behavior is necessary to reduce the adversary's ability to gain deep insight into the device's actual functional behavior and control flow.

5.3.3 Diverse Threat Spectrum

Obfuscation solves the problem that all locations for Trojan insertion are apparent if the control flow can be fully analyzed and understood. Therefore, there are diverse threat situations. In one case, the primary threat is that an adversary could potentially perform a deep analysis of the available layout data in the manufacturing process, which leads to detailed understanding of control flow and security critical nets. In this situation, a high degree of obfuscation is very desirable to mitigate the adversary's ability to fully understand the underlying logic. This threat is referred to as the Analytical Threat in the following. A different threat situation exists if the secrecy of the exact control flow is less critical, for example when the manufacturing adversary is also expected to perform a significant share of the post-manufacturing validation and thus requires intimate understanding of the underlying device behavior. Similarly, it is also possible that the control flow is already highly complex or is protected with other countermeasures, such that reverse engineering and analysis are not primary concerns. Split-manufacturing is another technique where the analytical threat is reduced as the adversary has limited understanding of the full design. In such situations, the trusted party would emphasize the physical threat and therefore information dispersion over the analytical threat or increased obfuscation. To achieve a

synthesis flow that is widely applicable, the flow must seamlessly support defenses against the entire threat spectrum, ranging from an analytical threat to a primarily physical threat.

5.4 Threat-Targeted Synthesis

The proposed security-centric HLS flow is implemented in the LLVM compiler infrastructure [104]. At the core of LLVM is the intermediate representation (IR), which is a low-level programming language that can be considered to be a machine-independent assembly language. The overall flow is presented in Figure 5.2. The typical HLS flow consists of allocation for determination of available hardware units under consideration of constraints, scheduling to assign operations into clock cycles and to generate a finite state machine (FSM), and binding, which assigns operations to hardware units. These HLS steps are further described in sections 5.4.8 and 5.4.9. The flow extends these common steps by further steps for security optimization. First, a dispersion analysis derives key characteristics that allow highly resource efficient defense measures. Then, a resource analysis with co-optimization of security target parameters for dispersion and obfuscation is performed. Information dispersion is the key technique to ensure that the difficulty of Trojan insertion is greatly increased by dispersing security critical values across multiple operands, so that the adversary's ability to leak such information is greatly reduced. Information dispersion is complemented with obfuscation to hide the internal results of the dispersion flow and scramble the control flow to further reduce the adversary's ability to understand the design and derive adequate leakage payload or trigger signals.

First, the security critical instructions are derived by analyzing security annotations provided by design engineers. Then, the initial Trojan insertion points (TIPs) and instruction entropy are determined for each critical instruction. From these results, resource driven information dispersion is performed to introduce artificial dispersion for operators and registers to increase both the difficulty of Trojan insertion and the probability of Trojan detection. Through resource-targeted security optimization, the proposed flow inherently minimizes the available unused circuit area and removes the need for abundant dummy cell insertion.

5.4.1 Definitions and Notation

The term entropy, in general, is used to refer to the expected amount of information. For instance, in the case of error correction in PUF [82], this term refers to the amount of *security*

critical information specifically. Entropy of instruction i is denoted as $S_i \in [0,1]$. We define information decay κ as the degree to which a value loses entropy when combined with non-critical values. We denote the initial number of TIPs for instruction i to expose the original secret value (OSV) by τ_i. The degree of information dispersion d is defined as the minimum number of circuit signals to be simultaneously observed to guarantee full observation of the OSVs under consideration of entropy loss. The degree of obfuscation is denoted by σ. Vectors and mathematical sets are denoted by an uppercase character, e.g. the set of processed instructions V. Tables are denoted by an uppercase T, e.g. the entropy table for operand types T_S.

5.4.2 Dispersion Analysis

Through dispersion of critical information, the secret information is distributed among different physical paths and only one of the alternate paths processes the actual value. Therefore, the adversary is required to insert a Trojan into all locations to guarantee successful leakage of the critical information. As part of the automated synthesis flow, the individual critical paths are clock gated such that the overhead in power consumption is negligible and only the active path operates normally.

5.4.2.1 Identification of Critical Instructions

The starting point of the proposed flow is the high-level description of the functionality in C-code with (minimal) additional security annotation: (i) those functions or variables i_c that contain OSVs have to be annotated. (ii) the functions or variables that are explicitly not security critical i_{safe}, despite possibly having critical inputs. These annotations allow the synthesis to perform highly targeted operations. Note that the algorithm automatically determines all dependent instructions, and hence it is sufficient to specify only the initial occurrence of a critical value, which introduces minimal engineering overhead. To continue the example in Figure 5.1, specification of the cipher key as critical is sufficient, and the flow will automatically derive the XOR operations and the round keys are security critical as well. In addition to this, the designers may optionally specify that the cipher, which is the result of the encryption, should not be treated as critical and would therefore mark the outcome of the encryption method as *safe*. This helps to reduce the cost, but is largely already contained in the automatic entropy estimation. The proposed security flow iterates through the IR and resolves direct and indirect dependencies of the *critical* instructions

while terminating the exploration at the *safe* instructions. The outcome of this exploration is a complete list of every instruction that directly or indirectly depends on the OSVs, which includes memory accesses and registers.

5.4.2.2 Determination of Artificial Dispersion

To achieve the target dispersion d_t, which is derived as part of the dispersion and optimization co-optimization in section 5.4.5, the security optimization flow determines the required degree of artificial information dispersion $d_{A,i}$ for each critical instruction. It describes the required information dispersion such that the OSVs achieve an overall target dispersion d_T. It specifically describes the number of locations into which the original information content of instruction i has to be dispersed, and is derived from the entropy content S_i and the initial TIP count τ_i, which are further specified in the following sections. The artificial dispersion is derived as $d_{A,i} = \lceil d_T S_i - \tau_i \rceil$. Consider an example similar to Figure 5.1 of a highly secure IoT device where the cipher key has to be fully protected against information leakage, and where sufficient unused circuit area is available. The target dispersion could be selected to $d_t = 10$ such that an adversary would be required to insert Trojans into at least 10 circuit locations to extract the cipher key. For the XOR operation, the initial number of TIPs could be $\tau_i = 1.6$ due to exchange with non-critical operands, and the entropy could be reduced to $S_i = 0.3$ due to an upstream logical operator. In this case, $d_{A,i} = 2$, and the information contained in the XOR instructions has to be dispersed across two circuit locations. After determination of all artificial dispersion factors, the flow iterates through all critical instructions to disperse the value contained in each instruction

Algorithm 5.1 Derivation of the initial Trojan insertion points (TIPs) and of the entropy for critical instructions.

Constants:	
κ	Criticality decay
T_S	Entropy channel capacity for all operand types
w_m	Weight of the maximum parent entropy
Input:	
V	Instruction call chain
i	Critical instruction to be processed
Output:	
S_i	Entropy of instruction i
τ_i	Initial TIPs of instruction i

DeriveTIPs

$V = V \cup i$
If SAFE(i)
  $\tau_i = \infty$
Else If ISALLOCATION(i)
  V_c = GetStoresInto(i) ∪ GetPointersInto(i)
  $\tau_i = \min$ DERIVETIPS$(i_c, V), i_c \in V_c$
Else If NoCriticalParents(i)
  $\tau_i = 1$
Else
  V_p = GETPARENTS(i)
  $\kappa_{total} = \kappa \cdot |i_{nc} \in$ NOTCRITICAL$(V_p)|$
  $\tau_{min} = \min$ DERIVETIPS$(i_c, V), i_c \in$ CRITICAL(V_p)
  $\tau_i = \kappa_{total} + \tau_{min}$
$V = V \backslash i$

DeriveEntropy

$V = V \cup i$
If SAFE(i)
  $S_i = 0$
Else If NoCriticalParents(i)
  $S_i = 1$
Else
  V_p = GETPARENTS(i)
  If IsAllocation(i)
    $V_p = V_p \cup$ GETPOINTERSINTO(i)
  $V_{S,P}$ = DERIVEENTROPY$(i_c) T_S$[OPTYPE(i_c)], $i_c \in$ CRITICAL(V_p)
  $S_{max} = \max V_{S,P}$
  $S_i = \frac{S_{max} w_m + (1-w_m) \sum (V_{S,P} \backslash S_{max})}{(|V_{S,P}|-1)(1-w_m)+w_m}$
$V = V \backslash i$

by duplicating it by the $d_{A,i}$. In a second sweep, the operands in the instructions are updated to reference duplicated instructions of the corresponding path.

5.4.2.3 Derivation of Initial Trojan Insertion Points

As stated in section 5.2.3, the minimum number of Trojan insertion points (TIPs) for critical information leakage is a key characteristic for the prevention of Trojan insertion by the manufacturer. From the set of critical instructions, the TIPs are computed for each instruction per method DERIVETIPS in Algorithm 5.1. By default, an instruction has a TIP count τ of one, as an adversary can directly leak its output net or register to extract its secret

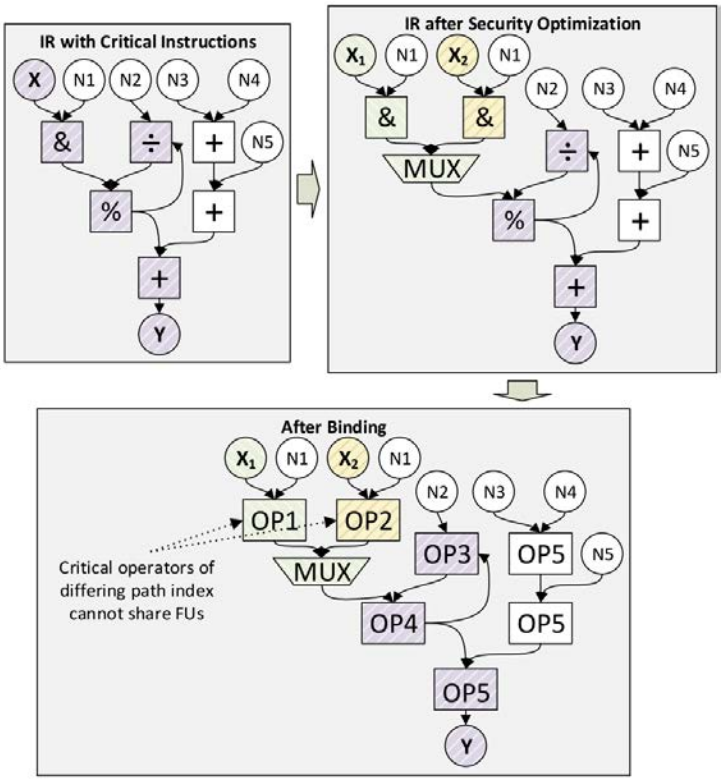

Figure 5.3 Example of targeted insertion of information dispersion. Due to modulo entropy loss, only upstream critical instructions are dispersed.

information. We introduce a variable criticality decay κ to characterize the loss of reproducibility when security critical values are combined with non-critical values. Consider the division operation in Figure 5.3, which depends on the values from the security critical modulo operator, and the (non-critical) constant value $N2$. The output of the modulo operation cannot directly be inferred from the output of the divider, unless certain knowledge about N2 exists, which is captured in the criticality decay κ. For the experimental evaluation, we use a constant criticality decay that was empirically set to $\kappa = 0.3$. The algorithm recursively visits instructions and derives their TIPs as the minimum of the TIP of the instructions' predecessors and subtracts the total criticality decay. We briefly discuss the edge cases in this approach. Circular dependencies, e.g. through loops, are broken by terminating the exploration before completing the circle by maintaining a chain of instructions that had already been visited.

Table 5.1 Information Capacity of operators for entropy estimation.

Operator	Information Capacity T_S
Equality Comparator ('==')	0.1
Comparator (e.g. '<')	0.15
Modulo ('%')	0.3
Logical exc. XOR ('&', '|')	0.3
Other (e.g. '+', '^')	1

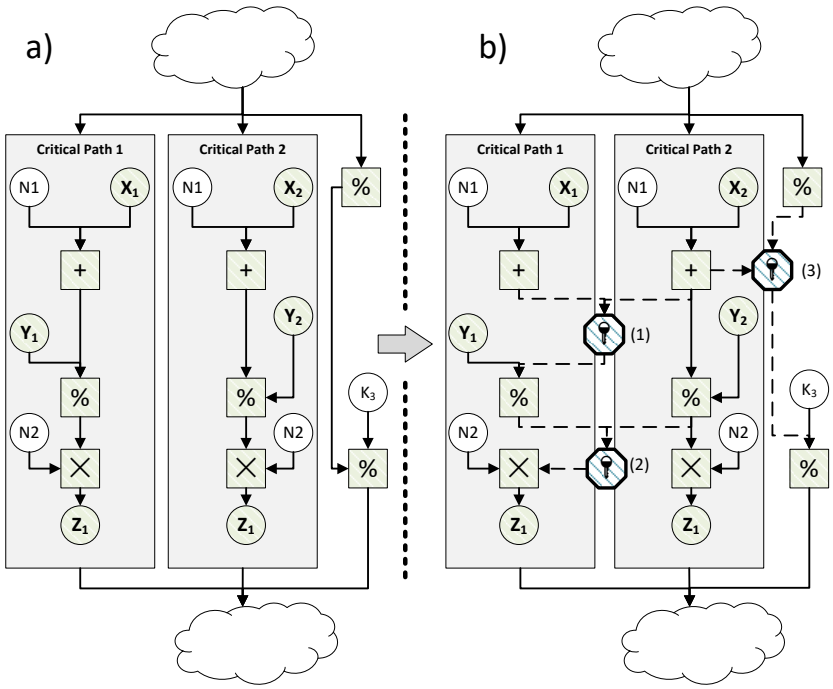

Figure 5.4. a) Dispersion leads to clear separation of critical paths that can be exploited by adversaries. b) Obfuscation introduces links among critical paths (operators (1) and (2)), and between critical and non-critical paths (operator (3)).

5.4.2.4 Consideration of Entropy Loss

For resource-efficient secret information dispersion, we consider not only the decay of criticality through exchange with non-critical operands, but also the information capacity of instructions. The high-level implementation is described in method DERIVEENTROPY of Algorithm 5.1. The underlying principle is the fact that each type of operator transmits different degrees of information, and therefore a variable amount of entropy is lost. The

entropy library used in our experiments is in Table 5.1 and shows the information capacity C of different operators that were empirically determined and can be adjusted in a trade-off between resource effectiveness and worst-case information loss upon successful Trojan insertion. To illustrate the need for empirical entropy estimation, consider the modulo operation: $x = c \; mod \; N$, where c is a security critical value. When only the values x and N are considered, it is not possible to directly determine the original value of c, though it is possible to derive an equation that describes all possible values of c. The outcome of this modulo operation does not contain as much security critical information as input c, and this entropy loss is conservatively modeled as 0.3 for this operand type.

5.4.3 Obfuscation to Defeat Reverse-Engineering Vulnerability

After information dispersion, security critical operations are performed in multiple, parallel paths such that an adversary is forced to introduce multiple Trojans to guarantee successful information leakage. However, due to the nature of the critical paths and to ensure a minimum number of TIPs, functional units are not shared across critical paths. This has the effect that simple information dispersion is reflected in clearly separated logical and physical operators for critical instructions. Thus, a motivated adversary can exploit this understanding to facilitate the identification of security critical instructions.

Therefore, we introduce an obfuscation scheme to (i) hide the actual transition probabilities, (ii) remove the clear separation of security critical paths, and (iii) more concisely quantify the threat and corresponding security solution.

An overview of the obfuscating operations is provided in Figure 5.4. It shows that the initial control flow graph exhibits a clear separation between individual critical paths, as well as between critical and non-critical paths. This can be exploited to identify security critical nets and can facilitate the infiltration of all security critical paths. The figure shows two types of obfuscating operations. We refer to links among critical paths as critical-to-critical (C2C) links; obfuscating operators (1) and (2) in the figure are such links. A link between a critical and non-critical (CNC) path is keyed operator (3). These links are inserted based on a keyed operator. The control flow after obfuscation will only follow the original flow if the device is operated with the correct key. Otherwise, a large number of random links between the critical paths is activated, which will lead to non-deterministic functional behavior as well as incorrect switching profiles. Moreover, adversaries would no longer be able to exploit clear separation of paths to insert Trojans into all critical paths.

5.4.4 Threat-Targeted Security Metric

Information dispersion limits the available free area and requires the adversary to simultaneously insert multiple Trojans to successfully leak security critical information. In previous work, the number of Trojan insertion points (TIPs) was the primary metric for the security of the synthesized design. With the introduction of obfuscation, it is apparent that TIPs can be complemented with the degree of obfuscation for a resource efficient security solution.

The metric for obfuscation has to adhere to several principles:

- Different designs should be comparable, such that similar values in the metric of different designs provide a similar level of obfuscation. This should include designs of considerable size and resource variations.
- Increasing the count of artificial links increases the difficulty of reverse engineering the information dispersion and should accordingly increase the value in the metric.
- Links among critical paths, and between security critical and non-critical paths have different purposes. Thus, application and threat dependent weighting must be possible.

From the aforementioned description, the metric for obfuscation is determined to $\sigma = \frac{2*(\alpha_1 L_{CNC}+\alpha_2 L_{CC})}{N_c}$, where N_c is the number of security critical instructions, and L_{CNC} and L_{CC} are the number of links between critical and non-critical paths and between critical and critical paths, respectively.

To describe the overall resilience against Trojan insertion by a malicious adversary in the manufacturing stage, the number of TIPs has to be considered in conjunction with the degree of obfuscation. This metric has to abide by the following constraints:

- Whereas increasing dispersion will linearly increase the security of the device, increasing obfuscation should yield diminishing returns based on the cost and security characteristics of the device.
- The offset of the weighting of obfuscation as well as the velocity of the diminishing return for obfuscation should be user-specified threat-parameters to control the targeting of the synthesis.

From these requirements, the combined security metric is quantified as

$$\psi = \frac{d_t\,(k_2 + \sigma)}{1 + k_1\,\sigma} \tag{5.1}$$

where k_1 and k_2 describe constant offsets that allow tuning the security optimization for specific threat scenarios. The former determines the velocity of diminishing return for obfuscation, and the latter determines the offset with regard to dispersion.

5.4.5 Resource Analysis and Optimization

The resource oriented co-optimization of information dispersion and obfuscation is initiated after an automated cost analysis of dispersion. In this cost analysis, the resource cost is divided among critical instructions and non-critical instructions. Then, the cost of increasing information dispersion is computed by considering the cost of adding new multiplexing units for shareable instructions, as well as the cost for new functional units where sharing is not a possibility. The resource cost of increasing dispersion is denoted with ΔC_{d_t}.

The cost of increasing the obfuscation metric is similarly computed by first deriving the cost for inserting the links from a library of known keyed links. The increase in resource cost due to increase of obfuscation can then be quantified as $\Delta \mathrm{C}_\sigma = \frac{C_{avg,link} * N_C}{2}$, where $C_{avg,link}$ is the average cost of inserting a link weighted by the relative insertion probability of the link, and N_C is the number of critical instructions. In combination with the metric for resiliency against Trojan defense in equation (5.1), this results in the equations:

Maximize

$$\psi = \frac{d_t\,(k_2 + \sigma)}{1 + k_1\,\sigma},$$

$$d_t * \Delta C_{d_t} + \sigma * \Delta \mathrm{C}_\sigma + R_{const} \leq R_{\max}$$

Algorithm 5.2. Obfuscation flow through manipulation of the LLVM IR after resource analysis and optimization.

Input:	
$BB.V_c$	Critical instructions for a given basic block
$BB.V_a$	All instructions for a given basic block
v_{obf}	Critical instruction to be processed
l_σ	The link target to achieve the desired σ value

Obfuscate

Repeatedly until $l_{count} = l_\sigma$
  For Each BasicBlock BB
    $M_C = \text{DEPENDENCYSORT}(BB.V_c)$
    $M_a = \text{DEPENDENCYSORT}(BB.V_a)$
    $i_t = M_c[\text{RAND}()]$
    $type = \text{RAND}() > \frac{\alpha_1}{\alpha_1+\alpha_2}$
    If $type = 1$
      $\text{M}_{C,S} = \{c_i \in M_C, \text{Path}(c_i) \neq \text{Path}(i_t)\}$
      $i_s = M_{c,S}[\text{Rand}(0, index(M_c, i_t))]$
    Else
      $i_s = M_a[\text{RAND}(0, index(M_c, i_t))]$
    If $\text{COMPATIBLE}(i_t, i_s)$
      $\text{INSERTLINK}(i_t, i_s)$
      $l_{count} += 1$

InsertLink - Select

$\text{i}_s = SelectInst(v_{obf}, v_t, v_s)$
$BB.Insert(i_s)$
For Each $\text{Parent}(i_t)$ i_p
  $\text{ReplaceOperand}(i_p, i_t, i_s)$

This constraint optimization problem in two variables (d_t and σ) can be solved to derive the following optimal values for the degree of information dispersion and obfuscation, respectively:

$$R = R_{max} - R_{const}$$

$$\Omega_1 = \Delta C_\tau^2 \Delta C_\sigma^2$$

$$\Omega_2 = \sqrt{-\Omega_1 k_1 k_2 + \Omega_1 - \Delta C_\tau^2 \Delta C_\sigma k_1^2 k_2 R + \Delta C_\tau^2 \Delta C_\sigma k_1 R}$$

$$d_t = \frac{\pm\Omega_2 + \Delta C_\tau \Delta C_\sigma + \Delta C_\tau k_1 R}{\Delta C_\tau^2 k_1} \tag{5.2}$$

$$\sigma = \frac{R - \Delta C_\tau \, \tau}{\Delta C_\sigma} \tag{5.3}$$

These values of d_t and σ are dynamically computed as part of the automated synthesis, and yield in a security-optimized threat-targeted solution.

5.4.6 Obfuscation Flow

The obfuscation flow is shown in Algorithm 5.2. When performing obfuscation in the control flow, an initialization step has to catalog non-critical and critical instructions to establish clear dependencies. These dependencies are critical for adequate insertion of links, as values cannot be arbitrarily consumed before they are initialized to achieve successful RTL synthesis. To achieve efficient obfuscation, the algorithm makes heavy use of randomization and therefore does not expose any patterns that could be exploited in a reverse engineering process.

Based on the weighting of C2C and CNC links that is established as a result of threat targeting, the respective count of links is statistically determined by employing a random number generator with corresponding thresholds. Then, the links are introduced into the LLVM intermediate representation by iterating over the basic blocks and randomly choosing a critical instruction that serves as a target, i_t. For C2C links, a corresponding source instruction i_S is randomly selected in a subset of critical instructions on a different critical path that appear earlier in the dependency order. For CNC links, the source instruction i_s is selected among all instructions in earlier dependency order that are not critical. The actual insertion of the link depends on the type of link chosen; for this chapter a keyed-select is employed. The obfuscation flow takes a list of obfuscation operations as its input, such that these operators match the existing device and so that the obfuscation flow achieves a high degree of flexibility to be applicable in varying scenarios. For the results presented in section 5.5, the obfuscation flow only utilized 'select' operations as shown in Algorithm 5.2. Similar to [100], an applied value is compared against a known key which can be device specific, as in [99]. If the supplied value matches the key, the original operand is passed through the select instruction – otherwise, an entirely different value is forwarded, thus scrambling the actual control-flow graph.

5.4.7 Dispersion Flow

Once the dispersion analysis as well as dispersion and obfuscation co-optimization are completed, the desired target dispersion d_t is known. From this target dispersion, the required amount of artificial dispersion is determined for each instruction as specified in

section 5.4.2.2. Then, the flow iterates through all critical instructions to disperse the value contained in each instruction by replicating it $d_{A,i}$ times. In a second sweep, the operands in the instructions are updated to reference duplicated instructions of the corresponding path. As a result of this action, instructions that previously had a low number of TIPs were artificially strengthened and achieve a higher number of TIPs to match the target dispersion.

The security optimized flow contains additional modifications. In scheduling, replicated critical instructions are forced into the same cycle so that the overall system performance is not degraded. As further described in section 5.4.9, this is inherently required to achieve security guarantees across critical paths.

We propose two alternate approaches to determine the active path and outline the differences. The first case is dynamic and truly random path activation; the second case is static path activation. In dynamic path activation, a true random number generator (TRNG) is used to determine at runtime which of the possible circuit paths should be taken. As the path is dynamic and selected at runtime, the adversary is not able to predetermine the net that will contain the critical information and hence has to tap into multiple nets. The disadvantage of this approach is that the adversary can statistically leak critical information in a subset of the operations when Trojans are inserted in only a subset of the required paths. The preferred approach is static path activation through a keyed physically unclonable function (PUF) after the device is received from the foundry. The benefits of this approach are: (i) testing and side-channel analysis can focus on a single path, allowing higher coverage; (ii) the adversary may not expect statistical information leakage if Trojans are introduced only into partial paths; (iii) a backdoor or Trojan is detected in one path after deployment, and recovery by removing the information leakage is possible by reprogramming the device to use a different path when a PUF type supporting this is employed [105]. This allows immediate mitigation if leakage is detected after deployment in critical scenarios. We emphasize that, independent of the method for path activation, only one security critical path is active for any given operation. Combined with clock-gating, this ensures that the power overhead of the proposed security optimization is negligible.

5.4.8 Security-Driven Allocation

To achieve security driven synthesis, we have modified the allocation to either target a given degree of dispersion d_T, or to achieve the highest degree of dispersion possible given a resource constraint. Both approaches are evaluated in the experimental evaluation, and the

latter technique to target a given resource limitation is of particular benefit as it simultaneously reduces the available unused area which could otherwise be used for Trojan insertion by a malicious manufacturer. The available unused area after this synthesis flow is minimal and can be complemented with limited dummy cell insertion. To model resource consumption, a technology library that maps operators and their bit-width to a normalized resource cost is loaded during synthesis. In our experiments, this table was derived as the gate equivalent cost of each module, using optimized modules where available. The resource computation further considers whether operator types are shareable or not, which is heavily architecture dependent [106] and is therefore also loaded from a configuration table. In allocation, the available hardware units are determined based on sharing within a security critical path and between critical and non-critical instructions. Different critical paths do not share instructions for security reasons. After deriving the set of critical instructions, resource constraints are modeled for the entire design by determining the cost of the non-critical instructions C_{NC} and the cost of critical instructions C_C that approximately linearly increases with the degree of dispersion.

5.4.9 Security-driven Binding

Binding is concerned with assigning instructions and variables to functional units and registers. Typically, it is driven by a cost-function to reduce hardware implementation cost or achieve better timing. To truly achieve dispersion of secure information, operations have to be bound to functional units with great care, such that the ability of an adversary to leak information by inserting Trojans that tap into fewer than d_T signals is not increased. In the weighted bipartite matching algorithm that is used for binding in the HLS implementation, the highest positive weight is given to instructions of the same critical path, and negative weights are given to instructions of different critical paths. Shareable output registers or operands are secondary characteristics considered in binding. Avoiding mismatches in functional unit assignment is critical, as the adversary may otherwise extract the secret values of multiple critical paths by inserting a Trojan that taps into a single signal. The result is shown in Figure 5.3. From an initial intermediate representation (IR), the critical instructions are determined, and after derivation of the target dispersion, information is dispersed among two paths. As the *And* operations have significant loss of entropy, the *modulo* operations and other downstream operations do not undergo any dispersion optimization. In binding, functional units are shared as much as possible, and a single Adder

is used for both critical and non-critical instructions. However, the *And* operators are not shared, as they correspond to different critical paths. Sharing between critical instructions of different critical paths would allow the adversary to simply tap into the output net of the shared operator and would therefore diminish the security benefits, and is hence forbidden.

5.5 Experimental Evaluation

The security flow presented in this work was implemented as an extension of the LegUp HLS tool [107], which builds on the modular design of the LLVM compiler framework. The implementation primarily consists of two passes – security preparation, and security optimization. In security preparation, the security annotation is understood and applied to the LLVM IR to identify critical instructions and their downstream users. In security optimization, the bulk of the security-relevant work is performed, which includes estimating the initial cost and deriving an achievable target dispersion, deriving the initially required tap counts and entropies, and introducing the artificial dispersion into the design. Additionally, existing LegUp code was modified in allocation, binding, and scheduling to adjust for the needs by the security-driven flow, as described in the last parts of section 5.4.

5.5.1 Benchmarks

We present the evaluation of the proposed flow against a subset of the CHStone benchmarks [108]. In addition to the common benchmarks that show the HLS characteristics for specialized modules, we introduce an IoT-specific benchmark. It computes the running average temperature and determines whether a person is present based on a door sensor. Additionally, the benchmark performs voice command recognition by cross correlating a received time series signal with a secret stored signal by performing a fast Fourier transform (FFT) and the inverse of it.

5.5.2 Analysis of Information Dispersion

For each benchmark, we specify the critical value as shown in Table 5.2. We quantitatively compare the proposed entropy-based security optimization against two different baseline approaches. As this is the first work that targets this problem, there is no previous solution that can serve as a comparison. However, we employ a modular defense approach that is the natural extension of synthesis efforts against destructive Trojans in [91], [94] as the first baseline. Here, security information dispersion is achieved by introducing

multiple IP modules and distributing the secret information across these modules. The targeted defense baseline is based on intermediate results of the proposed HLS flow and utilizes the fine-grained critical instructions to achieve security dispersion without taking full advantage of the entropy and TIP count analysis.

Table 5.2 shows that the proposed security optimization consumes on average 54.4% fewer resources compared to the modular baseline when the same degree of security is targeted. This resource consumption is in gate equivalent units from the technology library.

Table 5.2 Resource utilization in established benchmarks when a given security level is to be achieved. The proposed security optimized defense reduces the hardware implementation cost on average by 54.4% (Δ_1) and 26.3% (Δ_2) compared to the modular defense and targeted defense, respectively. Resource costs are reported in thousands.

Benchmark	**Critical Values**	**Disp. d_T**	**Modular Defense**	**Targeted Defense**		**Proposed Security Optimization**		
			Cost	**Cost**	**Δ_1**	**Cost**	**Δ_1**	**Δ_2**
AES	key	5	247.2	182.2	26.3%	104.5	57.7%	42.7%
SHA	sha_info_digest	5	76.8	48.0	37.5%	30.4	60.4%	36.7%
Blowfish	indata	5	109.2	79.8	27.0%	63.9	41.5%	19.9%
GSM	LARc	5	667.2	292.2	56.2%	236.2	64.6%	19.1%
Entropy Chain	key	5	68.3	68.3	0.0%	43.6	36.1%	36.1%
Smart-Sensor IoT	secret_voice, key	5	412.4	144.6	64.9%	139.1	66.3%	3.7%

Table 5.3 Evaluation of the ability to maximize the target security information dispersion d_t under resource constraints. The proposed flow achieves on average three times higher information dispersion than the modular defense baseline (Δ_1) and 41% higher dispersion than the targeted defense (Δ_2).

Benchmark	**Resource Target**	**Modular Defense**		**Targeted Defense**			**Proposed Security Optimization**			
		Cost	**d_t**	**Cost**	**d_t**	**Δ_1**	**Cost**	**d_t**	**Δ_1**	**Δ_2**
AES	125	98.9	2	115.8	3	50.00%	104.5	5	150%	66.67%
SHA	75	61.4	4	72.5	8	100.00%	72.8	14	250%	75.00%
Blowfish	100	87.4	4	94.2	6	50.00%	89.5	7	75%	16.67%
GSM	350	266.9	2	331.8	6	200.00%	346.3	8	300%	33.33%
Entropy Chain	100	95.6	7	95.6	7	0.00%	96.5	11	57%	57.14%
Smart-Sensor IoT	200	165	2	191.1	8	300.00%	185.5	8	300%	0.00%

Table 5.3 compares the three techniques under identical resource constraints, and the proposed flow achieved on average 188.69% higher security than the modular baseline. The effectiveness of the targeted defense, which is a subset of the proposed security optimization, is most pronounced when a module contains a significant share of instructions that do not have data dependencies to critical instructions. Here, the targeted approach allows dispersion of the information in only critical operations and therefore achieves significant gains, whereas the multiplication of the module, as done in e.g. [94] for higher reliability, would include multiplying such non-critical instructions. The true benefits of the fully optimized security flow show when numerous complex operations are performed on the critical information. The AES benchmark clearly demonstrates that the security optimized approach reduces the resource utilization by 66.67% compared to the targeted defense and by 150% compared to the modular defense. The cause for this significant improvement is in the complex and costly cryptographic operations performed within AES. Multiple modulo and divider operations are chained such that significant entropy loss is incurred, which is exploited by the proposed flow.

The ratio of security improvement to area overhead is four on average. We emphasize that the area overhead does not have a corresponding power overhead, as (i) a single critical path performs the critical operations and (ii) the other operations can be clock-gated. Furthermore, the area overhead is reported respective to a very small security component that is almost entirely security critical. Compared to a full chip, the overhead is minimal. Our experiments have shown that dedicating at most 10% of the circuit area of a Xilinx XC2V4000 FPGA to a security improved AES implementation [24] would allow increasing the dispersion target to 15. The real increase in Trojan defense is significantly higher, as the security enhancements make it very difficult for the adversary to find unused circuit area to insert this number of Trojans. Even if the adversary manages to insert and route the trigger signal to all of them despite these countermeasures, this would significantly increase the detection probability. The primary threat of hardware Trojans, their small footprint, is defeated.

5.5.3 Analysis of Threat-Targeted Synthesis

The ability to target a range of threats is very important in the diverse landscape of circuit design and manufacturing, as further explained in section 5.3.3. A comparison among three configurations is performed: information dispersion by itself, obfuscation with a primarily

physical threat, and obfuscation with a primarily analytical reverse-engineering threat. For the physical threat, the available levers use default values of $k_1 = 1$ and $k_2 = 0.1$ to construct ψ_1. In contrast to this metric configuration, the analytical threat emphasizes obfuscation stronger with $k_1 = 0.01$ and $k_2 = 0.01$ to form ψ_2. The first value k_1 is proportional to the negative effect of increasing obfuscation; reducing this value allows obfuscation to be applied in larger amounts to increase the overall security metric. Similarly, k_2 controls the offset in obfuscation and determines the importance of obfuscation – a smaller value of k_2 implies higher importance of obfuscation.

The results for threat-targeted synthesis are shown in Table 5.4 for two different configurations. The physical threat configuration ($k_1 = 1$,$k_2 = 0.1$) favors both dispersion and obfuscation, whereas the analytical threat configuration prefers obfuscation ($k_1 = 0.01$,

Table 5.4 Comparison of the threat-targeted synthesis under resource constraints for two different configurations. Resource costs are reported in thousands.

Benchmark	**Resource Target**	**Information Dispersion**		**Obfuscated – Physical Threat**			**Obfuscated – Analytical Threat**		
		Cost	$\boldsymbol{d_t}$	**Cost**	$\boldsymbol{d_t}$	$\boldsymbol{\sigma}$	**Cost**	$\boldsymbol{d_t}$	$\boldsymbol{\sigma}$
AES	125	104.4	5	123.1	5	9	123.2	3	30
SHA	75	72.8	14	74.2	11	14	74.6	6	41
Blowfish	100	89.4	7	99.7	7	16	99	5	55
GSM	350	346.3	8	344.7	7	3	343.4	4	12
Entropy Chain	100	96.5	11	99.1	10	71	99.8	8	185
Smart Sensor IoT	200	185.5	8	198.3	8	5	198	5	23

Table 5.5 Continued evaluation of the security metric from data presented in Table 5.4. Δ_1 and Δ_2 are the respective factors of improvement over pure information dispersion.

Benchmark	**Information Dispersion**		**Obfuscated – Physical Threat**			**Obfuscated – Analytical Threat**		
	$\boldsymbol{\psi_1}$	$\boldsymbol{\psi_2}$	$\boldsymbol{\psi_1}$	$\boldsymbol{\psi_2}$	$\boldsymbol{\Delta_1}$	$\boldsymbol{\psi_1}$	$\boldsymbol{\psi_2}$	$\boldsymbol{\Delta_2}$
AES	0.5	0.05	4.55	41.33	9.10	2.91	69.25	1385.08
SHA	1.40	0.14	10.34	135.18	7.39	5.87	174.51	1246.50
Blowfish	0.70	0.07	6.63	96.61	9.47	4.92	177.45	2535.02
GSM	0.80	0.08	5.43	20.46	6.78	3.72	42.89	536.16
Entropy Chain	1.10	0.11	9.88	415.26	8.98	7.96	519.33	4721.15
Smart Sensor IoT	0.8	0.08	6.80	38.17	8.50	4.81	93.54	1169.21

$k_2 = 0.01$). The degree of obfuscation σ is notably higher in the latter configuration, in exchange for a lower degree of information dispersion d_t. This evaluation is continued in Table 5.5, where the security metrics for each of the configurations are provided. It is notable that the two metrics are not exchangeable and only values of the same metric are comparable. For the physical threat configuration, it is notable that obfuscation increases the metric score in comparison to information dispersion by a factor 8.37 on average. This improvement comes at no resource cost, as the same resource limitations were applied for all configurations. In the analytical threat configuration, the improvement is even more pronounced, and the average improvement factor is 1932. As the primary target of the defenses in this configuration is to eliminate the adversary's ability to reverse engineer the control flow and determine the dispersed security critical operators, the introduction of obfuscation carries significant weight.

This improvement also shows when comparing the two obfuscated configurations. For the first metric, the physical threat configuration achieves an improvement of 46%. Similarly, the analytical threat configuration achieves a 77% higher value in the second metric.

5.6 Conclusion

Hardware Trojans are a significant threat for emerging devices that rely on the highest levels of security, due to the increasingly outsourced manufacturing. This chapter presented a security optimization flow that utilizes resources with high efficiency to identify and disperse security critical information through multiple operators and registers. Moreover, this work introduced an obfuscation flow that is embedded in the high-level synthesis flow and allows threat-targeted security optimization under resource constraints. Engineers merely need to define the initial security critical variable, and the downstream vulnerabilities are automatically detected and defended. The evaluation showed security enhancements with up to 5-times higher information dispersion and significantly higher Trojan insertion difficulty under the same resource constraints as a baseline technique. A threat-targeted evaluation showed that the co-optimization of obfuscation and dispersion can improve security by a factor between 8.37 and 1932, emphasizing the strength and flexibility of threat-targeted synthesis.

Due to the strong capabilities of the manufacturer, defending against Trojan insertion at this stage is very difficult. Circuitry that is inserted with defensive purposes may be removed

or manipulated by a well-educated and capable adversary. In addition, the problem of limiting intentional secret information leakage is of particular difficulty, as information can be leaked with minimal modifications to a circuit, which may not exhibit notable differences in power traces to be detectable. Therefore, we propose an entirely different approach to defend against this type of HTH. Our proposed approach simultaneously increases the difficulty of HTH insertion and probability of detection by dispersing secret values across a device such that it can be processed in one of multiple different locations. As the actual processing occurs in a dynamically and randomly selected path, the adversary is forced to insert an HTH into each of the possible locations. Thereby, the likelihood of detection is increased, as the Trojan payload is necessarily larger. Additionally, the link between the Trojan payload (the leakage) and the trigger will require more engineering work for customized locations on the device, or will have to travel a longer distance on the device. Finally, we propose to combine this methodology with resource utilization maximization, such that this defense security dispersion is employed to minimize the available empty circuit area. This increases the difficulty of inserting one HTH, let alone many, even further. To make this attack feasible, we propose to implement it as part of a high-level synthesis (HLS) framework, such that security critical information can automatically be detected and dispersed to meet resource utilization targets.

CHAPTER 6
HIGH-LEVEL SYNTHESIS FOR SIDE-CHANNEL DEFENSE

6.1 Introduction

Embedded devices are prevalent in every aspect of human life, a recent development that is only exasperated by the emergence of the Internet of Things (IoT) and cloud computing. In IoT, a large number of devices, buildings, vehicles, and sensors are interconnected to form a network that can gather and process data and respond to it. As part of emerging cyber-physical systems, these devices cover a wide range of applications from home automation in residential buildings to the coherent operation and control of the smart-grid. An array of military applications has also been proposed and tested, for example wireless sensor-network monitoring of borders and demilitarized zones [109], [110]. The public and private cloud are employed to control billions of IoT devices and analyze a massive and perpetual stream of sensor data. However, cloud computing is not limited to its IoT applications – there is rapid development and adaptation in industrial, marketing, and financial segments.

Today, security is considered to be one of the most significant obstacles to both IoT and cloud computing [109]. The security threats to IoT and cyber-physical systems are as numerous as their applications: There are concerns about privacy in home automation, resiliency in smart-grids, and confidentiality in defense applications. One of the concerns for IoT is the need for low cost, rapid development, and the lack of standardization or control, which dramatically increases the likelihood of security flaws. The security concerns for cloud computing are different but equally important: while secure data transfer between the data source and the cloud provider is a largely solved problem, extracting insight through analytical queries over encrypted data is very limited. This means that data has to be available in plaintext for meaningful analysis. Although many enterprise customers would prefer to outsource processing of confidential and sensitive data, this limitation on plaintext is an important blocker, since they cannot allow cloud providers to operate on plaintext. FPGAs with customer-supplied programming can be a solution to this.

For both cloud and IoT, FPGAs provide excellent solutions with scalability, maintainability, cost, and efficiency. Various proposals for FPGA-friendly protocols and security architectures have been made for these emerging fields [111], [112]. High-level synthesis (HLS) enables developers and designers to synthesize a low-level hardware description from a high-level system specification in widely known programming languages such as C. As such, HLS is a primary contributor to wider enablement of FPGA devices.

Although basic security practices such as encryption of network traffic or proper selection of encryption algorithms are well-known and incorporated in many designs, hardware security concerns are commonly overlooked. Moreover, countermeasures typically require low-level understanding and fine-grained cost balancing [113]. The hardware implementation is a significant source of information leakage which is often not considered when selecting higher-level algorithms or security techniques. This information leakage can be exploited through side-channel attacks, which perform statistical analysis to extract confidential values through the side-channel leakage. For example, when the execution of operations depends on any confidential value, the presence or absence of operations in a captured power trace can reveal information about the confidential values through a simple power analysis (SPA). Moreover, the hardware implementation of even the simplest logic operations is typically very susceptible to information leakage, as dynamic power consumption is dependent on the number of switching bits, capacitances, and a number of other identifying characteristics. Therefore, input and output values can be retrieved through differential power analysis (DPA). Numerous countermeasures against side-channel attacks have been proposed at the algorithmic and at the gate- or layout-level. Most of the lower-level techniques against side-channel attacks are highly resource and power intensive, as they attempt to achieve a constant, input-independent power drain. Moreover, these countermeasures cannot be applied efficiently without expert-level understanding of hardware security as well as deep familiarity with the circuit and potentially vulnerable operations.

In this chapter, we propose the first fully automated high-level synthesis flow with the primary target of minimizing side-channel information leakage in defense of DPA. This contribution enables developers and design engineers to efficiently address side-channel leakage concerns in a practical manner: specifying confidential variables in addition to the high-level specification to be synthesized is sufficient for automatic analysis of leakage and scalable injection of countermeasures. The unique contributions of this chapter are:

- High-level side-channel leakage characterization through derivation of per-operation confidentiality and cycle-accurate simulations.
- First side-channel leakage resistant high-level synthesis flow. Minimal annotations in addition to high-level C-code are sufficient for automated leakage analysis and insertion of countermeasures. This flow can target resource-constraints as well as an allowable leakage threshold.
- Automated detection and mitigation against branch imbalances that otherwise enable simple power attacks.
- Experimental evaluation with established CHStone benchmarks and a custom IoT benchmark. The proposed flow achieves up to 81% better leakage reduction than the baseline under identical resource constraints.

The remainder of this chapter is structured as follows. In section 6.2 we introduce relevant background. The high-level synthesis flow is presented in 6.3 and experimentally evaluated in section 6.4. A conclusion with outlook is provided in section 6.5.

6.2 Background

6.2.1 Related Work

Recent research found that HLS can provide unique benefits to the design of secure systems. Reliability can be increased through automatically generated on-chip monitors (OCMs) [95]. Reliability in the face of destructive hardware Trojans can be achieved through selective module selection as part of synthesis [91], [94]. HLS can also be applied to detect vulnerable operations with granular insertion of hardware Trojan defenses through information dispersion [90]. However, this is the first work to explore the benefits of HLS in securing designs against side-channel leakage.

The threat of side-channel leakage has received significant attention with the introduction of DPA by Kocher et al. [114]. DPA is a side-channel attack that enables the extraction of secret keys through signal processing over a large number of power traces. It can reveal the internal secrets of cryptographically secure algorithms such as the advanced encryption standard (AES) [115]. While introduction of random noise effectively reduces the signal-to-noise ratio, it has been shown that arbitrary noise does not provide significant security benefits and cannot efficiently hinder exploitation of side-channel leakage [116]. Masking was proposed to reduce the correlation between captured power traces and the

actual underlying data. Masked-AND was an early proposal for generally applicable logic design to secure AES [117]. Several techniques have been proposed to equalize the dynamic power consumption of digital circuits to reduce side-channel leakage. In dynamic differential logic (DDL), the correlation between the power consumption of the circuit and processed input signals is reduced by adding the complement of a circuit. Wave dynamic differential logic (WDDL) uses standard building blocks to form secure compound gates which can be applied in a regular ASIC or FPGA design flow in place of standard cells [118]. WDDL aims to consistently consume power by combining standard cell gates such that both the positive and negative outputs are computed. In precharge, all inputs are set to 0 such that outputs evaluate to 0. Therefore, as a result of the evaluation phase, there is one transition per output bit – either in the positive or in the negative output. This provides for consistent dynamic power consumption. Leakage may still occur due to timing and load capacitance variations. Simple dynamic differential logic (SDDL) [118], [119] operates similar to WDDL and is derived by applying De Morgan's law and AND-ing the differential output with the precharge signal. SDDL cannot guarantee only one switching signal per clock cycle, and therefore is inferior to WDDL. Even though such logic styles reduced leakage, they did not eliminate it due to routing and load imbalances. To address this concern, duplication of fully routed circuits with switched positive and negative input signals was proposed for WDDL as Double WDDL (DWWDL) [120] and other logic styles [121].

Manual application of DDL on a subset of the circuit has been proposed to achieve reduced overhead in Partial DDL [122] while maintaining leakage resistance. Resource cost reductions of 24% were reported for AES.

The previously described techniques and architectures require a specific security-centric skillset for efficient and effective application. While full application of an advanced logic style provides meaningful security enhancements, it is extremely costly in terms of resource consumption. Hence, an automated HLS flow can be very resource effective by introducing specific countermeasures where they provide the most benefit while also reducing the need for constant security-engineering guidance.

6.2.2 FPGAs in Emerging Applications

6.2.2.1 Cloud Computing

Due to their unique customizability, FPGAs expose a much more defined surface area for attacks and are therefore a valuable building block in establishing trust in the emerging cloud computing environment. It is generally undesirable to fully trust the cloud provider both from a security standpoint from the customer's perspective, and from a liability perspective from the cloud provider's standpoint [111]. However, complex analysis and computation tasks often require direct processing of plaintext data, hindering full adoption of the cloud. It was proposed to offload data analysis and processing tasks for highly sensitive data to dedicated FPGA units which perform custom operations specified by the cloud customer. Specific applications include operating on personally identifiable information (PII) in healthcare data [111] and privacy preserving map reduce [123]. These FPGAs externally consume and produce encrypted data, such that the cloud provider is not directly exposed to the highly sensitive plaintext data. For this operation, the FPGA is programmed with encrypted bitstreams containing secret keys, which allow decryption as well as encryption of data, thereby removing the requirement for full trust of the cloud provider.

In this application of FPGAs in the cloud, it is important to note that network-level information assurance regarding confidentiality is achieved. However, the cloud provider maintains physical access, particularly for maintenance and operation of the FPGA devices. Therefore, extraction of the secret keys for decryption of the available data, as well as direct extraction of the plaintext from the physical device, must be considered.

6.2.2.2 Internet of Things

In IoT, many physical devices, sensors, buildings, and vehicles are interconnected to co-compute and co-operate almost all aspects of modern society. As a mass product which is often powered exclusively by batteries, power and resource efficiency are crucial considerations in IoT design. Due to their configurability to perform specific tasks very well and their cost efficiency compared to ASICs, FPGAs are well suited for many IoT applications. They have been shown to surpass ASICs in reliability, cost, time-to-market, and maintenance [124]. Dynamic reconfigurability of FPGAs is a relatively recent development that has sparked new IoT specific architectures and applications [112].

The emergence of IoT has already revealed numerous security problems, from plaintext network traffic that can easily be intercepted and read, to an IoT-based botnet that culminated in a massive DDoS attack that lead to widespread network outage on the U.S. east coast [125].

6.2.3 Leakage through Conditional Operations

One source of information leakage is derived from conditional operations whose execution depends on variables with confidential content. This type of leakage can be exploited without extensive computational analysis as part of an SPA attack, as the difference in operations following such a branching statement potentially reveals information about the confidential variable.

This vulnerability was very common in early implementations of the RSA algorithm [116], which used the text-book implementation of the square-and-multiply algorithm. In this algorithm, an exponentiation of the form x^n can be restated as $x * (x^2)^{\frac{n-1}{2}}$ when n is an odd number, or $(x^2)^{\frac{n}{2}}$ when n is even. As the operation for an odd exponent requires an additional multiplication with sufficiently different power signature, an adversary can determine each bit of the private key in a step-by-step attack. Attacks of this kind can be defeated by rearchitecting the implementation of algorithms to eliminate the dependency between executed operations and confidential values.

As the vulnerabilities in RSA implementations show, oversight of side-channel implication is a common problem. Especially in IoT applications which constantly process privacy and confidentiality critical information, these countermeasures are of critical importance, yet practical considerations and the lack of automated defense mechanisms virtually guarantee that many implementations will suffer from similar vulnerabilities.

6.2.4 High-level Leakage Estimation

The strength of side-channel leakage is measured in the number of measurements to disclosure (MTD) for well-known and well-studied circuits such as AES, which allows comparison of results across different studies [126]. As an estimate of dynamic power consumption and hence side-channel leakage, the Hamming distance is widely employed [127]. Menichelli et al. [128] describe how the difficulty with high-level power simulation (and hence, side-channel leakage) is its focus on evaluating average power consumption. Even where cycle accuracy is possible, such power estimates provide little insight into side-

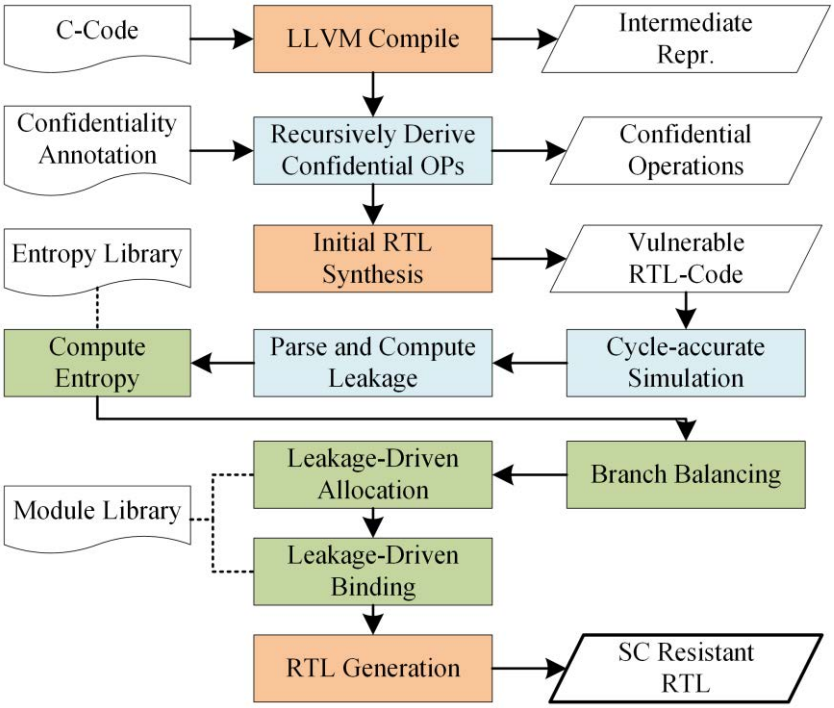

Figure 6.1 Overview of the side-channel leakage optimized synthesis flow. The flow combines typical HLS flows (orange) with analysis (blue) and culminates in leakage minimization operations (green).

channel leakage, as the absolute value of power consumption is not closely related to the actual signal-correlated power consumption. For accurate estimation, it was shown that only the internal logic values that correspond to the signal should be tracked. Hamming distance as the primary source for leakage estimation has successfully been used to analyze smartcard software for side-channel leakage [129].

6.2.5 Attack Goal and Involved Parties

The explicit attack goal of the adversary studied in this chapter is to reveal one of the user-specified secrets through side-channel analysis by evaluating the power trace. For this purpose, the adversary is assumed to have sufficient physical access to the device to measure and extract a large number of power traces. The primary focus of this chapter is the defense against side-channel analysis; therefore, we reference existing work as described in the background for defense against hardware Trojans or physical tampering.

As the nature of this chapter targets side-channel information leakage in active device operation, the designer, programmer, and end-user with confidential information are

together considered to be the first party. The goal of the first party is to produce a device that meets side-channel leakage requirements with highest resource efficiency, such that the majority of circuitry and computing power can be focused on the primary task of the device, e.g. cloud computation or IoT application.

The application environment of the device is assumed to be hostile – be it in the open space for IoT applications, or in the datacenter of an untrusted cloud provider. This differentiates the security needs from those of typical network security. Due to the hostile nature, any adversary is assumed to have physical access to the device to extract power traces in large numbers. However, physical tampering of any type, or malicious reprogramming of the device, is outside of the scope of this chapter.

The hardware vendor (i.e. FPGA manufacturer) or third party IP providers are an unrelated third party that is assumed to be neutral, and malicious intervention by this third party is outside of the scope of this chapter.

6.3 Synthesis Flow

The overall high-level synthesis flow for side-channel leakage minimization is shown in Figure 6.1. It consists of three primary phases: the initial synthesis, leakage characterization, and the final security synthesis.

In the initial synthesis, C-code is compiled into an intermediate representation (IR). User-specified annotations are employed to derive all operations that act on confidential information. The output values of such operations are also treated as confidential. The initial synthesis culminates in an RTL synthesis which produces vulnerable RTL-code.

In the leakage characterization phase, the RTL-code is simulated and the leakage of all operations which act on confidential information is computed from the simulation results in combination with entropy estimation for each operation.

The final security synthesis selects the appropriate module for each operation based on available resource and estimated side-channel leakage. Additionally, branch balancing is performed to reduce the vulnerability due to conditionals which depend on confidential information.

6.3.1 Initial Synthesis

The initial synthesis flow consumes user-provided C-code and compiles it into LLVM's intermediate representation (IR). This is an assembly-like language which is machine-

independent. This compilation phase already includes code optimizations provided by the LLVM compiler framework.

In addition to the code, the proposed synthesis framework consumes high-level annotations of variables to be treated as confidential. Such variables can include secret keys or authentication tokens, but can be used more widely in the IoT content to elevate the security treatment of user data such as the number of active operators and similar information.

These high-level annotations are utilized to automatically derive all operations which act on confidential information. Here, all outputs of confidential operations are treated as

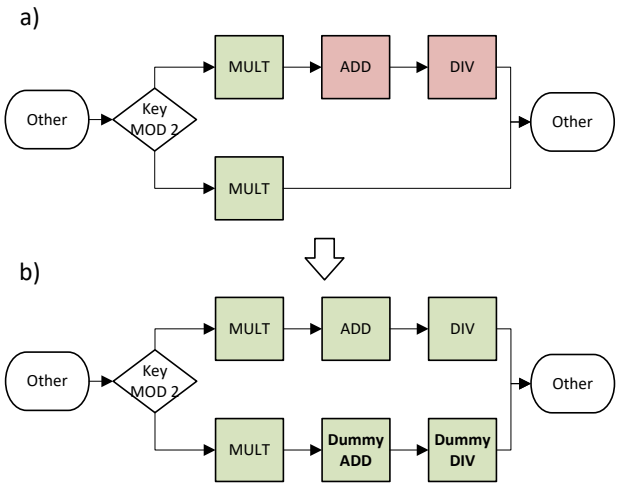

Figure 6.2 Example of branch balancing. One branch of a conditional statement is supplemented with dummy instructions to minimize information leakage.

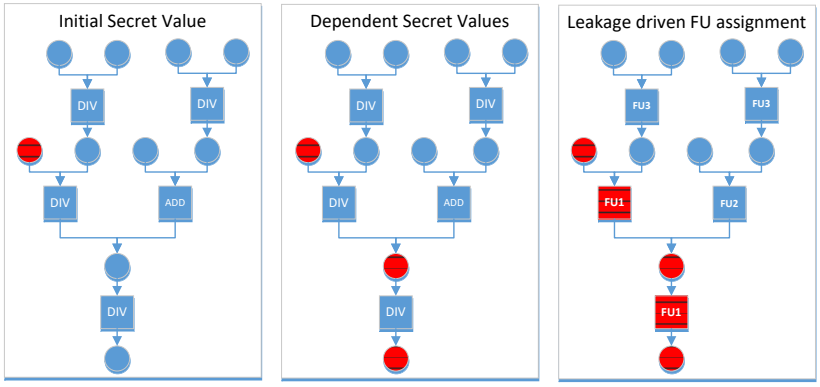

Figure 6.3 Example of leakage-driven binding. High risk operations are bound against one FU, while low risk instructions are bound against a different FU.

confidential as well. Therefore, minimal user input is sufficient to determine a graph of confidential operations. This graph of confidential operations is then employed to create targeted simulation directions which improve its time and space requirements.

The initial synthesis flow concludes with conventional steps of RTL synthesis, which are focused on resource utilization and device speed. As a result, the generated RTL-code is vulnerable to side-channel attacks and does not contain any countermeasures. This is a valuable starting point for deeper analysis of operations which are prone to leakage.

6.3.2 Leakage Characterization

As the primary metric for side-channel leakage of a given signal, the proposed flow employs the Hamming distance for switching operations. This is a commonly used metric [127], as dynamic switching operations are a primary source of power consumption for modern FPGAs and integrated circuits in general. In addition to the significant leakage through dynamic power consumption, it has been shown that side-channel leakage can result from static power consumption as well, which the proposed flow can consider with minor modifications.

For a high-level security analysis and defense against side-channel information, an overall metric of similar abstraction level is required. In the leakage characterization phase, the previously generated RTL-code is simulated to identify switching information for every operation. For this purpose, the proposed flow automatically generates simulation scripts that feed into professional simulation tools, and parses the simulation outputs for further processing. Only those operations that act on confidential information are relevant to the security optimization, and switching characteristics of other operations are discarded.

The leakage is primarily derived from the Hamming distance of a switching operation. As an estimate for dynamic power consumption and therefore leakage power, the Hamming distance of operation i is defined as $HD(i) = \left|i_{out,1} - i_{out,0}\right|$.

In addition to estimating the side-channel leakage through the Hamming distance, the proposed flow also considers the confidential information content through entropy estimation. The previous initial synthesis phase determined all operations and variable values that are recursively dependent on user-specified confidential values. Resource efficient synthesis requires further consideration of the degree of confidentiality. For instance, the result y of the operation $y = x \geq 25$ by itself has a comparatively smaller amount of confidential information than the secret key x. This reduction in confidential

information content is reflected in the entropy estimation. Recursively, the entropy of each instruction i is computed by multiplying the entropy factor of the functional type σ with the sum of input entropies of each parent operation:

$$h(i) = \sigma \cdot \sum_{p_i \in Parent(i)} h(p_i)$$

The overall leakage of a given instruction regarding the user specified secrets is thus specified as $\gamma_i = h(i) \cdot HD(i)$

6.3.3 Security Synthesis

6.3.3.1 Branch Balancing

Branch balancing is a problem that can be solved through algorithm-level modifications by the design engineer, as well as through automated mechanisms. A potential downside of automated solutions is that they can be resource intensive in both power and footprint. Therefore, the proposed flow generates detailed reports on the detected imbalanced branches to allow manual mitigation in addition to implementing an automated solution.

In branch balancing, all conditional statements of the entire synthesized device are searched for security sensitive information content. This encompasses the derived confidential operations discussed previously. Any such conditional statement triggers a code path for logging of a potential security breach as well as detailed analysis of the subsequent instructions. As discussed previously, any deviation between the subsequent paths can cause considerable side-channel leakage; therefore, the number of deviating operations is computed. An automated mitigation is attempted by inserting dummy operations with similar functional types to counterbalance the deviations.

In Figure 6.2, an example of branch balancing is shown. In Figure 6.2.a), the branches are imbalanced, as the upper branch contains additional ADD and DIV operations. These are balanced through dummy ADD and DIV operations in the lower branch, which counteract the initial imbalance.

6.3.3.2 Leakage-Driven Allocation and Binding

For simple and efficient security mechanisms, the synthesis must automatically determine the security requirements as well as applicable countermeasures for each instruction and functional unit. In a typical HLS flow, the binding step is responsible for

assigning instructions or operations to functional units and variables to registers. Binding algorithms are primarily guided by cost and timing concerns. When multiple operations bind against the same functional unit, large multiplexers are introduced to enable sharing. Due to the high cost of large multiplexers in FPGAs, only the most expensive operations are suitable for sharing [107], such as dividers or modulo operations.

In the proposed flow, allocation and binding are interweaved to maximize the efficiency of side-channel leakage reduction. The binding algorithm determines the assignment of IR instructions to functional units. At this stage, these functional units are the most basic and resource efficient implementations for a given function. As part of binding, a notable security enhancement is introduced: high risk operations (HROs) are assigned to the same FUs, whereas low risk instructions (LROs) are assigned to other FUs. This sharing pattern is shown in Figure 6.3 and is utilized in the allocation of more expensive side-channel countermeasures. The binding algorithm is driven by a weighted bipartite matching algorithm, and the proposed weights between operations are determined from the estimated leakage level.

After all operations are assigned to FUs, the leakage per FU is computed as a sum of the individual leakage terms. This is a conservative approximation, as partial correlation between the underlying signals is common. This estimation enables weighted module selection for each FU. Using the module assignment vector M, the vector of leakage estimates γ, and the total leakage metric θ, this problem can be described as a linear programming problem:

$$\text{Minimize: } \theta = \gamma \times M,$$

$$M \times C \leq r$$

Through this formulation, functional units with high leakage potential are upsized in terms of defense mechanisms, such that the overall leakage for a given resource utilization is minimized.

Table 6.1 Countermeasures employed in the evaluation. Resource overhead and effect on leakage are component-dependent – general estimates are provided.

Logic Style	Resource Overhead	Leakage Factor
Base	1	1
SDDL	4	1/7
WDDL	5.3	1/10
DWDDL	10.6	1/20
DAWDDL	13.1	1/25

Table 6.2 Evaluation of the side-channel optimized synthesis. Compared to the baseline, leakage is reduced between 32% and 72% ($\%_B$). Compared to the modular synthesis, leakage is reduced by up to 38% ($\%_M$).

Benchmark	Res. Target	Balancing Cost	Baseline		Modular Synthesis			Proposed SC Synthesis			
			Cost	θ_B	Cost	θ_M	%	Cost	θ	$\%_B$	$\%_M$
AES	187.0	0.6	187.0	394.7	187.0	323.8	0.18	186.9	202.0	0.49	0.38
Blowfish	88.0	0	87.4	341.3	87.6	239.3	0.30	88.0	231.6	0.32	0.03
SHA	62.0	0	61.4	99.4	60.2	64.6	0.35	61.9	49.3	0.50	0.24
SIMON	25.0	0	24.3	80.4	24.9	77.6	0.03	25.0	54.8	0.32	0.29
GSM	59.0	57.5	591.3	271.8	591.9	81.4	0.70	592.0	79.4	0.71	0.03
IoT	258.0	.5	257.9	305.3	***240.3***	85.5	0.72	***224.8***	85.5	0.72	0.00

6.4 Experimental Evaluation

The proposed side-channel information leakage optimized HLS flow was implemented based on LLVM and the LegUp HLS tool [107]. ModelSim was employed for the simulations, and Altera Quartus II was utilized for synthesis of generated Verilog into a Cyclone V FPGA. The proposed flow is evaluated in multiple benchmarks and with regard to different characteristics. Several benchmarks are adapted from the CHStone benchmarks [108]: AES, Blowfish, SHA, and GSM. AES [130] and Blowfish [131] are symmetric-key block ciphers, and SHA is a secure hashing algorithm. The GSM benchmark is an implementation of linear predictive coding analysis for the global system for mobile communications. Additionally, we have adapted the reference implementation of SIMON [132] as a benchmark, as it is a lightweight block cipher publicly proposed by the US National Security Agency's (NSA) Research Directorate with focus on efficient hardware implementation. SIMON is particularly interesting, as it has the expressed purpose of

Table 6.3 Evaluation of the proposed side-channel optimized synthesis against a modular baseline under more stringent resource constraints. Leakage is reduced by 20% to 40%.

Bench-mark	**Res. Target**	**Modular Synthesis**		**Proposed SC Synthesis**		
		Cost	$\boldsymbol{\theta_M}$	**Cost**	**θ**	**%**
AES	75.0	75.0	2052.5	75.0	1221.4	0.40
Blowfish	35.0	34.6	2217.9	35.0	1691.8	0.24
SHA	23.0	18.6	592.0	23.0	361.4	0.39
SIMON	9.5	8.1	505.1	9.4	395.3	0.22
GSM	235.0	235.0	763.0	235.0	536.0	0.30
IoT	75.0	74.7	1611.1	74.8	1079.2	0.33

facilitating security for IoT [133]. It is notable that these benchmarks implement a very specific functionality, which does not necessarily reflect the reality of embedded system applications in cloud or IoT environments. Therefore, we further created an IoT benchmark which performs several general-purpose tasks including Fourier transformations for voice recognition, temperature computation with corresponding light control, and presence detection with control of a door lock.

6.4.1 Evaluation Baseline

As baseline for the evaluation, the entire device is created using a side-channel resistant design style. This reflects today's design flows consisting of largely uniform technology and IP libraries. A second baseline is presented as the modular baseline. In this scenario, the proposed side-channel synthesis is applied at a modular level to represent the modular granularity of design engineering for large ICs in enterprises. Here, each module is selected to be implemented in one of the logic styles presented in Table 6.1 to achieve highest leakage resistance. We refer to section 6.2 and references for further background on cost and leakage evaluation of SDDL [118], [119], WDDL [119], [121], [134], DWDDL [121], DAWDDL [121].

6.4.2 Resource Targeting

In Table 6.2, the benchmarks were synthesized with a resource target. Costs in bold italics indicate that the maximum available countermeasures were applied – further reduction of leakage (θ) was not possible despite available resources. The resource target was selected such that the baseline could be fully implemented in the most resource efficient

logic style with reduced side-channel leakage, which is SDDL. The strong improvements of the proposed side-channel synthesis are clear; leakage is reduced by up to 72% compared to the baseline, and up to 38% compared to the modular baseline.

The synthesis of the GSM benchmark provided interesting results, as both the proposed synthesis and the modular baseline provide similar results and are significantly better than the full-device baseline. This can be explained by the complexity of the GSM benchmark: in comparison to the other CHStone benchmarks, this benchmark circuit performs more computations, many of which are independent of the user-specified confidential variables. Therefore, both the modular baseline and the proposed synthesis can be significantly more resource efficient and therefore achieve higher leakage reduction. Additionally, it can be observed that many of the confidential values (and their dependencies) are constrained to specific modules (basic blocks), which explains why the proposed synthesis was not significantly more efficient than the modular baseline.

In contrast to the GSM benchmark, the experimental results for SIMON show limited improvements in the modular baseline, but significant improvements in the proposed synthesis compared to both the full-device and the modular baseline. This can also be explained by the underlying design: The share of confidential operands is roughly uniform across the modules in SIMON; therefore, the modular baseline cannot significantly outperform the baseline. However, the proposed synthesis can perform leakage optimization down to the instruction or functional unit level, and is therefore able to extract further leakage reduction.

In addition to the resource-targeted evaluation in Table 6.2, the proposed side-channel optimized synthesis flow is further evaluated against the modular baseline with a significantly lower resource limit. For this evaluation, the simple baseline would not yield a result, as the low resource limit prohibits application of the side-channel reducing logic styles at the full-device level. The evaluation shown in Table 6.3 demonstrates the strength of the proposed flow under restrictive resource constraints, yielding between 22% and 40% reduced leakage as compared to the modular approach. Notably, the proposed flow achieved only a minor improvement of 3% compared to the modular approach under higher resource constraints for GSM, but achieves a leakage reduction of 30% for the same benchmark in a more constrained environment. This signifies the suitability of the proposed flow for embedded IoT applications.

6.4.3 Branch Balancing

The results presented in Tables 6.2-6.5 include branch balancing, but it is only presented once in Table 6.2 as the results are identical across the experiments. Notably, several benchmarks did not require any branch balancing – the reason for this is the efficient and dedicated design. Blowfish, SHA, and SIMON all have a single purpose, namely the encryption or hashing of an input. As the benchmark only covers this specific functionality, the risk of accidental branch imbalance was greatly reduced. The results for the GSM benchmark provide an example of the other spectrum of branch balancing: the log area ratio (LAR) computed by the benchmark was considered to be the user-specified secret, and the benchmark code contains a large number of expensive operations that are controlled by conditionals which (indirectly) depend on the LAR value. Thus, branch balancing requires extensive resources to mitigate this situation. To illustrate the effectiveness of branch balancing, consider the case that virtually the full GSM benchmark is considered to be security critical – this can be achieved by specifying the input signal to be confidential. In this case, the branch balancing algorithm would require 184960 GE resources, 3.2 times as much as described in Table 6.2. This illustrates the significance of imbalanced branches, as well as the convenience of HLS.

6.4.4 Leakage Targeting

In addition to the evaluation of the resource-targeted synthesis flow, the leakage-targeted flow is extensively evaluated in Table 6.4. This mode synthesizes the provided high-level specification such that the estimated side-channel leakage is below a specified leakage

Table 6.4 Evaluation of the proposed synthesis under explicit side-channel leakage target. Compared to the baseline, the leakage target is achieved with 31% to 81% less overhead ($\%_B$). Compared to modular synthesis, overhead is reduced by up to 44% ($\%_M$).

Benchmark	$\boldsymbol{\theta}$ **Target**	**Baseline**		**Modular Synthesis**			**Proposed Side-Channel Synthesis**			
		O.H.	$\boldsymbol{\theta_B}$	**O.H.**	$\boldsymbol{\theta_M}$	**%**	**O.H.**	**θ**	$\%_B$	$\%_M$
AES	395	139740	394.7	109144	385.4	0.22	61568	395.0	0.56	0.44
Blowfish	342	65520	341.3	45528	341.7	0.31	43012	341.9	0.34	0.06
SHA	100	46080	99.4	28496	99.9	0.38	22160	99.9	0.52	0.22
SIMON	81	18240	80.4	16320	80.5	0.11	12512	81.0	0.31	0.23
GSM	272	400320	271.8	104624	271.6	0.74	93672	271.9	0.77	0.10
IoT	306	193080	305.3	42930	304.3	0.78	36004	305.6	0.81	0.16

target. This flow is interesting in different practical scenarios, for instance when the sub-module of a larger device is being synthesized. Here, it may be preferable to uniformly achieve a given leakage target throughout the circuit, rather than fully utilizing arbitrarily assigned resources. In comparison to the full-device baseline, the proposed synthesis achieves the leakage target with significantly reduced resource utilization. The reduction ranges from 31% for the highly efficient SIMON cipher to 81% for the IoT benchmark. The improvements over the modular baseline are more modest, ranging from 6% for the Blowfish benchmark up to 44% for AES. In contrast to the presented cost reduction of 56% over the baseline, note that manual partitioning has only achieved a reduction of 24% for AES [122]. The presented improvement can be attributed to the fine-grained recognition of leakage-prone operations. For this benchmark, the resource savings over the baseline is equivalent to 12,413 adaptive logic modules (ALMs) for the studied Cyclone V FPGA.

An evaluation with more modest leakage targets is presented in Table 6.5. This scenario can be desirable when the priority for side-channel defense is to function as a deterrent for weakly motivated adversaries with comparably low resource overhead. The results show that the proposed synthesis flow achieves fine-grained result tuning and therefore achieves considerably improved resource utilization. Compared to the modular baseline, the resource cost is reduced by up to 42% while achieving the same leakage targets. In this experiment, it is notable that the modular baseline has significantly exceeded the leakage target for the AES and Blowfish benchmarks, resulting in a leakage that is less than half of the requirement. The underlying cause for this over-engineered solution is the coarse granularity of block-based optimization, which emphasizes the strength of the proposed security synthesis with FU-based leakage optimization.

6.5 Conclusion and Outlook

Security is a core requirement for IoT and cloud computing due to the vast amounts of confidential and privacy sensitive data that is processed. Side-channel leakage is an important problem as it provides malicious adversaries indirect access to internal state and data such as secret keys. Defending against side-channel attacks such as DPA that exploit this leakage is an active research area, and is not easy to apply efficiently in practice without detailed hardware security and circuit-level understanding. In this chapter, we presented the first HLS flow that inherently minimizes side-channel leakage by inferring all security critical operations from a small number of user-specified confidential variables in their C-code to be synthesized. The HLS flow automatically analyzes the information content for all relevant operations and performs detailed simulations to perform Hamming distance based leakage estimation. The flow identifies and corrects any imbalanced branches that pose easy attack targets, and selectively upgrades functional units based on leakage potential and available resources. An extensive evaluation showed that side-channel leakage can be reduced by up to 72% under identical resource constraints when compared against typical full-device application of countermeasures. The results further show a reduction in resource consumption by up to 81% compared to the baseline to achieve a given leakage limit. Further investigation is needed in two directions: i) detailed evaluation and consideration of power and operation speed during synthesis; ii) incorporation of masking to further strengthen the resistance against DPA.

Table 6.5 Evaluation of leakage targeted synthesis under less severe leakage targets. Side-channel leakage is denoted by θ. Target is achieved with up to 30% to 67% reduced overhead.

Bench-mark	**Leakage Target**	**Modular Synthesis**		**Proposed SC Synthesis**		
		O.H.	$\boldsymbol{\theta_M}$	**O.H.**	**θ**	**%**
AES	1000	90120	480.2	32160	998.2	0.64
Blowfish	1000	39360	504.3	27532	999.9	0.30
SHA	450	8160	398.9	5280	449.7	0.35
SIMON	450	8640	304.2	2880	448.4	0.67
GSM	1000	22800	984.7	15840	999.3	0.31
IoT	1000	21600	802.3	12236	999.4	0.43

CHAPTER 7
CONCLUSIONS

In this dissertation, we have studied the security of hardware implementations and have introduced new defense mechanisms. The security of hardware requires special attention, as even widely accepted and used algorithms or protocols can be breached with limited effort unless proper countermeasures are implemented. There exist three primary threat to hardware implementations: i) physically invasive attacks which include physical modification of circuit behavior post-manufacturing; ii) the insertion of hardware Trojan horses by a malicious foundry which can leak secret information with minimal overhead; iii) side-channel analysis of power traces, which can reveal the values of secrets being processed due to the correlation between input signals and dynamic power consumption of the hardware. This dissertation covers all three of these areas.

In Chapter 2, we presented a PUF design based on intrinsic physical variations of CNTs. It takes advantage of the metallic to semiconducting CNT ratio in CNFETs to increase reliability, while strongly reducing the average power consumption and energy usage per bit. CNPUF was experimentally evaluated with SPICE-accurate simulations and showed strong results for security relevant properties such as reliability and inter-chip distance. Furthermore, we presented and evaluated an extension of CNPUF that allows a power-security tradeoff for dynamic usage in high security circuits. CNPUF and ex-CNPUF provide the future basis for authentication and secret key generation by offering security at a very low area and power cost. This can open the field of PUF for a variety of new applications and is especially relevant for current research areas such as wireless sensor networks or ubiquitous computing.

In Chapter 3, we introduced a new system-level security model that bridges the chasm between application-level security analysis and design of secure hardware, and models for isolated components. From this model, we analyzed and explained several hardware security requirements using existing protocols, and showed that they cannot be fulfilled without extensive cost. We presented a multilevel authentication protocol which is verified using the system-level security model and which takes advantage of a combination of different PUF-designs to minimize resource allocation. SoP does not require expensive error-correction, as high reliability designs are employed where required. Furthermore, the need

for latency and power intensive hash functions on the PUF circuit is replaced by a combination of strong PUFs and off-chip cryptographic hash. With breach recognition and recovery, new security features are introduced and shown to increase the attack-difficulty while enhancing reusability. A low-cost implementation of SoP was shown to reduce the area by 64% in a gate-level comparison. This low resource allocation and high flexibility allow SoP to provide a security solution tailored for ubiquitous computing devices.

In Chapter 4, we proposed PolyPUF, a widely applicable PUF architecture that employs challenge and response self-divergence to provide polymorphous PUF behavior. This changes the challenge-response behavior to be non-deterministic and unpredictable, while still being verifiable in an authentication scenario. In an extensive evaluation, this polymorphic behavior was shown to provide strong resistivity against model-building attacks while simultaneously providing very low overhead.

In Chapter 5, the focus changed to the threat of hardware Trojan horses. We presented a security optimization flow that utilizes resources with high efficiency to identify and disperse security critical information through multiple operators and registers. Moreover, we introduced an obfuscation flow that is embedded in the high-level synthesis flow and enables threat-targeted security optimization under resource constraints. Engineers merely need to define the initial security critical variable, and the downstream vulnerabilities are automatically detected and defended. The evaluation showed security enhancements with up to 5-times higher information dispersion and significantly higher Trojan insertion difficulty under the same resource constraints as a baseline technique. A threat-targeted evaluation showed that the co-optimization of obfuscation and dispersion can improve security by a factor between 8.37 and 1932, emphasizing the strength and flexibility of threat-targeted synthesis.

In Chapter 6, the problem of side-channel information leakage is further studied. We presented the first HLS flow that inherently minimizes side-channel leakage by inferring all security critical operations from a small number of user-specified confidential variables in their C-code to be synthesized. The HLS flow automatically analyzes the information content for all relevant operations and performs detailed simulations to perform Hamming distance based leakage estimation. The flow identifies and corrects any imbalanced branches that pose easy attack targets, and selectively upgrades functional units based on leakage potential and available resources. An extensive evaluation showed that side-channel leakage can be reduced by up to 72% under identical resource constraints when compared

against typical full-device application of countermeasures. The results further show a reduction in resource consumption by up to 81% compared to the baseline to achieve a given leakage limit. Further investigation is needed in two directions: i) detailed evaluation and consideration of power and operation speed during synthesis; ii) incorporation of masking to further strengthen the resistance against DPA.

In conclusion, this dissertation contributed to hardware security research by introducing new PUF designs and systems to improve defenses against physical attacks. Furthermore, it introduced high-level synthesis flows that utilize existing work to increase the difficulty of hardware Trojan horse insertion and reduce side-channel information leakage which could be exploited in side-channel analysis attacks.

REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Comput. Netw.*, vol. 38, no. 4, pp. 393–422, 2002.

[2] T. Starner, "The challenges of wearable computing: Part 2," *Micro*, vol. 21.4, no. IEEE, pp. 54–67, 2001.

[3] R. Lee, S. Sethumadhavan, and G. E. Suh, "Hardware enhanced security," *Proc. 2012 ACM Conf. Comput. Commun. Secur. - CCS 12*, p. 1052, 2012.

[4] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," *Proc. 9th ACM Conf. Comput. Commun. Secur.*, 2002.

[5] R. Nithyanand and J. Solis, "A Theoretical Analysis: Physical Unclonable Functions and the Software Protection Problem," *2012 IEEE Symp. Secur. Priv. Workshop*, pp. 1–11, May 2012.

[6] D. Lim, J. W. Lee, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "Extracting Secret Keys From Integrated Circuits," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 13, no. 10, pp. 1200–1205, 2005.

[7] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proceedings of the 44th annual Design Automation Conference*, 2007.

[8] S. S. Kumar, J. Guajardo, R. Maes, G. Schrijen, and P. Tuyls, "Extended Abstract : The Butterfly PUF Protecting IP on every FPGA," Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on. IEEE, 2008.

[9] Y. Yao, M. Kim, J. Li, I. Markov, and F. Koushanfar, "ClockPUF: Physical Unclonable Functions based on Clock Networks," in *Design, Automation & Test in Europe*, 2013.

[10] M. Majzoobi, G. Ghiaasi, F. Koushanfar, and S. R. Nassif, "Ultra-low power current-based PUF," *2011 IEEE Int. Symp. Circuits Syst. ISCAS*, pp. 2071–2074, May 2011.

[11] S. Selvarasah, "Ultrathin and highly flexible parylene-c packaged carbon nanotube field effect transistors," PhD dissertation, Northeastern University Boston, 2010.

[12] S. T. C. Konigsmark, L. K. Hwang, D. Chen, and M. D. Wong, "CNPUF: A Carbon Nanotube-based Physically Unclonable Function for Secure Low-Energy Hardware Design," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, 2014, pp. 73–78.

[13] M. Zhang and J. Li, "Carbon nanotube in different shapes," *Mater. Today*, vol. 12, no. 6, pp. 12–18, Jun. 2009.

[14] S. J. Tans, A. R. M. Verschueren, and C. Dekker, "Room-temperature transistor based on a single carbon nanotube," *Nature*, vol. 672, no. 1989, pp. 669–672, 1998.

[15] H. Park *et al.*, "High-density integration of carbon nanotubes via chemical self-assembly.," *Nat. Nanotechnol.*, vol. 7, no. 12, pp. 787–91, Dec. 2012.

[16] M. Shulaker *et al.*, “Sacha : the Stanford Carbon Nanotube Controlled Handshaking Robot,” in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 2–4.
[17] M. M. Shulaker *et al.*, “Carbon nanotube computer,” *Nature*, vol. 501, no. 7468, pp. 526–30, Sep. 2013.
[18] J. Zhang et al., “Robust Digital VLSI using Carbon Nanotubes,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 31, no. 4, pp. 453–471, 2012.
[19] J. Zhang, “Variation-Aware Design of Carbon Nanotube Digital VLSI Circuits,” PhD dissertation, Stanford University, August, 2011.
[20] H. Wei *et al.*, “Efficient metallic carbon nanotube removal readily scalable to wafer-level VLSI CNFET circuits,” in *VLSI Technology (VLSIT), 2010 Symposium on*, 2010, pp. 237–238.
[20] J.W.G. Wilder, L.C. Venema, A.G. Rinzler, R.E. Smalley, and C. Dekker, “Electronic structure of atomically resolved carbon nanotubes,” Nature, vol. 584, no. 10, pp. 1996–1999, 1998.
[22] J. Deng and H.-S. Wong, “A compact SPICE model for carbon-nanotube field-effect transistors including nonidealities and its application—Part I: Model of the intrinsic channel region,” *Electron Devices IEEE Trans. On*, vol. 54, no. 12, pp. 3186–3194, 2007.
[23] J. Deng and H.-S. Wong, “A compact SPICE model for carbon-nanotube field-effect transistors including nonidealities and its application—Part II: Full device model and circuit performance benchmarking,” *Electron Devices IEEE Trans. On*, vol. 54, no. 12, pp. 3195–3205, 2007.
[24] S. Sinha, G. Yeric, V. Chandra, B. Cline, and Y. Cao, “Exploring sub-20nm FinFET design with predictive technology models,” in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 283–288.
[25] A. R. Krishna and S. Bhunia, “ScanPUF: Robust ultralow-overhead PUF using scan chain,” *2013 18th Asia S. Pac. Des. Autom. Conf. ASP-DAC*, pp. 626–631, Jan. 2013.
[25] S. B. Ors, G. Frank, E. Oswald, B. Preneel, and A. Graz, “Power-Analysis Attack on an ASIC AES implementation,” in Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on. Vol. 2, 2004.
[27] A. Salomaa, *Public-key cryptography*, vol. 23. Springer-Verlag New York Incorporated, 1996.
[28] C. Helfmeier, D. Nedospasov, C. Tarnovsky, J. S. Krissler, C. Boit, and J.-P. Seifert, “Breaking and entering through the silicon,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 733–744.
[29] D. Nedospasov, J.-P. Seifert, C. Helfmeier, and C. Boit, “Invasive PUF analysis,” in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*, 2013, pp. 30–38.
[30] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach, and S. Devadas, “Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by

Substring Matching," in *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, 2012.

[31] U. Ruhrmair and M. van Dijk, "PUFs in Security Protocols: Attack Models and Security Evaluations," in *Security and Privacy (SP), 2013 IEEE Symposium on*, 2013, pp. 286–300.

[32] S. T. C. Konigsmark, L. K. Hwang, D. Chen, and M. D. F. Wong, "System-of-PUFs: multilevel security for embedded systems," in *CODES+ISSS*, 2014.

[33] R. Maes and I. Verbauwhede, "Physically unclonable functions: A study on the state of the art and future research directions," *Hardw.-Intrinsic Secur.*, pp. 3–37, 2010.

[34] C. Bohm, M. Hofer, and W. Pribyl, "A microcontroller SRAM-PUF," *5th Int. Conf. Netw. Syst. Secur.*, Sep. 2011.

[35] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," *Comput. IEEE Trans. On*, vol. 58, no. 9, pp. 1198–1210, 2009.

[36] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 237–249.

[37] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Testing Techniques for Hardware Security," *IEEE Int. Test Conf.*, Oct. 2008.

[38] U. Rührmair, H. Busch, and S. Katzenbeisser, "Strong PUFs: Models, Constructions, and Security Proofs," in *Towards Hardware-Intrinsic Security*, A.-R. Sadeghi and D. Naccache, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 79–96.

[39] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, K. U. Leuven, and E. Cosic, "Reverse Fuzzy Extractors : Enabling Lightweight Mutual Authentication for PUF-enabled RFIDs," *Financ. Cryptogr. Data Secur.*, pp. 374–389, 2012.

[40] M.-D. (Mandel) Yu and S. Devadas, "Secure and Robust Error Correction for Physical Unclonable Functions," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 48–65, Jan. 2010.

[41] M. Hiller, D. Merli, F. Stumpf, and G. Sigl, "Complementary IBS: Application specific error correction for PUFs," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, 2012, no. i.

[42] H. Busch, S. Katzenbeisser, and P. Baecher, "PUF-Based Authentication Protocols–Revisited," in *Information Security Applications*, Springer, 2009, pp. 296–308.

[43] M. Feldhofer and C. Rechberger, "A case against currently used hash functions in RFID protocols," in *On the move to meaningful internet systems 2006: OTM 2006 workshops*, 2006, pp. 372–381.

[44] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varıcı, and I. Verbauwhede, "SPONGENT: A lightweight hash function," in *Cryptographic Hardware and Embedded Systems–CHES 2011*, Springer, 2011, pp. 312–325.

[45] N. Beckmann and M. Potkonjak, "Hardware-based public-key cryptography with public physically unclonable functions," in *Information Hiding*, 2009.

[46] M. Potkonjak, S. Meguerdichian, A. Nahapetian, and S. Wei, “Differential public physically unclonable functions: architecture and applications,” *48th ACMEDACIEEE Des. Autom. Conf.*, 2011.

[45] E. Öztürk, G. Hammouri, and B. Sunar, “Towards robust low cost authentication for pervasive devices,” in Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on, 2008, pp. 170–178.

[48] G. Hammouri, E. Öztürk, and B. Sunar, “A tamper-proof and lightweight authentication scheme,” *Pervasive Mob. Comput.*, vol. 4, no. 6, pp. 807–818, Dec. 2008.

[49] R. Plaga and F. Koob, “A formal definition and a new security mechanism of physical unclonable functions,” in *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, Springer, 2012, pp. 288–301.

[50] F. Armknecht, R. Maes, A.-R. Sadeghi, F.-X. Standaert, and C. Wachsmann, “A Formalization of the Security Features of Physical Functions,” in *Security and Privacy (SP), 2011 IEEE Symposium on*, 2011, pp. 397–412.

[51] W. Stallings and L. Brown, *Computer security: principles and practice*. Boston: Pearson, 2012.

[52] M. G. Kuhn and Oliver Kommerling, “Design Principles for Tamper-Resistant Smartcard Processors,” *USENIX Workshop Smartcard Technol.*, 1999.

[53] K. Pongaliur, Z. Abraham, A. X. Liu, L. Xiao, and L. Kempel, “Securing Sensor Nodes Against Side Channel Attacks,” in *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*, 2008, pp. 353–361.

[54] J. Ferrigno and M. Hlaváč, “When AES blinks: introducing optical side channel,” *IET Inf. Secur.*, vol. 2, no. 3, p. 94, 2008.

[55] N. Vachharajani *et al.*, “RIFLE: An architectural framework for user-centric information-flow security,” in *Microarchitecture, 2004. MICRO-37 2004. 37th International Symposium on*, 2004, pp. 243–254.

[56] O. Goldreich and R. Ostrovsky, “Software Protection and Simulation on Oblivious RAMs,” *J ACM*, vol. 43, no. 3, pp. 431–473, May 1996.

[57] M. Maas *et al.*, “PHANTOM: practical oblivious computation in a secure processor,” 2013, pp. 311–324.

[58] U. Rührmair, F. Sehnke, and J. Sölter, “Modeling attacks on physical unclonable functions,” *Proc. 17th ACM Conf. Comput. Commun. Secur.*, 2010.

[59] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, “FPGA intrinsic PUFs and their use for IP protection,” in *Cryptographic Hardware and Embedded Systems-CHES 2007*, Springer, 2007, pp. 63–80.

[60] E. Simpson and P. Schaumont, “Offline hardware/software authentication for reconfigurable platforms,” in *Cryptographic Hardware and Embedded Systems-CHES 2006*, Springer, 2006, pp. 311–323.

[61] J. Delvaux and I. Verbauwhede, “Side Channel Modeling Attacks on 65nm Arbiter PUFs Exploiting CMOS Device Noise,” *Hardw.-Oriented Secur. Trust HOST 2013 IEEE Int. Symp. On*.

[62] C.-E. Yin and G. Qu, "Lisa: Maximizing ro puf's secret extraction," in *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, 2010, pp. 100–105.
[63] H.-M. Sun, "An efficient remote use authentication scheme using smart cards," *Consum. Electron. IEEE Trans. On*, vol. 46, no. 4, pp. 958–961, 2000.
[64] Z. Benenson, N. Gedicke, and O. Raivio, "Realizing robust user authentication in sensor networks," *Real-World Wirel. Sens. Netw. REALWSN*, vol. 14, 2005.
[65] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Cryptographic Hardware and Embedded Systems-CHES 2004*, Springer, 2004, pp. 357–370.
[66] F. Koushanfar *et al.*, "Can EDA combat the rise of electronic counterfeiting?," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 133–138.
[67] G. Hammouri and B. Sunar, "PUF-HB: A tamper-resilient HB based authentication protocol," in *Applied Cryptography and Network Security*, 2008, pp. 346–365.
[67] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks," Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on. 2005.
[69] S. T. C. Konigsmark, D. Chen, and M. D. F. Wong, "PolyPUF: Physically Secure Self-Divergence," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 7, pp. 1053–1066, Jul. 2016.
[70] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, *FPGA intrinsic PUFs and their use for IP protection*. Springer, 2007.
[71] U. Ruhrmair *et al.*, "PUF Modeling Attacks on Simulated and Silicon Data," *IEEE Trans. Inf. Forensics Secur.*, vol. 8, no. 11, pp. 1876–1891, Nov. 2013.
[72] M.-D. M. Yu, I. Verbauwhede, S. Devadas, and D. M'Raïhi, "A Noise Bifurcation Architecture for Linear Additive Physical Functions," *Hardw.-Oriented Secur. Trust HOST 2014 IEEE Int. Symp. On*.
[73] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.
[74] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Neural Networks, 1993., IEEE International Conference on*, 1993, pp. 586–591.
[75] P. L. Bartlett and W. Maass, "Vapnik Chervonenkis Dimension of Neural Nets," *Handb. Brain Theory Neural Netw.*, pp. 1188–1192, 2003.
[76] K. Tiri *et al.*, "A side-channel leakage free coprocessor IC in 0.18μm CMOS for embedded AES-based cryptographic and biometric processing," in *Design Automation Conference, 2005. Proceedings. 42nd*, 2005, pp. 222–227.
[77] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 709–720.
[78] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and R. Srivaths, "Security as a new dimension in embedded system design," in *Proceedings of the 41st annual Design Automation Conference*, 2004, pp. 753–760.

[79] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, “Secure Lightweight Entity Authentication with Strong PUFs: Mission Impossible?,” in *Cryptographic Hardware and Embedded Systems–CHES 2014*, Springer, 2014, pp. 451–475.

[80] S. P. Skorobogatov and R. J. Anderson, “Optical fault induction attacks,” in *Cryptographic Hardware and Embedded Systems-CHES 2002*, 2003.

[81] A. Van Herrewege *et al.*, “Reverse fuzzy extractors: Enabling lightweight mutual authentication for PUF-enabled RFIDs,” in *Financial Cryptography and Data Security*, Springer, 2012, pp. 374–389.

[82] P. Koeberl, J. Li, A. Rajan, and W. Wu, “Entropy loss in PUF-based key generation schemes: The repetition code pitfall,” in *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, 2014, pp. 44–49.

[83] M. Rostami, M. Majzoobi, F. Koushanfar, D. S. Wallach, and S. Devadas, “Robust and Reverse-Engineering Resilient PUF Authentication and Key-Exchange by Substring Matching,” *IEEE Trans. Emerg. Top. Comput.*, vol. 2, no. 1, pp. 37–49, Mar. 2014.

[84] R. Bonetto, N. Bui, V. Lakkundi, A. Olivereau, A. Serbanati, and M. Rossi, “Secure communication for smart IoT objects: Protocol stacks, use cases and practical examples,” in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, 2012, pp. 1–7.

[85] S. Bhunia *et al.*, “Protection Against Hardware Trojan Attacks: Towards a Comprehensive Solution,” *IEEE Des. Test*, vol. 30, no. 3, pp. 6–17, Jun. 2013.

[86] J. Li and J. Lach, “At-speed delay characterization for IC authentication and Trojan horse detection,” in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, 2008, pp. 8–14.

[87] S. Skorobogatov and C. Woods, in *Breakthrough silicon scanning discovers backdoor in military chip*, 2012.

[88] L. Lin, W. Burleson, and C. Paar, “MOLES: malicious off-chip leakage enabled by side-channels,” in *Proceedings of the 2009 International Conference on Computer-Aided Design*, 2009, pp. 117–122.

[89] M. Yoshimura, A. Ogita, and T. Hosokawa, “A smart Trojan circuit and smart attack method in AES encryption circuits,” in *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on*, 2013, pp. 278–283.

[90] S. T. C. Konigsmark, D. Chen, and M. D. F. Wong, “Information Dispersion for Trojan Defense through High-Level Synthesis,” presented at the Proceedings of the 53rd Annual Design Automation Conference, 2016, pp. 1–6.

[91] X. Cui, K. Ma, L. Shi, and K. Wu, “High-level synthesis for run-time hardware Trojan detection and recovery,” in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[92] K. Xiao and M. Tehranipoor, “BISA: Built-in self-authentication for preventing hardware Trojan insertion,” in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, 2013, pp. 45–50.

[93] R. S. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.

[94] J. Rajendran, H. Zhang, O. Sinanoglu, and R. Karri, "High-level synthesis for security and trust," in *On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International*, 2013, pp. 232–233.

[95] M. Ben Hammouda, P. Coussy, and L. Lagadec, "A design approach to automatically synthesize ansi-c assertions during high-level synthesis of hardware accelerators," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, 2014, pp. 165–168.

[96] Y. Jin and Y. Makris, "Hardware Trojan detection using path delay fingerprint," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, 2008, pp. 51–57.

[97] F. Wolff, C. Papachristou, S. Bhunia, and R. S. Chakraborty, "Towards Trojan-free trusted ICs: Problem analysis and detection scheme," in *Proceedings of the conference on Design, automation and test in Europe*, 2008, pp. 1362–1365.

[98] B. Khaleghi, A. Ahari, H. Asadi, and S. Bayat-Sarmadi, "FPGA-Based Protection Scheme against Hardware Trojan Horse Insertion Using Dummy Logic," *IEEE Embed. Syst. Lett.*, vol. 7, no. 2, pp. 46–50, Jun. 2015.

[99] J. Zhang, "A Practical Logic Obfuscation Technique for Hardware Security," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 24, no. 3, pp. 1193–1197, Mar. 2016.

[100] R. S. Chakraborty and S. Bhunia, "Security against hardware Trojan through a novel application of design obfuscation," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, 2009, pp. 113–116.

[101] M. Rostami, F. Koushanfar, J. Rajendran, and R. Karri, "Hardware security: Threat models and metrics," in *Proceedings of the International Conference on Computer-Aided Design*, 2013, pp. 819–823.

[102] S. Bhasin and F. Regazzoni, "A survey on hardware trojan detection techniques," in *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, 2015, pp. 2021–2024.

[103] Y. Liu, Y. Jin, and Y. Makris, "Hardware Trojans in wireless cryptographic ICs: Silicon demonstration & detection method evaluation," in *Proceedings of the International Conference on Computer-Aided Design*, 2013, pp. 399–404.

[104] C. Lattner, "LLVM and Clang: Next generation compiler technology," in *The BSD Conference*, 2008, pp. 1–2.

[105] K. Kursawe, A.-R. Sadeghi, D. Schellekens, B. Škorić, and P. Tuyls, "Reconfigurable physical unclonable functions-enabling technology for tamper-resistant storage," in *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on*, 2009, pp. 22–29.

[105] S. Hadjis et al., "Impact of FPGA architecture on resource sharing in high-level synthesis," in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2012, pp. 111–114.

[107] A. Canis *et al.*, "LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 2, pp. 1–27, Sep. 2013.

[108] Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Proposal and Quantitative Analysis of the CHStone Benchmark Program Suite for Practical C-based High-level Synthesis," *J. Inf. Process.*, vol. 17, pp. 242–254, 2009.

[109] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[110] D. Liu, P. Ning, and W. Du, "Detecting malicious beacon nodes for secure location discovery in wireless sensor networks," in *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, 2005, pp. 609–619.

[111] K. Eguro and R. Venkatesan, "FPGAs for trusted cloud computing," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 63–70.

[112] A. P. Johnson, R. S. Chakraborty, and D. Mukhopadhyay, "A PUF-Enabled Secure Architecture for FPGA-Based IoT Applications," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 1, no. 2, pp. 110–122, Apr. 2015.

[113] I. M. R. Verbauwhede, Ed., *Secure Integrated Circuits and Systems*. Boston, MA: Springer US, 2010.

[114] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, vol. 1666, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[115] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-Analysis Attack on an ASIC AES implementation," in *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, 2004, vol. 2, pp. 546–552.

[116] K. Tiri, "Side-channel attack pitfalls," in *Proceedings of the 44th annual Design Automation Conference*, 2007, pp. 15–20.

[116] E. Trichina, "Combinational Logic Design for AES SubByte Transformation on Masked Data," IACR Cryptol. EPrint Arch., vol. 2003, p. 236, 2003.

[118] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proceedings of the conference on Design, automation and test in Europe-Volume 1*, 2004, p. 10246.

[119] R. Velegalati and J.-P. Kaps, "DPA resistance for light-weight implementations of cryptographic algorithms on FPGAs," 2009, pp. 385–390.

[120] P. Yu and P. Schaumont, "Secure FPGA circuits using controlled placement and routing," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2007 5th IEEE/ACM/IFIP International Conference on*, 2007, pp. 45–50.

[121] A. Wild, A. Moradi, and T. Güneysu, "Evaluating the duplication of dual-rail precharge logics on FPGAs," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2015, pp. 81–94.

[122] J.-P. Kaps and R. Velegalati, “DPA Resistant AES on FPGA Using Partial DDL,” 2010, pp. 273–280.
[123] L. Xu, W. Shi, and T. Suh, “PFC: Privacy Preserving FPGA Cloud - A Case Study of MapReduce,” 2014, pp. 280–287.
[124] M. Rao, T. Newe, and I. Grout, “Secure Hash Algorithm-3 (SHA-3) implementation on Xilinx FPGAs, Suitable for IoT Applications,” in *8th International Conference on Sensing Technology (ICST 2014), Liverpool John Moores University, Liverpool, United Kingdom, 2nd-4th September*, 2014.
[124] R. Dobbins, “Mirai IoT botnet description and DDoS attack mitigation,” Arbor Threat Intell., vol. 28, 2016.
[126] K. Tiri and I. Verbauwhede, “Simulation models for side-channel information leaks,” in *Proceedings of the 42nd annual Design Automation Conference*, 2005, pp. 228–233.
[127] X. Fang, P. Luo, Y. Fei, and M. Leeser, “Balance power leakage to fight against side-channel analysis at gate level in FPGAs,” in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2015, pp. 154–155.
[128] F. Menichelli, R. Menicocci, M. Olivieri, and A. Trifiletti, “High-Level Side-Channel Attack Modeling and Simulation for Security-Critical Systems on Chips,” *IEEE Trans. Dependable Secure Comput.*, vol. 5, no. 3, pp. 164–176, Jul. 2008.
[128] J. den Harog and E. de Vink, “Virtual Analysis and Reduction of Side-Channel Vulnerabilities of Smartcards,” in Formal Aspects in Security and Trust. Springer US, 2005.
[130] N.-F. Standard, “Announcing the advanced encryption standard (AES),” *Fed. Inf. Process. Stand. Publ.*, vol. 197, pp. 1–51, 2001.
[131] B. Schneier, “Description of a new variable-length key, 64-bit block cipher (Blowfish),” in *International Workshop on Fast Software Encryption*, 1993, pp. 191–204.
[131] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, “The SIMON and SPECK Families of Lightweight Block Ciphers,” in Cryptology ePrint Archive Report 2013/404, 2013
[132] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, “The SIMON and SPECK lightweight block ciphers,” in Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE, 2015.
[134] A. Moradi, “Side-channel leakage through static power,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, 2014, pp. 562–579.