Technische Universität Berlin

Faculty IV - Electrical Engineering and Computer Science
Chair for Security in Telecommunications

Master Thesis

Security Monitoring of FPGAs using Physically Unclonable Functions

Julian Fietkau
fietkau@campus.tu-berlin.de
Matrikelnummer: xxx
November 25, 2016

Supervisor: Prof. Dr. Jean-Pierre Seifert
Co-Supervisor: Shahin Tajik

Abstract

Secure systems must be built on top of secure hardware, resistant against a big variety of attacks trying to extract cryptographic keys and confidential data. For this reason, system manufacturers want to develop satisfying solutions which impede attacks or ideally render them impossible. Mostly, this is done by implementing sensors and watchdogs that permanently monitor the device and trigger a mitigation routine for malicious attempts, like system clock manipulation, modified device logic or unusual environmental conditions. Certainly, most of these solutions do not address the root cause of the problem and generally have always been bypassed or leveraged by sophisticated attacks spoofing sensors or thwarting detection logic. Meanwhile, a much better approach have been found in the area of intrinsic security that change the way how secrets are stored on secure devices. So-called Physically Uncloneable Functions (PUFs) will allow us to derive secrets from a unique and tamper-evident device property in a repeatable and unpredictable way. So far, this will mitigate several attack types and allow us to build secure key storages, but indeed it cannot prevent all occurring threats to secure devices.

To overcome the limitations of these approaches, this thesis will show how the traditional idea of actively monitoring the device can be linked to an intrinsic device property, which cannot be spoofed or tampered with. Therefore we use a well-known digital-intrinsic PUF architecture, namely RO Sum PUF, to monitor the physical integrity of an integrated circuit. This novel idea will allow us to detect attack types not covered by PUFs originally, in remote or local applications and quickly react to suspicious situations for instance by triggering a mitigation routine. Moreover, the PUF is still able to be used for its origin purposes, e.g. authentication and secure key storage, which promotes the idea to a comprehensive security solution covering multiple threats.

To enable this, we will firstly contribute the theoretically considerations and present a generic architecture to demonstrate how the solution can be realized in real world scenario under different constraints. Secondly, we provide a prototype implementation covering software and hardware components using the versatile FPGA platform. To prove the overall performance of this approach, we exposed a prepared device containing our implementation to various attack scenarios including Trojan insertions as well as glitching, laser and temperature attacks. Based on this experiments, we finally evaluate the capability of our approach to improve the overall device security and figure out its advantages and tradeoffs.

Zusamenfassung

Sichere IT Systeme basieren grundlegend auf sicherer Hardware, die Angriffen widerstehen müssen bei denen kryptografische Schlüssel und vertrauliche Daten extrahiert werden könnten. Aus diesem Grund versuchen Hersteller dieser Systeme seit langem nachhaltige Lösungen zu präsentieren, die es erlauben Angriffe zu erschweren oder bestenfalls unmöglich zu machen. Üblicherweise wird dies mit einfachen Sensoren und Alarmsystemen umgesetzt, die das Gerät permanent überwachen und Notfallroutinen durchführen, sobald eine bösartige Verhaltensweise festgestellt wird, wie bspw. die Manipulation des Systemtakts oder ungewöhnliche Umgebungsparameter. Leider adressieren die meisten dieser Lösungen nicht das eigentliche Problem und können im allgemeinen ausgehebelt werden, indem ein Angreifer Sensorwerte fälscht oder das Alarmsystem selbst manipuliert. Aus dieser Perspektive betrachtet stellt dieser Ansatz keine hinreichende Lösung dar. Ein ganz anderer Ansatz ist dafür im Bereich der intrinsischen Sicherheit zu finden. Die Verwendung von sogenannten “Physically Uncloneable Functions“ (PUFs) ermöglicht unter anderem einen kryptografischen Schlüssel von einer einzigartigen und abhörsicheren Komponente abzuleiten und dies in wiederholbarer und unberechenbarer Weise. Somit können bereits sichere Schlüsselspeicher gebaut werden, die viele invasive Angriffe im allgemeinen verhindern können, aber dennoch bleiben viele Angriffstechniken weiterhin möglich. In dieser Arbeit präsentieren wir eine neuartige Lösung, die beide Ansätze kombiniert indem die traditionelle Idee des aktiven Monitoring auf eine intrinsische Komponente übertragen wird, die nicht manipuliert oder getäuscht werden kann. Dafür haben wir eine bewährte PUF-Architektur verwenden, den RO Sum PUF, um aktiv die physikalische Integrität eines Schaltkreises zu überwachen. Dieser neue Ansatz ermöglicht uns online und offline Angriffsszenarien zu detektieren, die bisher von PUFs unentdeckt geblieben sind und zudem auf diese zu reagieren, indem zum Beispiel Notfallroutinen ausführt werden. Darüber hinaus kann der PUF weiterhin als Schlüsselspeicher oder zur Authentifikation verwendet werden, was aus unserem Ansatz eine umfassende Sicherheitslösung macht, die mehrere Probleme gleichzeitig adressiert.
In dieser Arbeit haben wir die dafür notwendigen theoretischen Grundlagen erarbeitet und eine generische Architektur entwickelt, die verschiedene Anwendungsszenarien berücksichtigt. Weiterhin haben wir eine Implementation mit Software- und Hardwarekomponenten erstellt, dessen Leistungsfähigkeit wir unter Zuhilfenahme von FPGAs evaluiert haben. Dafür wurde die Implementation unterschiedlichen Angriffsszenarien ausgesetzt wie bspw. Glitching, Temperatur- und Laserangriffe, anhand dessen wir belegen können, dass unser Ansatz die allgemeine Gerätesicherheit erhöhen kann.

Acknowledgments

First of all, I would like to thank everyone from the SecT department: Prof. Seifert for supervising my thesis and for giving me the opportunity to be part of the team, Shahin Tajik for all the great discussions, motivating ideas and helpful advices and everyone else who taught me much and shaped my way of thinking in recent years like Dmitry Nedospasov, Jan Nordholz and Bhargava Shastry. Furthermore, special thanks to the HLB research group for providing the necessary tools and to Shahin and Heiko for their guidance during the experiments.
But most of all, I would like to thank my family for supporting and motivating me ever since I can remember and most notably Vivian, who stuck with me through thick and thin. Thank you all!

Contents

List of Figures

List of Tables

Glossary

AES	Advanced Encryption Standard
CMOS	Complementary Metal–Oxide–Semiconductor
CRP	Challenge Response Pair
DES	Data Encryption Standard
FIB	Focused Ion Beam
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuit
IDE	Integrated Development Environment
IP	Intellectual Property
JTAG	Joint Test Action Group
LAB	Logic Array Block
LFSR	Linear-Feedback Shift Register
MSE	Mean Squared Error
NIST	National Institute of Standards and Technology
PUF	Physical Unclonable Function
RO	Ring Oscillator
SCA	Side-Channel Analysis

1 Introduction

1.1 Motivation

Secure devices are an important key technology in modern IT systems. They are able to provide confidentiality, integrity and authenticity and represent the root of trust in several applications. A secure device typically execute cryptographic algorithms and thus must be able to safely store secrets. Examples for such devices can be found anywhere in daily life: smart cards used for financial transactions and health insurance, electronic car keys or mobile phones. Additionally plenty of them are much more embedded in daily infrastructure and more unlikely too perceive. For instance mobile network base stations, internet access points, connected control systems and in near future any device participating on the internet of things. Obviously these devices have to be secured because modification, duplication and other threats yield high financial and social damage. A successful attacker can steal confidential data, poses as another person or harm asset owners in various ways. The practical relevance of such attacks and their resulting consequences can exemplary be seen in the large variety of reported incident in recent years.
For example, in mid 2016 security researchers of the University of Birmingham reported a huge security weakness in the wireless key technology of Volkswagen Group cars. In their study [24], they show that the entry system deployed in most vehicles manufactured between 1995 and 2016 rely only on a few, global master keys. In fact, just a single key is used across millions of Volkswagen Group cars while overall only four keys are deployed in other VW Group cars including Audi, Bentley, SEAT and more. According, to their own estimation the total number of affected cars might be count several million worldwide. Indeed, the researchers were able to extract these shared keys from one of the car's internal electronic control units with common attack techniques and furthermore reversed the overall implementation design. To protect the customers they did not fully disclose their findings, but exploited the flaw and showed the practical consequences. To do so, they build a cheap $40 device which eavesdrop the radio signals from a wireless key fob of the legitimate car owner. By receiving just a single signal and combine it with the extracted cryptographic keys, they were able to create a clone of the origin key in less than a minute, allowing them to access the car or lock out the legitimate car owner. [24]
As we can see in this example, those electronic devices contain critical secrets and intellectual property that represents highly interesting and valuable targets for adversaries. Because of that, device manufactures progressively trying to integrate countermeasures and hardening their systems against common attacks. Traditionally this is done through firmware encryption, design and algorithmic obfuscation or physically by embedding shields and meshes into the integrated circuit. Some of these techniques allowing to prevent or retard occurring attacks, while others trying to detect attacks and initiate mitigation routines which quickly removes any secrets from the device.

1.2 Problem Description

Unfortunately, security mechanisms are often implemented badly by vendors because of conceptual misunderstandings or simply to save money. Besides that, even when done correctly the countermeasures are often insufficient or do not address the root cause at all. Because of that, sophisticated attackers with a decent set of tools and expertise have always been able to spoof, thwart or bypass any of these solutions so far, at least by using fully-invasive techniques. Hence, these traditional countermeasures approaching their limits and seems not able to provide the required level of protection for secure devices.
That's why intrinsic security is trying to address these shortcomings and provide a new way of safeguarding intellectual property and device secrets. To do so, cryptographic secrets will be linked to tamper-evident properties, such as manufacturing varieties which are unpredictable and unique for each device. So far, the most promising approach for implementing those identifiers are Physically Uncloneable Functions (PUFs). A PUF allows us to derive a secret key from unique device properties in a repeatable and unpredictable way. Thereby the security of the PUF relies simply on the integrity of the component embedding it. An attacker trying to tamper with the device, might permanently alter the IC or its package and thereby simultaneously alter the PUF intrinsics which make it behave suspiciously. Indeed, this is a strong weapon against many invasive techniques, but do not cover all less or non-invasive attacks, which doesn't alter the IC or just introduce very slight changes. Because of that, many attacks will still be a threat for devices integrating this technology by attacking the PUF itself or its derived secrets during execution. In conclusion, PUFs as well as traditional countermeasures will not represent a satisfying solution overall and further ways must be found to protect cryptographic devices.

1.3 Goal of Thesis

The goal of this thesis will be to combine the idea of traditional and intrinsic mechanisms to derive a new solution that improves the overall device security. Therefore, we will use a physically uncloneable function to actively monitor the physical integrity of an integrated circuit. This strategy will promise a way to detect attack types not covered by PUFs originally while further allow us to quickly react on suspicious situations for instance by triggering mitigation routines.
Probably, this approach will not work with all known types of PUFs, hence an initial objective of this thesis will be to elaborate the general requirements of this idea and try to figure out the best technological candidate related to that. Based on a chosen type of PUF, we will further discuss evaluation mechanisms necessary to classify the current PUF behaviour and computationally distinguish between malicious and harmless situations. Additionally, a conceptual architecture must be elaborated that can be used in real world applications trying to integrate the solution under different constraints. Besides these theoretic contributions, we will create a real implementation of the presented idea containing hardware and

software components. Using that, we can evaluate the performance for different attacks scenarios, allowing us to understand what types of attacks might be detectable and which not in a plausible way. Finally, this will also answer the question if our idea might be able to enhance the overall security or not and what kind of tradeoffs will be introduced thereby.

1.4 Structure of Thesis

For reasons of clarity the thesis will be structured into six dedicated chapters. Chapter 2 introduce the reader the basic terms and technologies required to understand the following chapters. Furthermore, it will include the most important related work according to the thesis. Chapter 3 explains how the solution is derived from previous work and our empirical observations. This chapter will also introduce the overall design of our approach, discuss the required components and how this solution might be applied in practice. In chapter 4 we discuss our implementation of the integrity monitor. This will especially cover which tools and technologies we have used, explain hardware and software components and finally discuss the most difficult parts of this implementation. Chapter 5 continue with a close look at the performance evaluation of this implementation. In greater detail it will discuss several experiments, the used setup and finally the expected and actual results. Finally the last chapter will summarize the achievements of this thesis, discuss answered and open questions and cover our thoughts about possible future enhancements.

2 Background and Related Work

This chapter introduces the reader main ideas, technology basics and all terms necessary to understand the following work. It will also reference the most important related work in academia and industry, explain who else is working on the topic and highlight differences between already existing approaches.

2.1 Field-Programmable Gate Arrays

A Field-Programmable Gate Array (FPGA) is a special type of integrated circuit that is still configurable by the customer even after manufacturing. The reader might imagine an FPGA like an array of configurable logic blocks, which are able to perform simple combinational functions like basic Boolean operations (e.g. AND, OR, NOT) and more complex ones. Furthermore, these simple, small blocks can be wired together to create more elaborated and bigger functions composed of multiple connected blocks.

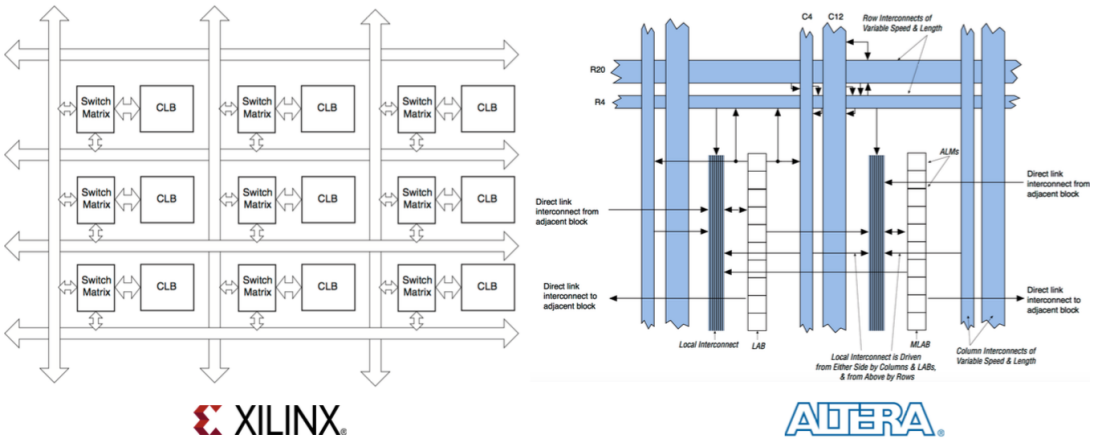

Figure 1: Schematic architecture of Xilinx and Altera FPGAs [18]

The configuration of such an FPGA is typically done by hardware description languages such as Verilog or VHDL. These languages are developed to describe concrete digital logic circuits within abstract modules. The compilation of those modules into a logical implementation is done with by a logic synthesis tools and hence called synthesis. Thereby each synthesis yield a fitting process, which is responsible for the physical placement of the single elements on a specific device, and finally generate a bitstream which can be programmed on the device, e.g. via JTAG. Indeed these steps are highly dependent on the working principle of the target system itself. As a consequence of this, each FPGA vendor is developing its own tools for system development. Today, the biggest vendors of FPGAs in the market are Altera, Xilinx and Microsemi. Each is using different approaches and architectures to facilitate this reconfigurable technology, while all of them are proprietary and only described roughly by the vendors.
The lion's share of FPGA systems are manufactured with volatile SRAM storage

technology. SRAM is preferred because it is easy to manufacture, have a great logic density, high power efficiency and is furthermore updatable. [47]
FPGAs are used in many scenarios like digital signal processing, computer vision, cryptography, communication systems and much more. Their clear advantage is their parallel nature which accelerates the computational performance while having a low power consumption. Another advantage is their flexibility and reusability because they can be reconfigured thousands of times. This way, FPGAs filling the gap between slow general-purposed computer and expensive full-custom application-specific integrated circuits (ASICs).

2.2 Physical Integrity

Confidentiality, integrity and availability are key principles in any security measure and solution. According to [9], confidentiality is generally defined as the prevention of unauthorized disclosure of information, availability describes the prevention of unauthorized withholding of information and integrity represents the prevention of unauthorized modification of information. Following this, the term physical integrity will be used in the following as the prevention of unauthorized modification in the physical domain addressing frequency, supply voltage, environmental effects or logic modifications.

2.3 Cryptography and Secure Devices

Modern computer systems does rely on cryptographic algorithms which ensure confidentiality, authenticity and non-repudiation properties. A cryptographic algorithm is thereby defined as a mathematical function that is capable to transform a secret plaintext (P) with a secret key (K) into an opaque ciphertext (C). This direction of processing is called encryption, while its inversion is called decryption. An essential property of this functions is that the decryption have to be unfeasible without knowing the key. Furthermore any cryptographic system is typically distinguished into symmetric or asymmetric systems. [46]

- In **symmetric cryptography** all communication partners share a common secret key used for encryption as well as decryption. A well-known example for this is the Advanced Encryption Standard (AES) or its insecure predecessor Data Encryption Standard (DES). [46]
- In **asymmetric cryptography** each communication partner has a key pair, composed of a public and a private key. A message can be encrypted and verified with the public key and decrypted and signed by using the private key. Examples for asymmetric cryptography are the Rivest-Shamir-Adleman (RSA) algorithm or elliptic curve cryptography (EEC). [46]

Breaking a cryptographic algorithm generally means revealing the secret key by examining any kind of information we can access [46]. A cryptographic algorithm is considered secure if there is no attack known that can break it within

a reasonable amount of time and resources. Cryptographic algorithms that are popular today will be typically considered *computationally secure*, which means breaking the system is possible in theory, but requires computing power that is not available. Related to that a fundamental principle, known as Kerckhoffs's law, states that that secrecy of a cryptographic algorithm should only depend on the secrecy of the key and never on the secrecy of the algorithm itself.

A **cryptographic device** is an electronic device that implement a cryptographic algorithm and hence has to manage secret keys. This device can be anything for instance a personal computer, a smart card, an electronic authentication token or an FPGA system. The options to store a secret on these devices in a non-volatile way are rather limited and can either be done with ROM (Read-only memory), Fuses (e.g. laser fuses, e-fuses), Flash technology or with battery backed volatile memory [1].

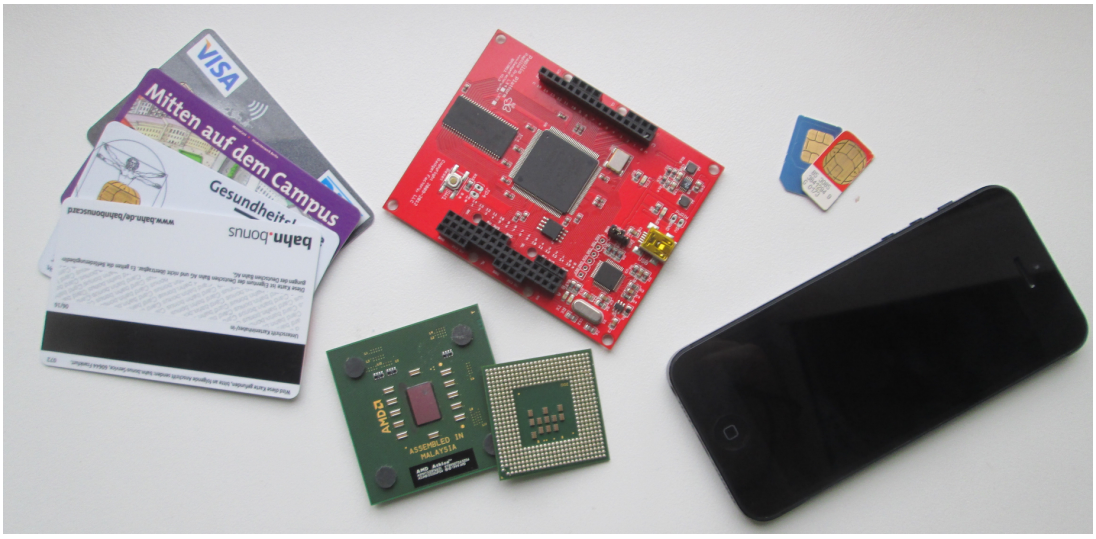

Figure 2: Examples of common cryptographic devices

Unfortunately these storage technologies and more general the overall electronic devices are rather insecure and can be attacked in various ways virtually as well as physically[46]. If successful an adversary can steal confidential data, poses as another person or harm asset owners in various other ways.

2.4 Threats and Physical Attacks

The following section present some of the huge possibilities of attacking secure devices. According to [40], many basic techniques are originally discovered by hackers trying to extract PayTV smart card keys, jailbreaking gaming consoles and tricking printer cartridges. However, since secure devices becoming more and more ubiquitous in modern life (e.g. payment cards, car keys, medical devices) many researchers and companies trying to enhance attacks and create new ways to do so. This way, several techniques have been found to attack a system

within the physical domain. Thereby each approach differs in terms of cost, time, equipment and expertise needed to accomplish it. According to [46], the different attacks can be categorised into active and passive ones:

- **Passive Attacks**: The device is operated largely or even entirely within its specification. A secret is revealed by observing physical properties of the device like execution time or power consumption.
- **Active Attacks**: The device, its inputs, and/or its environment are manipulated in order to make the device behave abnormally. Secrets are revealed by exploiting this abnormal behaviour of the device.

Additionally, [46] distinguish between attacks based on the exploited interface of the device into non-invasive, semi-invasive and fully-invasive techniques. All of these methods will still allow passive or active penetration.

2.4.1 Non-invasive techniques

Non-invasive techniques are by far the most common, just because of their simplicity and little costs. The attacker doesn't have to modify the target, neither the IC nor the package or carrier embedding the chip. Hence, this approach mitigate most kinds of evidence through physical damage.
A very popular example of passive, non-invasive techniques are side-channel analysis (SCA). By doing SCA the attacker is measuring the power consumption or computation timings and hence is capable to examine internal secrets. This is possible because different operations done by microprocessors having different power consumption profiles. This way essential operations used in cryptographic algorithms (e.g. squaring and multiplication) can be distinguished in measurement traces and thus leak internal information. Side-channel analysis based on power consumption and computation timing were introduced by Paul Kocher et al. in the mid-1990 [37, 38]. In [31] and [28] similar attacks are presented based on the effects of electromagnetic radiation and acoustics.
Another way of attacking a device in a non-invasive way are simple protocol based attacks. For instance we can attack the device communication by recording and replaying messages (Replay attack), withholding them (Jamming) or furthermore relay and alter messages (Man-In-The-Middle). Generally any communication interface will introduce new threats, like an unprotected JTAG port, which allow us to reprogram or debug a device and give us access to internal secrets.
An active, non-invasive way to attack the device are **Fault attacks**, which are trying to provoke faults into the computation of the device. Hereby the attacker might suspend the device to high temperature, unsupported supply voltage, excessively high overclocking, etc. to force the device to output incorrect results due to physical data corruption. Based on this faulty data a cryptanalyst can deduce executed instructions or reveal internal data states afterwards [22]. Other ways to do so have been shown by illuminating the device with UV light or irradiate it with electromagnetic or X-radiation [25].

Equally the exploitation of the system clock or supply voltage line will allow the attacker to introduce operational glitches. For instance, doubling the clock frequency for a few microseconds can cause a data misread or an instruction miss (clock glitching) and yield a possibility to modify the device's control flow, e.g. stepping over a branch instruction to yield a failed password check [25]. Equally a variation of the supply voltage level during execution, e.g. dropping it to ground for a few microseconds, may cause a processor to misinterpret or skip single instructions (voltage glitching) [40].
Finally, the last attack we will discuss in the non-invasive domain is based on the fact that device packages are typically transparent to special kinds of near-infrared light. This way a device can be scanned and examined with laser scanners, allowing us to create optical images of containing metal layers and deduce the internal device architecture [17]. The expertise and workload to do this may initially appear high, but many components on the device are purchased as IP and hence will be recognisable as regular structures to schooled attackers, allowing them to identify vulnerable logic within hours.

2.4.2 Semi-invasive attacks

Semi-invasive attacks involve to access the chip surface, but do not require penetration of the passivation layer or direct electrical contact to the chip internals [40]. For accessing the device package there are several methods proposed. One way is selective wet-chemical etching using fuming nitric or sulphuric acid, another way is to apply chemical mechanical polishing which might harm the chip through mechanical stress [42].
In [14] we can found an exemplary attack of passive, semi-invasive techniques. They developed a technique to read out multiple memory types, without using the provided read-out circuitry while keeping the stored value intact. To do so, they exploited low temperature data remanence effects, while using electromagnetic analysis and eddy current induction. Another approach, presented in [14] will use a laser to change the leakage characteristic of single transistors while simultaneously monitor the overall power consumption of the device. This way, the attacker can simply read out single bits of non-volatile memory.
A further way to attack is introduced by using photonic emission analysis. This kind of attack is exemplary used by [2] and requires a system that captures extremely weak photonic emissions from switching CMOS transistors. The traced emissions can hence be related to the code executed by the device and allow an adversary to identify internal device components like branching and execution logic and even to obtain a detailed physical map of the device memory. According to the paper, costs and required time for this kind of attack are comparable to common power analysis techniques, while a complete reversing of the target logic might be costly. Instead, the attacker can use this technique to quickly find weaknesses and interesting spots for succeeding attacks. In general, semi-invasive attacks are more costly compared to non-invasive techniques. A more comprehensive collection of semi-invasive attacks can be found in [42] and [17].

2.4.3 Fully-invasive techniques

Fully-invasive techniques are the most powerful and strongest type of attack an adversary may perform. There are actually no limits to what is done to the device and typically very expensive failure analysis equipment is used for this kind of attacks, including laser cutters, microprobing stations or focus ion beams [46]. Fully-invasive attacks evenly start by depackaging the chip. Hereby accessing the chip from the top is getting more and more unattractive because interconnection layers and an increasing amount of countermeasures and special material has to be evaded [17]. Hence, accessing the backside of an IC is more interesting today, at least because the interesting part (switching logic) is placed at the bottom of the chip [13]. Afterwards, different components of the integrated circuit can be accessed directly using a probing station. This allows an attacker to precisely contact and electrically stimulate any reachable conducting component of a device by using a thin needle. In doing so, passive attacks will try to directly monitor specific signals or wiretap the entire data bus. Finally active full-invasive attacks will grant full control to the adversary. He can alter accessible data or permanently modify single parts of the wired logic. A pretty interesting target in this domain are configuration and security fuses, which are widely used by manufactures to deactivate secret device functions originally implemented for testing and debugging. When editing the circuit, for instance with a focus ion beam, the adversary is capable to reactivate a secret debug mode or disable the copy protection of the memory. Some exemplary research in this field can be found in [17], [42].

2.4.4 Hardware Trojans

Another related topic in hardware security are hardware Trojans. As argued by [35], globalization of semiconductor industry is a big threat for secure devices. Many tools and steps in IC Fabrication Process are untrusted and cannot be verified by end-users. Hence, HDL level Trojans (inserted during design) and layout Trojans (inserted during IC fabrication) represent a serious threat for any security device. According to the taxonomy of [35], a Trojan can be activated either externally or internally in a combinational or sequential way and will for instance transmitting information or modify specific data and functions of a device. This way attackers are capable to eavesdrop internal and external communication or sabotage complete installations. Applied to IC used in military, financial or other infrastructure systems this will yield high economic and political consequences. The relevance of these kinds of attacks is documented in various reports like [27] and [43]. Concerning this, an unsolved question is how hardware Trojans can be defeated or at least detected? According to [35], this is a really difficult problem, but can partly be done with attack techniques introduced previously. One way is optical imaging of individual chips, which is quite expensive and not scaleable. Much better might be to find only one unbiased reference (golden chip), which can be used to compare large amounts of unproven chips by apply faster methods like side channel analysis. When comparing their

power consumption, timing behaviour and EM characteristic we might be able to identify some kinds of unauthorized modifications.

2.4.5 FPGA Security

According to [20] and [48] FPGAs are equally susceptible to all security threats that were previously described, which opens the door for unauthorized copy of data (cloning and reversing of intellectual property), tampering, spoofing and overbuilding of firmware to introduce malware and hardware Trojans. According to [47], Xilinx, Altera, and Microsemi FPGAs are also vulnerable to most side channel attacks and do not prevent extraction of keys. Furthermore, [44] is demonstrating that side channels such as power consumption and photonic emission can be used to measure individual logic elements typically used in FPGA applications, too. Indeed, simple optical imaging is not meaningful for FPGAs because most logic blocks look identical and only differ in their configuration.
The major drawbacks in FPGA security is the tough environmental assumption. The device may be in hands of the attacker with no trusted party available and hence cannot verify an identity like possible for internet security scenarios [47]. Attackers might gain physical access and can mount any attack just limited by their own expertise or resources. Additionally, if there is no backchannel, any defence mechanisms have to be implemented early and can not be updated or sanitised on already deployed systems.
According to [47], a big advantage of FPGAs compared to ASICs is the fact that final application designs are not revealed to vendors and suppliers, just because the FPGA is build without knowledge of the end application. However, this is only partially true: Indeed, it is imaginable that vendors may add "special components" for customers in general to trigger vulnerabilities and yield planned weaknesses during the life cycle like discussed in Section 2.4.4. Considering new features of modern FPGAs like partial reconfiguration, hardware Trojans are an even more dangerous threat to those systems. These technologies, like the Xilinx AXI Hardware ICAP, might be misused in near future to add malicious logic on already configured FPGAs even while they are operating in the field [50].

2.5 Countermeasures

While the last chapter introduced a broad variety of attacks, the following section will try to explain how countermeasures can be implemented against. Thereby in security practice it is not necessary to design a full-proof solution, instead a solution that makes analysis uneconomical for an adversary is often entirely sufficient [17]. Furthermore, not all countermeasures will directly prevent attacks, it is also feasible to detect them and raise an exception to reset the device or destroy secret data stored [17]. Those countermeasures are commonly distinguish into hardware and software countermeasures, while hardware countermeasures will further be divided into active and passive ones:

2.5.1 Active Hardware Countermeasures

Active countermeasures covering various detectors, which will be largely distributed on the device. They monitor the system clock, supply voltage or detect light of special wavelengths and yield alarms in suspicious situations. Another possibility are meshes, which are signal-bearing interconnects that are largely scattered on multiple IC layers. The constant evaluation of the mesh will validate the integrity of the device because common invasive techniques, using microprobing and focused ion beams might probably shorting the mesh to other signals [13].
Another way of active countermeasures is logic duplication. This idea does originally come from the area of fault-tolerant computing necessary for application with difficult environmental assumptions. In doing so, critical modules of the application will be implemented redundantly and their computational results will be compared against each other. When using multiple instances a majority voting evaluation might be applied to figure out the most probable result. [16]

2.5.2 Passive Hardware Countermeasures

Passive hardware countermeasures rely on shields, which can for instance block inbound and outbound electromagnetic radiation and hence reduce the possibility of successful attacks e.g. using EM based side channels or trying to introduce faults [16]. Furthermore, by using an unstable internal oscillator to generate the system clock, we can impede a possible takeover of the clock and make timing predictions for glitching attacks much harder [16]. Additionally data bus and memory encryption can be applied to mitigate probing of sensitive data or read out memory optically. Another way to cumber this kind of attacks is obfuscation of bus line and memory cell by scrambling their structure to confuse the attacker.

2.5.3 Software Countermeasures

Software countermeasures can be done one a generic system level or more specific on an algorithmic level. In [30] several ways are presented to defeat side channel attacks especially on an algorithmic level. The general idea is that the programmer has to break the relation between algorithmic values and processed values and hence make the computation independent from the key. In [8] they demonstrated a way of mitigation by using value blinding and random padding. Another way is done by removing any execution time dependence between data and key, for example by using dummy operations (operations with no effect) making conditional cases indistinguishable in time consumption. Moreover, obfuscation of the execution can be applied by shuffling independent operations to make a correlation more tedious.
Because, hardening the algorithm itself is an elaborate and error-prone proposal, more generic solutions are proposed in [30]. To increase the noise of side channel measurements we can introduce random timing variations in processing by using timeouts or by randomly inserting dummy instructions in the computation.

To circumvent fault attacks one can also duplicate logic by simply executing single operations multiple times and compare their results or create redundant variables and check frequently their integrity [8]. A more general approach to validate data integrity might use checksums, which can be applied for instance to evaluate the integrity of an whole FPGA configuration [8].

According to [17], the combination of software and hardware delays, masking and dummy rounds to yield obfuscation and randomisation of the execution flow will be the most effective way do defeat multiple kinds of attacks today. While many attacks remain possible in theory, it requires many measurement iterations, large acquisition times and serveral executions for an successful attack.

2.6 Physical Unclonable Functions

A physical unclonable function (PUF) is defined in [41] as an expression of an inherent and unclonable instance-specific feature of a physical object. This intrinsic feature idea has a strong equality to biometric features of human beings used today for authentication processes, like fingerprints, palm veins or retina. Similar to this, PUFs use manufacturing process variations and intrinsic difference of ICs to create a unique and unpredictable function on each device.
To do so, a PUF has to fulfil some general requirements, like described by [41]:

- A PUF is a physical entity that has to be embodied by a physical device
- A PUF has to be easy to evaluate and hard to predict
- A PUF has to be easy to create, but practically impossible to duplicate

Considering that, a PUFs should be able to return an reproducible, random output (response) depending on a given input (challenge). If satisfied, PUF are imagined to be used in a challenge-response based protocol covering two phases: In a secured environment, the enrolment phase take place, where huge amounts of challenge-response pairs (CRPs) are collected and stored in a database (CRP database). After deployment of the device, the verification can be done by comparing an actual response with the stored response of a challenge contained in the CRP database.

Figure 3: The PUF principle

Based on their unclonability and other useful properties, PUFs can this way fulfil a number of physical security objectives [41]. For instance, an often referred application is secure generation of uniformly random keys. A truly random function linked to physical source of randomness will represent a promising source of entropy to generate cryptographic secure keys. Because of this, we can even use the PUF as a secure key storage. This means, instead of storing a secret key directly in a traditional way, we can apply a set of challenges and compute a secret key based on the related responses. This key might then be used for a cryptographic operation and can be securely removed afterwards. This way, the key is only present in the device for a minimal amount of time and stored within the device intrinsics otherwise [1]. Another application is device fingerprinting or system identification. To do so, some random challenges from CRP database can be applied to the PUF by an proving entity, which might then compare the actual response with the pre-sampled one from the database. If matching, one can authenticate the device with a high probability that its not faked or spoofed.

In general, the approach of intrinsic PUFs can either be realized with non-electrical phenomena, like optical, magnetic and acoustic PUFs or by measuring electrical phenomena like Coating or LC PUFs will do [1]. While all of them are undoubtedly interesting, this thesis will focus on digital intrinsic PUFs for ICs, related to the electrical PUF section. These digital intrinsic PUFs will completely integrate the measurement system into the embedding device and have to be constructible with current manufacturing technologies. Considering this, two promising ways have been found so far: Delay-Based Intrinsic PUFs and Memory-based Intrinsic PUFs.

2.6.1 Memory-based PUFs

SRAM and Butterfly PUFs are main examples for intrinsic memory-based PUFs. An SRAM PUFs will exploit the bistable behaviour of SRAM cells, caused by MOSFET mismatches. SRAM memory cover an array of cells connected to read and write logic where each cell stores a single bit of information. One cell include two cross-coupled inverters and two additional transistors for the external connection, hence six transistors in total. On power up, each memory cell tilt repeatable into one logical state: a logical 1 or 0. This leads to a completely random start-up behaviour of SRAM cells, which can be exploited to create a PUF. To generate a PUF response, one can simply challenge or address the SRAM memory, e.g. by reading out a particular location within the memory. [1, 12]

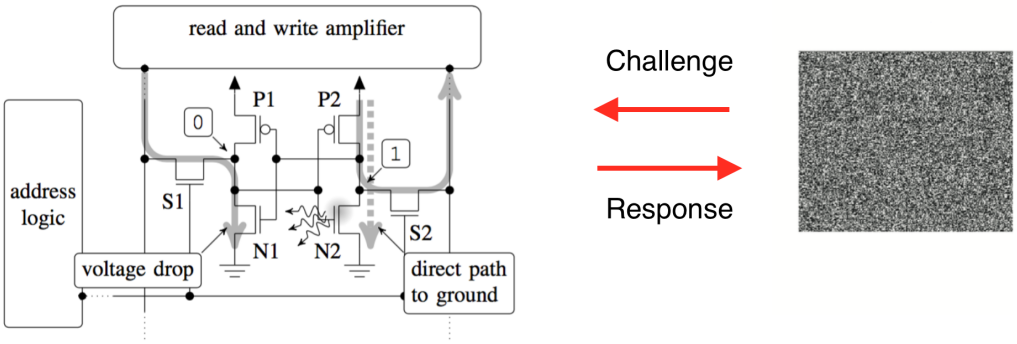

Figure 4: SRAM cell architecture [12] and PUF behaviour of SRAM memory, where black pixels represent a 1 and white pixels a 0 response [1].

2.6.2 Delay-based PUFs

Arbiter and Ring Oscillator PUFs are examples for intrinsic delay-based PUFs. A Ring Oscillator PUF is composed of an oscillating loop constructed by serially connecting an uneven number of inverter elements and feeding its last signal back to the starting point. Because of the intrinsic delays of these elements, this circuit will generate a oscillating signal with a partially random frequency. [1]

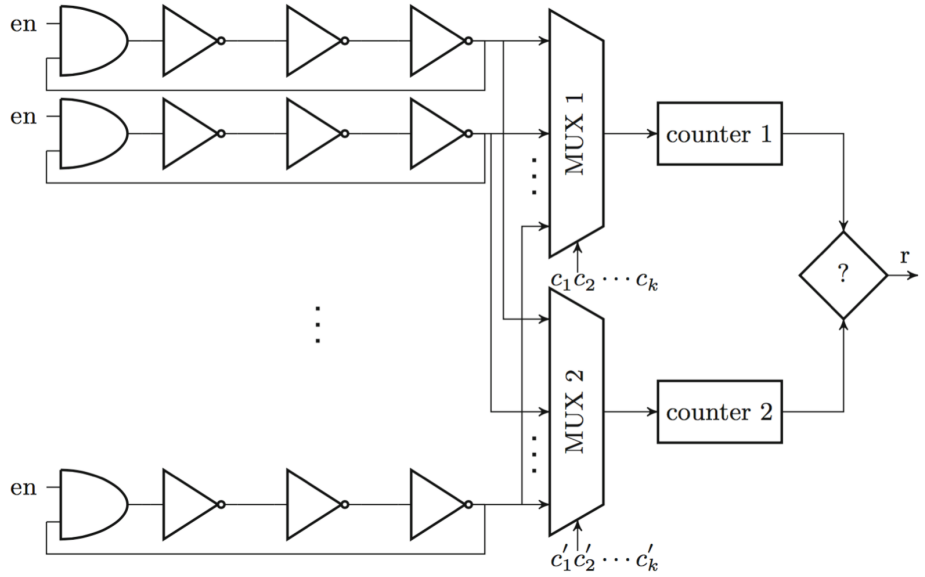

Figure 5: RO-PUF architecture [23]

A challenge response based PUF can be derived by building two equally sized sets of these ring oscillators and connect them with two multiplexer components. By applying two distinct challenges on their select inputs, always two ring-oscillators are chosen, leading to switch their outputs to the clock inputs of two succeeding counters. Hence these counters will count the number of rising edges during a predefined sample time. Finally, a comparator matching both values will then generate a binary response. [23]

2.6.3 Advanced Concepts

Indeed, PUFs are not perfect in reality. Many physical and environmental effects (e.g. temperature, supply voltage, electromagnetic interference, flicker noise, random effects) are influencing an electronic device and equally disturbing the PUF. Depending on the design, this noise can be larger or smaller and make an implementation unrealistic in reality or require further steps. According to [41], fuzzy extractors know from biometric technologies can be used to sanitize noisy responses. Other implementations, like [32], trying to compensate those effects with good results by using differential elements.
Furthermore, even in theory none of the presented PUFs satisfy the requirements to cover all imagined PUF applications at all. Thats why, PUFs are generally categorised into strong and weak PUFs in the literature. The fundamental difference between both is the number of unique challenges that can be meaningful processed. Weak PUFs only support a small number of challenges (or just one), while strong PUFs support huge amounts of challenges, which makes a complete iteration of all challenge–response pairs within a limited time frame not feasible. For that reason, strong PUFs are commonly imagined for all types of applications, while weak PUFs can just be considered to perform as secure key

storages [10]. The already presented SRAM and RO PUF are major examples for weak PUFs, while Optical-, BR- and Arbiter PUFs are representing strong PUFs [10, 11].
Concluding all these constraints, a really difficult question is: How to build a PUF that satisfy all these requirements, will work on multiple devices and will be stable under various environmental conditions? Regarding to that, many ideas to construct proper PUFs have been proposed in recent years, but the perfect solution has not been found so far. However, the following section will describe some more advanced concepts, which will also build the foundation of our solution.

2.6.4 Bistable Ring PUF

The Bistable Ring PUF (BR-PUF) covering both properties of memory- and delay-based PUFs and was proposed and evaluated in [11].

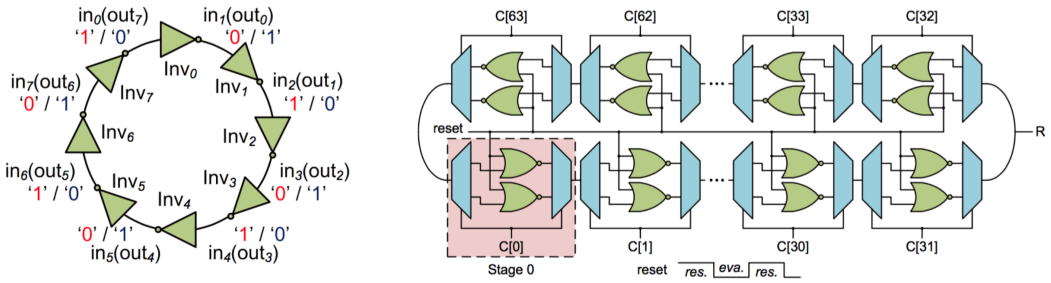

Figure 6: Two possible states of an eight-stage bistable ring and BR-PUF architecture with 64 stages. [11]

The basic idea of BR-PUFs is based on bistable rings, consisting of an even number of inverters arranged in a ring loop. Similar to an SRAM cells, the ring will certainly fall into one of two possible stable states, when powered up. For instance an 8 stages ring, like shown in Figure 6, can stabilise in a "01010101" or "10101010" state. Tapping one point of this ring, a primitive BR-PUF can this way generate a single binary response. To generate an exponential number of CRPs the idea will be augmented by building a longer chain (e.g. 64 stages). As explained in [11] the inverters will be duplicated and each pair will be complemented with a multiplexer and demultiplexer to select one of the inverters and switch it into the overall inverter loop. As usual the inverters are build out of NOR logic elements, to facilitate enable and disable operations. By building 64 stages, a 64-bit challenge can be applied to select a unique combination of logic gates, yielding a unique response each. Furthermore, this architecture can be easily scaled by adding or removing stages yielding also a larger or smaller challenge space. [11, 15]

2.6.5 RO Sum PUF

In 2010, [32] introduced a more advanced ring oscillator PUF deduced from the idea of recombination. They explain that in the area of genetic engineering, recombination describes the process of rearranging and merging genetic material to create new characteristics not previously present. In a similar way, hardware intrinsics may allow recombination to obtain new characteristics not natively present in the origin circuit. By applying this methodology, they augment the basic ring oscillator PUF to a new PUF called RO Sum PUF, like presented in Figure 7.
This design use mainly several ring oscillators subdivided into k pairs, with each pair having a certain frequency difference δf. This δf will be measured with individual counters for a previously defined sample time. Furthermore, each counter value will be accumulated into a sum S, while doing this the challenge will determine if a counter will be added or subtracted from this accumulator. A final thresholding step will then binarize the accumulator to a response bit, by validating the accumulator ≤ 0. [29]

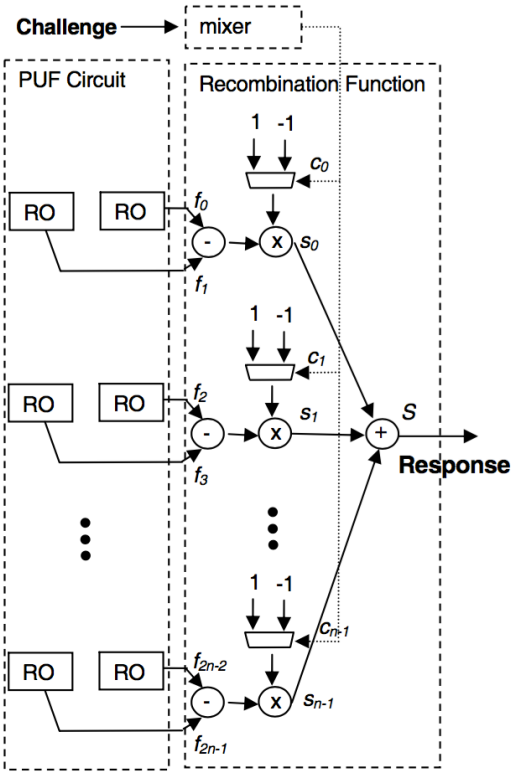

Figure 7: RO Sum PUF architecture [32]

According to [32], the design features a large challenge space (equals 2^k [29]) which was not present before. The randomness was successfully tested against NIST specifications, based on an implementation with k = 64 across 4 devices. This measure, failed by standard oscillator PUFs, has validated that the RO Sum PUF can reliably be used for trustworthy random data generation. Additionally

the thresholding and differential component will strongly de-biasing the PUF output yielding a superior bias compared to standard PUFs including oscillator, arbiter and memory PUF. A further advantage of the implementation claimed by [32], is the additional real-valued output S, which can be used for advanced error correction and further applications like Trojan detection. Concluding all these properties, the RO Sum PUF is suitable for all standard PUF applications and provides ideal PUF properties and great routing variability. [32]

2.6.6 PUF Security

Obviously, even PUFs are susceptible to non-, semi- and fully-invasive attacks. This way, some vulnerabilities has been revealed by researchers in recent years. In [23] a machine learning technique (probably approximately correct learning) is used to model RO-PUFs and predict their behaviour. In addition, even more sophisticated PUFs show mathematical weaknesses: In [15] differential and linear analysis has been used to show correlations in the CRPs of BR PUFs (on FPGAs), which can help an attacker to predict special quantities of responses. Equally [32] is talking about this problem in case of recombination functions, when the functions are too simple.
As usual, by using semi- and fully invasive techniques an attacker is able to do much more. In [45], a laser fault injection attack is used to manipulating the configuration of an FPGA. In doing so, the effectiveness of modelling attacks can be increased and the entropy of responses can be decreased dramatically. Another example is shown in [33], where EM analysis is used to identify single ring oscillator frequencies of an RO PUF, making its behaviour predictable. Finally, even cloning is possible, like shown in [12] for SRAM PUFs. This could be done by characterising the intrinsic PUF features with photon emission analysis while respectively modifying another PUF with several FIB circuit edits.
However, in comparison to traditional solutions PUFs are not broken by default and still provide a high security level, especially when combined with countermeasures presented in Chapter 2.5. Furthermore using intrinsic properties might enable us to detect compromise of the IC package, while altering the IC and hence its intrinsic features will at least result into a non-functioning device [17].

2.7 Related Work

2.7.1 Vendor Countermeasures

Meanwhile, to tackle the problems of security breaches, counterfeiting and cloning the main FPGA vendors have introduced their countermeasures. As usual, these are proprietary technologies and nobody except the vendor itself knows precisely what and how it is done [47]. At the end of the day, researchers like [5, 6, 24] have to reverse parts of these implementations to reveal the secrets. Quite often, this research is showing that solutions are insufficient, what probably tell us another reason for implementation secrecy. However, in the following we will discuss these concepts introduced by the industry related to our work.
To mitigate IP-theft all vendors equally agree on bitstream encryption technology using AES or 3DES. This way any application design can be encrypted and afterwards safely transmitted and stored on the FPGA. To finally use this bitstream, the FPGA has to decrypt it. That's why a decryption key has to be stored on the FPGA device itself, which yields some already presented problems [5], [6]. Furthermore, the vendors introduced authentication based on asymmetric cryptography. This way a developer can additionally sign the encrypted bitstream by using a private key. To authenticate a stored bitstream on the device, we can verify the signature by using the corresponding public key [21].
Beside encryption, also passive and active security features are built into FPGAs nowadays. Passive ones do not require the user to do anything, while active ones have to be added by the user into the logic design [51]. Generally, this is equally done by all considered solutions (Xilinx, Altera, Microsemi) by releasing a security related soft IP core, which allows the user to configure individual layers of tamper protections and tamper responses.
Tamper protections are capable of monitoring an assortment of internal security flags and system conditions. To do so, the vendors using multiple detectors directly built into the FPGA silicon. On detection, the security core can report threats and/or react autonomously. Reports will just announce internal security flags and system conditions, while an autonomous approach can initially respond and initiate tamper responses [34]. Tamper responses are mitigation routines the device can perform in order to prevent misuse or modification of the device and mitigate disclosure of critical data contained within [9]. In practice, this is done by partially or fully zeroization of the FPGA configuration removing any application design information and stored secrets. Additionally a lockdown mode can be enabled, pulling the I/Os to high impedance and blocking any JTAG access.
In [34], Microsemi is introducing their security IP block, called EnforcIT Security Monitor, available for some of their FPGA systems (SmartFusion2, IGLOO2). The monitor can be configured for reporting or act on its own. In detail, the module includes a JTAG monitoring application, a clock frequency monitor, a system heartbeat function and a watchdog timer. Other features are customizable clocks, timeout monitoring, logic integrity and fault detection as well as a runtime IP version reporting option.

Microsemi promises that the core will only use less than 5 percent of the FPGA device in a typical configuration and can be implemented without noticeable system costs or development overhead [34]. To arouse trust, they promise that all software, and hardware products covering EnforcIT Security Monitor are built in the U.S. in their trusted facilities, and system specification fulfil the requirements of the NSA's Commercial Solutions for Classified Program. [34]

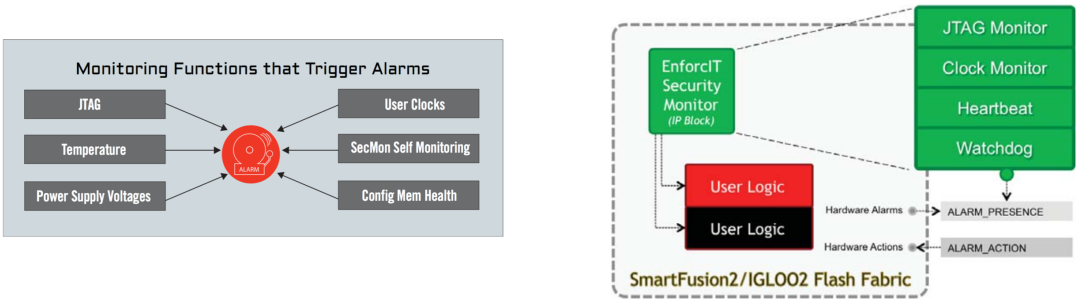

Figure 8: Schematic description of SecMon and EnforcIT system [51, 34]

The solution proposed by Xilinx is called Security Monitor (SecMon). They support many platforms like Virtex-5,6,7, Spartan-6, Zynq and all of their programmable SoCs. SecMon is working only autonomously and back monitoring functions for configuration and memory integrity as well as temperature, voltage, user clocks, and JTAG activity. Furthermore, they support monitoring for partial reconfiguration. In future Xilinx will allow multiple SecMon IP cores to be connected together, acting as a single unit on board level (device-to-device) and stacked silicon interconnect (SSI) devices. SecMon is delivered as a fully placed-and-routed design file and just have to be added to an existing design. The calculable resource impact is given with 60mW power draw and a maximum of 8% resources usage in the worst case. Xilinx's implementations are validated and evaluated by NIST-accredited security testing laboratory [51]
Altera's solution, called Security Supervisor IP (SSIP), supporting only their Arria V and Stratix V systems. It equally represents dedicated logic module used to set up security configuration and responses. The unique feature of SSIP is, that is has been designed into a separate region of the FPGA called logic lock. This region is able to interact directly with the device configuration block and have low latency access to device monitors, sensors and the partial reconfiguration control block that is used for zeroization by them. They argue, that this separate partition is also essential to make sure that zeroization only overwrite the area of interest and not the SSIP logic itself. Furthermore they cover monitoring functions for configuration errors, key values and states, temperature and heartbeat signals. SSIP is licensed and certified by the United States government and following secure design guidelines provided by some unnamed authorised government sponsors. [3]

2.7.2 Hardware Trojan Countermeasures

Besides that, also many researchers tying to develop strong countermeasures to mitigate several kinds of attacks, especially in the hardware Trojan domain. The most important approach, related to our work is shown in [53] and [36]. In booth papers the authors trying to address the problem of Trojans by presenting a method to distinguish between Trojan inserted ICs and Trojan-free ones. The overall principle is based on the fact that ring oscillators are sensitive to power fluctuations and local effects caused by surrounding circuit switching activity. To exploit this, [53] build a structure called ring oscillator network (RON), allowing to compute a signature over multiple oscillators and uniquely characterise the device activity in a certain area. To cover the whole chip, the ring oscillator components will be placed in different rows of a standard-cell design to monitoring a chip section each. A statistical evaluation of these networks will finally allow to identify biased and unbiased devices because their signature will be different.

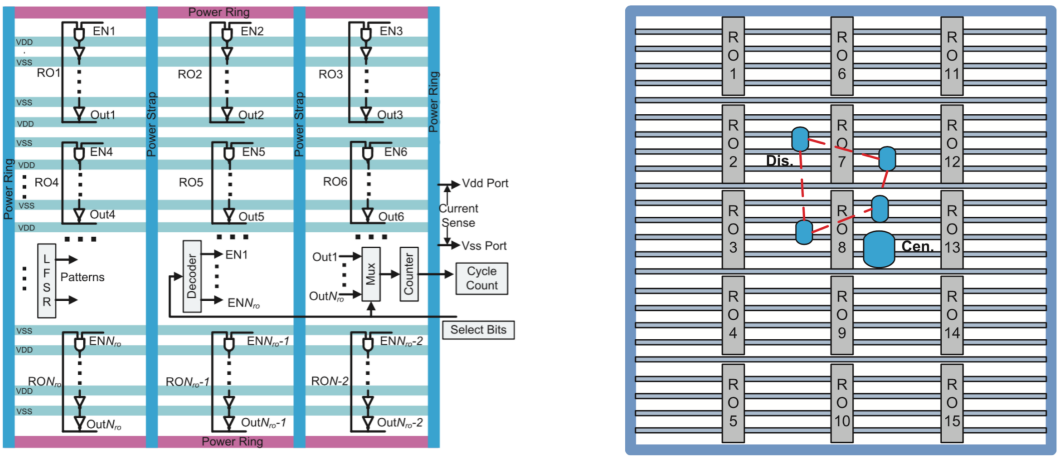

Figure 9: RON design and its distribution on an FPGA [53][36].

In [53], the researchers use statistical data analysis to separate between effects resulting from process and environmental variations from those caused by Trojans. Their analysis cover simple and advanced outliner methods as well as principal component analysis (Further explained in Chapter 3.2). To demonstrate their detection capabilities they use simulations and finally test the approach on an Xilinx Spartan-6 FPGA. Instead, [36] extended the work of [53] by investigating a machine learning approach for diagnosis. Thereby they present a genetic algorithm for feature selection and evaluate this against the methods used in [53] with good results.

3 Design

The following chapter will discuss how existing approaches are used to create a novel solution and explain the main achievements of this thesis. Furthermore, we will discuss the application requirements and present a generic architecture for our solution, which will explain how the application can be realized in real world scenario with several constraints.

3.1 Intrinsic Integrity Monitor

Traditional countermeasures, like introduced in Chapter 2, suffering from the fact that sophisticated attackers with decent skills and resources, might always spoof, thwart or circumvent these methods. For instance, a successful circuit edit can disable or corrupt any sensor, its evaluation logic or the mitigation response, while finally taking over control of the data bus will circumvent any software countermeasure as well [17]. Considering this, the question we have to ask is: How can we create a solution that is capable of protecting the physical integrity of a device like traditional methods trying while somehow link this monitoring component to an intrinsic properties of the device. Considering this, a PUF architecture seems like a good foundation to solve this issue and will even provide other features beyond that. PUFs are intrinsically linked and not predictable in general, which makes them resilient against attacks trying to tamper with. Furthermore, they get much attention by researchers and are well-studied technologies, which will ensure us extensive information resources and high acceptance. By investigating this initial idea, we have discovered that some basic PUF components and design artefacts are indeed sensitive to several physical effects.
Ring oscillators are one example of these components, which are often used to create delay-based PUFs like the RO-PUF. As presented in Chapter 2, ring oscillators are known sensitive to power fluctuations and local effects caused by circuit switching activity nearby. Considering this fact and the work presented in [53] and [36], we can assume that ring oscillator based PUFs might have a sensitivity for those and other effects related to the device security. Another example for this kind of sensitivity can be found for bistable rings like used in BR-PUFs. From our observations, this architecture is also quite sensitive to temperature changes and its behaviour can be correlated with reference measurements. Hence, it might be possible that further environmental modification introduce similar changes, too.
Concluding this observations, one can argue that some PUFs may be able to "observe" their physical environment much more precisely than normally considered. A general assumption about their physical entrenchment is thereby often expressed as: Altering the IC or package, will equally alter the physical intrinsics and hence make the PUF behave suspicious. Indeed, this assumption is slightly incomplete. On a closer look, most semi- or fully-invasive techniques will trigger this modification, but how about non-invasive ones? These techniques will

not alter the IC or package by definition and hence do not affect the intrinsic properties, but can still be used to attack the device. For instance, even a PUF generated secret will still be present in a device for a short time during execution and can be attacked in various non- or semi-invasive ways. Furthermore, also glitching attacks might still be an option to cause data misreads or leap instructions. Moreover, by evaluating the already exposed attacks on PUFs (Chapter 2.6.6) and reviewing the inter-noise statistics of current implementations, not all semi- or fully-invasive attacks will guarantees a strong enough modification of the PUF to make it suspicious.

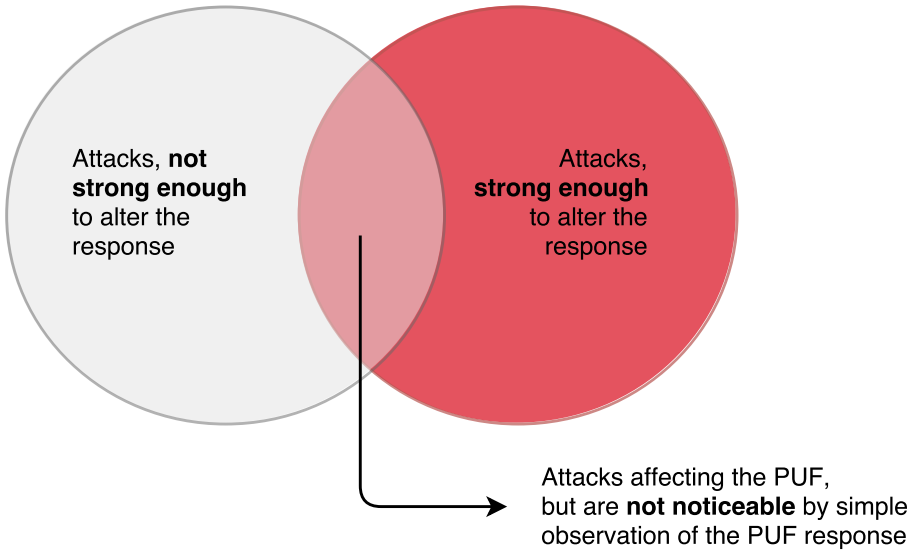

Figure 10: Extended PUF sensitivity model covering behaviour changes normally to small to make the PUF suspicious.

Concluding this issues, no matter what kind of attack will happen, we have to consider the magnitude. That's why, we would like to extend this so far binary assumption by an additional impact factor like shown in Figure 10. Attacking a device will not always change or keep its intrinsic properties, it can also have a very slight effect, which is only perceptual on a closer look. Additionally glitching and temperature attacks as well as Trojans will not influence a device always in a permanent way. They can occur in one moment and disappear in another, which means we have to consider the temporal resolution as well.
Indeed, not all PUFs will be capable to satisfy this model. Hence, we have elaborated a set of additional requirements that have to be satisfied by a PUF architecture trying to fulfil the task of monitoring the physical integrity:

- Strong PUF: The architecture should support a large challenge space to prevent behaviour prediction and allow support of origin PUF applications like authentication.

- Physical Sensitivity: The architecture must embed components that are sensitive to several physical phenomena which can be measured in an adequate way.
- Routable: The architecture must be able to be distributed over the whole IC to allow detection of local effects like introduced by laser and Trojans.
- Participation: Each measurement component of the architecture must equally contribute to the PUF response to ensure that local effect will not be outweighed by stronger components.
- Time Resolution: The architecture must allow a permanent observation of the device to detect non-permanent changes. Furthermore, the components should be evaluated simultaneously.

Based on our observations and the given requirements, we have figured out two key architectures, namely BR-PUF and ring oscillator PUFs like RO- and RO Sum PUF. In the following we will present our considerations about what will be the best option for our purposes.
Each architecture have a good routing variability, which makes them easy to distribute on the chip and provide the required susceptibility for local effects. Ring oscillator, in contrast to bistable rings, are well-studied components and a mathematical model as well as their environmental dependencies are given in the literature (e.g. [53], [36]). Furthermore, the naive implemented BR-PUF will only return a binary response and does not contain any further information. A far better option will be a real-valued output, which allow us to see even small changes on close observation. Such a real-valued response is natively available for RO Sum PUFs and can be easily derived for RO PUFs by using the counter difference before thresholding. Because of that, the ring oscillator PUFs will represent the most promising architecture so far. Thereby the default RO PUF is weak and hard to scale, because any duplication of the challenge space will yield a duplication of the ROs ($2 \cdot 2^m$ ROs required for m-bit challenge [39]). In contrast to this, the RO Sum PUF is strong and easy to scale ($2^{ROpairs}$), which will allow us to use it additionally for authentication and device fingerprinting. Finally each challenge applied to a RO Sum PUF will include an evaluation of all corresponding ring oscillators and yield a detection of local effects immediately. Instead, a standard RO PUF will require to iterate all challenges in the worst case to trigger the ring oscillator which might be suspicious.
Considering these arguments, the RO Sum PUF represents the best solution so far for an intrinsic monitoring component. Indeed, regarding to [32] a more advanced recombination functions may be developed, yielding new PUFs containing further components which increase its sensitivity to physical attacks. However, the RO Sum PUF in its origin form satisfy all stated requirements and will even allow additional applications.

3.2 Evaluation Concept

To monitor the physical integrity, we will examine the real-valued response (S) of the RO Sum PUF and evaluate its behaviour. To do so, we will slightly orient us on the approaches presented by [53] to evaluate their ring oscillator networks, which will be summarised in the following:

- Simple Outliner Analysis: The counter for each ring oscillator is normally within a certain range for unbiased ICs. If this value is outside the range, the IC is considered suspicious. This method is not considering relationships among ring oscillators, nor external measurements.
- Principal Component Analysis: This method will consider all ring oscillator counters as well as the externally measured power consumption and transform them into uncorrelated variables. By using the first three principal components, we can create a convex hull representing the golden chip behaviour. If a device under test is beyond this hull, it is considered suspicious.
- Advanced Outliner Analysis: This method will similar consider all RO counters and the power consumption to create $N \cdot (N-1)$ signatures for a golden chip. To do so, we can calculate the sum over all ring oscillators and furthermore take each possible combinations of two ROs (Ci, Cj) to create a signature like: $((sum - Ci)/Ci; (sum - Ci)/Ci; I)$. If one of these signatures of the device under test does not match, then it is considered suspicious.

Obviously this approaches cannot be easily transferred one-to-one into our solution. Besides the external measurement, we might add a dedicated evaluation logic beside the recombination logic of the PUF, but this is not necessary at all. According to [53], the best working approach in terms of detection rate is the advanced outliner analysis. Its superiority is caused by the fact that the method will consider the relationships between RO pairs, which is not covered by the simple outliner analysis yielding a lower detection rate. An analogous evaluation is already present in the default RO Sum PUF. Instead of comparing any combination of RO pairs, we just compute the difference between static pairs and sum these values according to the challenge. To compensate the missing relationships, we will do this operation for multiple challenges and furthermore take into account their relationships in our placement strategy (Chapter 4.3.2). This way we can collect an expression which can be threaten equally like the signature purposed in the advanced outliner analysis by [53]. The permanent observation of this expression will finally allow us to distinguish between unbiased (within a certain range) and biased (out of a certain range) device behaviour. Indeed, this out of bounds approach can be done in different ways:

- Min/Max
  The min/max approach will compute the minimum and maximum value of the unbiased data record R_{DB} and use these values to determine the boundaries for unsuspicious responses. Any response value r that doesn't satisfy the condition $r \in [min(R_{DB}), max(R_{DB})]$ is hence interpreted suspicious.

- Mean $\pm$ σ
  The standard deviation approach will compute the mean value (μ) and the standard deviation (σ) of the unbiased data record R_{DB} and use these values to determine the boundaries for unsuspicious responses. Any response value r that doesn't satisfy the condition $r \in [\mu - 3\sigma, \mu + 3\sigma]$ is hence interpreted suspicious. To change the acceptance threshold, one can lower the factor preceding σ (e.g. 2σ) to enhance the overall attack sensitivity, but this will also increase the amount of noisy responses.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} r_i, \qquad \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_i - \mu)^2} \qquad \text{with } r_i \in R_{DB}, N = |R_{DB}| \quad (1)$$

To make this procedure more understandable, we will cover a small example in the following: For instance we might record the values R_{DB} = [10,11,10,10,11,8] for a challenge c_1 during enrolment phase. This will result into boundaries of $[8, 11]$ for the min/max approach and $[10-\sqrt{\frac{6}{5}}, 10+\sqrt{\frac{6}{5}}]$ for a standard deviation approach. Any value outside of this ranges will hence be marked as suspicious response during the verification of our PUF.
Additionally, our evaluation method have to deal with noise, which is typically present in PUF applications through temperature changes, unstable supply voltage, EM interference and many other effects. To do so, we will implement two common techniques to take this into account:

- Majority Voting
  We will repeat a single measurement multiple times and rate the PUF behaviour based on a majority voting process. For instance, when applying one challenge 5 times each, we will require 3 and more responses to be out of bounds to finally yield a suspicious rating. Less than 3 will mean we cannot safely decide, so we have to do more iterations or ignore it and continue the evaluation.

- Detection Threshold
  Furthermore, we have to break down the decision if an attack is happening or not during the evaluation by analysing all the collected responses. This can be done by determining a fixed threshold value for the so far statistical analysis. Based on previous evaluations, a good threshold value might be 20%, which is small enough to keep a good sensitivity for slight effects, but is still high enough to remove any kind of distortion we could observe. Alternatively, we can also output an attack possibility by returning the percentage of suspicious responses.

3.3 Architecture

The general architecture of the integrity monitor can be implemented in two ways, offline and online. Thereby, both options will require the creation of a database covering several challenges and their related real valued responses. This process must be done before deploying the system and requires a secured environment and data storage. Thereby, large CRP databases can easily store millions of CRPs related to thousands of devices, while a local database stored directly on the device may only carry a few hundred CRPs caused by the limited resources. Furthermore, the communication with such a device and hence the PUF has to be confidential because attackers recording CRPs may be able to learn or imitate a PUF otherwise. This might be done easily, for instance by using secure key storage and fingerprinting mechanisms equally covered by the RO Sum PUF.

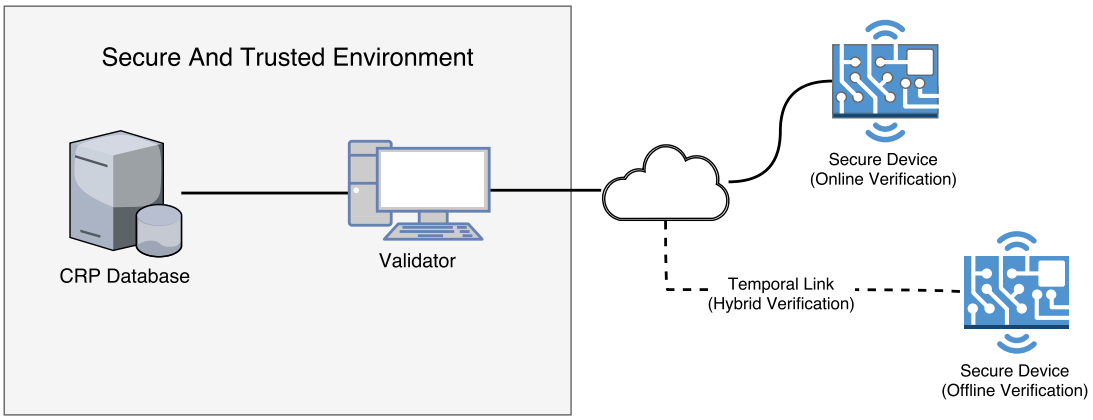

Figure 11: Schematic description for online, offline and hybrid verification.

- **Offline verification:** To monitor the physical integrity in offline only scenarios, we have to store the CRP database on the device itself. Based on that data we can trigger a verification routine in periodic intervals, for instance each 10 seconds or permanently. In doing so, the routine will compare the actual measurements against the previously stored ones. If the data look suspicious, the device can autonomously start a mitigation response to delete all secrets, stop critical operations and block I/Os.

- **Online verification:** This type of verification is more complex and require a connection to the verification instance. A remote validator entity can choose a set of random challenges from the CRP database and send these to the device under test. The received responses will then be compared by the validator, which can now figure out if a device is suspicious. Depending on the type of application build, such a validator may degrade the trustworthiness, exclude the device from the system or trigger some alarm. This approach has a big advantage, because the CRP database is

placed in a secure and trustworthy environment and can be protected from local attacks. On the other hand, adversaries will try to target the communication protocol by replaying, withholding or modifying communication data, which has to be mitigated in this scenario.

- **Hybrid verification:** By examine the previous scenarios, we can also imagine a hybrid approach. In such a case a device may be self-monitored (offline) in general, while also has to be validated by the external instance (online) from time to time. This may allow to update a local CRP database on a device or reactivate devices that are locked down by an offline monitor.

However, for this thesis we will focus on a much simpler architecture to initially evaluate the proposed solution and evaluate its performance in multiple attack scenarios.

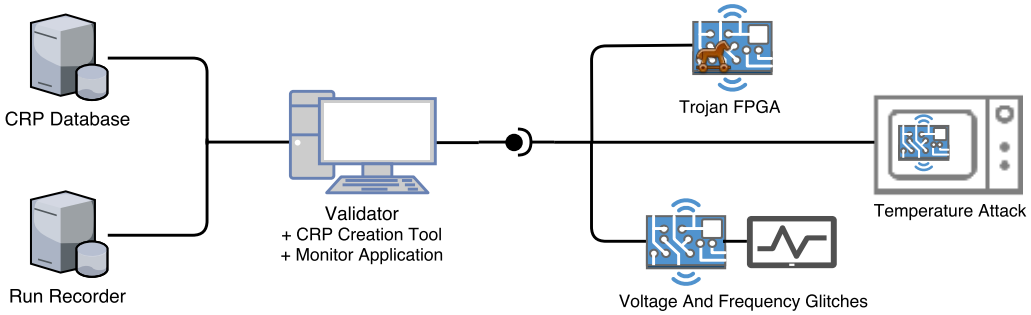

Figure 12: Separation of concern: Validation and database logic (software) are decoupled from PUF logic (hardware)

To be as flexible as possible we will logically separate the PUF design from the application design responsible for monitoring and PUF enrolment. This loosely coupled architecture will allow us to implement alternative solutions, which provide the same service and can be replaced by each other. For this reason, the FPGA will only contain the PUF, a basic communication module and some further elements required for testing. Separated from this hardware implementation, any other component (e.g. monitoring and enrolment application) will be done in software and executed externally. This will enable a flexible way to explore different methods for evaluation and avoid time-consuming hardware implementations. Nevertheless, in future each of these operations can also be done directly on the device like considered for offline verification.

4 Implementation

In this chapter we will present the most interesting parts of our implementation work. It will give the reader a deeper understanding of the techniques used to build an integrity monitor like purposed and will point out the most difficult parts of the implementation one have to solve.

The source code and all related project files created during implementation can be found in the dedicated Git repository: `https://gitlab.sec.t-labs.tu-berlin.de/jfietkau/TamperEvidentPUF.git`

4.1 Development Environment

The software have been built and tested on Mac OS X El Capitan 10.11.4 (15E65) using Sublime Text 2 and its integrated development environment. The implementation requires a decent Python 2.7 interpreter and listed extensions.

For hardware development we used 2 FPGA devices and their related IDE:

(A) DE0-Nano development board embedding the Altera Cyclone IV FPGA (EP4CE22F17C6N) build with 60-nm process technology.

(B) Custom board embedding a smaller version of the Cyclone IV FPGA (EP4CE6E22C8N), which is depackaged and prepared to be used for laser experiments and also build with 60-nm process technology.

Altera devices can be configured with the proprietary Quartus II design software. We are using the free available Quartus II Web Edition which has a reduced feature set, but is almost comfortable to use.
» Available at: `https://dl.altera.com/?edition=lite`

Because the implementation is partly created in hardware and software, our application requires a communication channel between the FPGA and the computer running the application. Therefore, we are using UART, which is an asynchronous, serial communication protocol. To do so, we will use an additional USB to UART converter (BTE13-007 CP2102) which have to be connected manually to the FPGA I/Os and the computer (see Figure 34). The chosen configuration is 8N1, which means each datagram includes eight data bits (LSB first), no parity bit, and one stop bit.

4.2 Software Implementation

In general the overall software implementation is entirely built with Python, which is an interpreted, dynamic programming language available for all major operating systems. A big advantage of Python is that it can be easily extended with packages allowing us to simply integrate advanced and well-tested functions into our application. One of these extensions we have used is pyserial, which is responsible to establish the UART communication on software side. Pyserial will encapsulates the access for the serial port within simple to use functions for device initialisation, data transmission and more. Unfortunately, it does not handle the synchronisation pretty well and hence require some external management. That is why we have to add short timeouts after each write instruction and also validate the received data with a checksum generated on the FPGA. If some faulty communication data is detected, the current operation will be dismissed and repeated after some sanitation steps.
» Pyserial is free available at: `https://github.com/pyserial/pyserial`.

After we have created a working communication core, we built several tools related to three consecutive phases of execution. In the following section we will explain their functionality and intention in detail with respect to their chronological usage.

4.2.1 Training

The training tool represents the first step in our overall scenario and is responsible for the database creation. Hence, it will successively iterate a given set of challenges, pass them to the FPGA and collect the returned responses. Each response will be converted into a signed integer and stored by using a python dictionary. Such a dictionary is just a list of immutable and unique keys which point to some further value or object. In our example, each challenge represents a key referring to a list of informations about it. This list contains at least some precomputed values of interest (min/max, mean, standard deviation, repetitions, etc.) and a chronological list of all collected responses. From this point of view, this dictionary almost provides anything we will need for any further application planned. For sake of simplicity, we have decided that there is no further need for a dedicated database system. The small amount of data we have to store effectively is not worth the management overhead of a database system and the dictionary already provide top-notch time complexity for data access (get and set with $\mathcal{O}(1)$). In order to exchange the dictionary data between the tools we are using a Python mechanism called Pickle. Pickle is able to store and load Python objects by serializing and de-serializing these objects, which means it covers a function to convert ("flatten") the object into a writeable byte stream and another to reverse this operation. To avoid confusions, we will continue to use the term database in the following work independent of its real implementation.

```
>> python training.py
```

4.2.2 Monitoring

After the database is created, the system can be deployed and is considered untrusted henceforth, so the monitoring phase can start. To evaluate the behaviour of the system we have created a monitoring tool. Similar to the training tool, we will iterate the challenges stored in the database. However instead of storing it, a consecutive statistical evaluation will take place this time. Based on this evaluation, the application can notify suspicious behaviour or not. One implementation we have created will evaluate the behaviour with min/max approach, which check if the response lies in between the minimum and maximum boundaries of the related database entry like introduced in Chapter 3. A fork of this tool is alternatively using the mean and the standard deviation (with 2 or 3 sigma) and equally tests the response against this range. Independent from the chosen method, a response out of this range will not simply mean the device is under attack. PUFs are noisy in general and may cause false positives which have to be smooth out for a sound evaluation. Hence, we extended our approach to do n repetitions for each challenge and evaluate its outcome based on majority voting. By default we will do 5 repetitions each, which does minimise the PUF noise sufficiently for our purposes.
Because all this statistical data, can not be interpreted by simple value observation of the user we have added a graphical feedback to the terminal output of this tool. As shown in Figure 13, each line will represent a complete iteration of the challenge set. Each majority evaluation will thereby result into a "-" for a proper set of evaluations, while "!" will indicate a suspicious response set. At the end of each line we additionally sum up each evaluations, which can hence be interpreted as an attack probability. Furthermore the monitoring run can be canceled by pressing "k", which will instruct the tool to stop after finishing the current round. Additionally we added a simple record function, which saves the collected data of the monitor application to a dedicated run file.

```
>> python monitor.py {savefile}
```

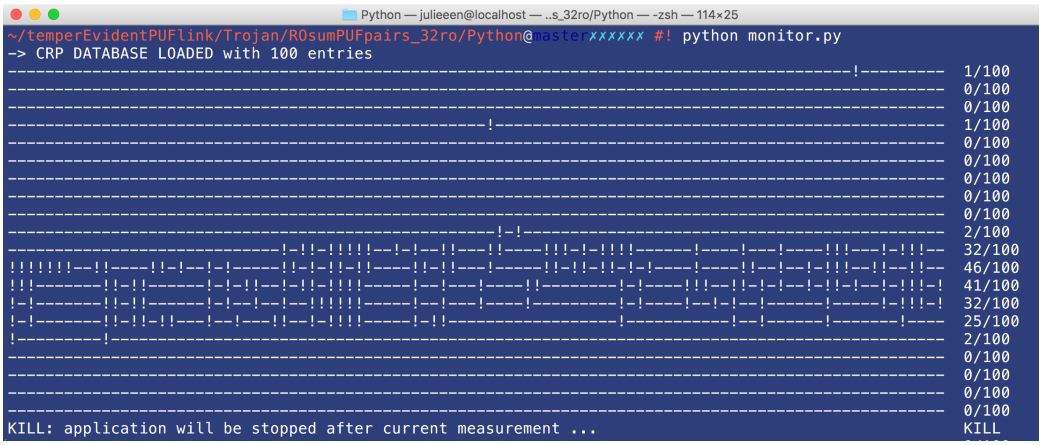

Figure 13: Monitor tool with visual feedback

4.2.3 Evaluation

An optional evaluation phase will take place after monitoring and shall allow a user to analyze interesting situations even when they lie in the past. This is especially important for us because most of the planned experiments are pretty elaborate to repeat and we couldn't predict how good the initial methods will work. Hence, we have implemented an additional replay tool, which allows to reevaluate the data recorded during a monitoring run. To start it, the user has to feed it with the CRP database and the related run file created by the monitor tool. In doing so, the basic replay tool will simply output the same results like the origin monitor, while different forks of it will allow us to review the experiment with other statistical methods.

```
>> python replay.py database.pkl savefile {replay_savefile}
```

Another used python extension, especially related to implement more advanced analysis strategies is NumPy. NumPy will offer a sophisticated N-dimensional array object, which feature many easy to use functions for mathematical operation including linear algebra, transformations or random number capabilities and is free available under `http://www.numpy.org/`.

4.3 Hardware Implementation

The following section will focus on the solved tasks during the hardware development. The implementation was done with Verilog, a hardware description language (HDL) used to model electronic systems like FPGAs.
Typically each HDL implementation consists out of different modules, which are separated by concern. These modules can be simply instantiated several times after implementation and provide this way a single point of maintenance. Such an instantiated entity can furthermore be plugged together related to the application design and will this way from an overall system hierarchy. In any hardware design the top module represents the highest entity in this hierarchy of entities and hence will provide the entry point of execution.
The top module we created contains mostly a high-level state machine linking all further implemented modules together. The execution flow of this machine can be described as follows: The initial state is responsible to reset all components and initialise the state machine itself. Afterwards, it will wait until a first UART datagram is received. Considering the transfer of a 64 bit challenge, this state will loop until all 8 bytes are read. The next state pass the received challenge to the PUF component, which autonomously handle all its assigned operations (e.g. sampling, sum calculation, etc.) on itself and finally return the PUF response. Based on that, the state machine will continue by calculating a checksum, which is added to mitigate transmission errors and data corruption. The final send loop state will now transmit 8 bytes for the response and an additional byte for the checksum to return afterwards to the initial state of the state machine.

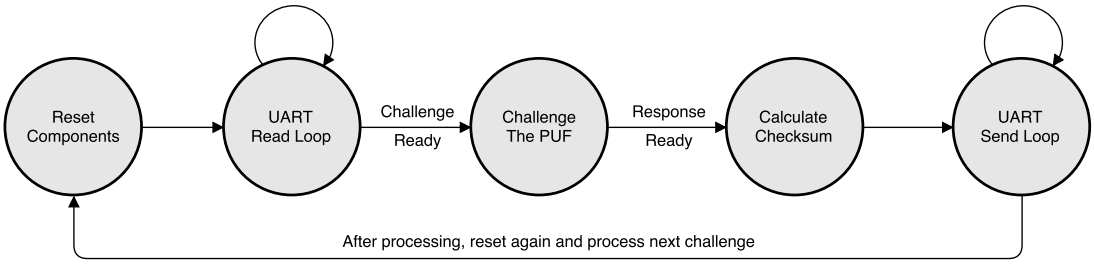

Figure 14: State machine linking and controlling all hardware components

The UART implementation we used is borrowed from the Datenkrake project, which is free available under: `https://github.com/ddk/`.

4.3.1 Ring Oscillators

One of the most difficult parts by implementing RO based PUFs is the creation of adequate ring oscillators. Fortunately, [53] and [19] does already cover much information and tips to build high quality ring oscillators on FPGA systems. With respect to this, we want to conclude some tradeoffs and dependencies we have to deal with during the fundamental RO design. A good starting point for this discussion will be the question about how many inverter stages should be

used, to constitute the ring oscillator. A large amount of stages results into lower oscillation frequency, while less of them will higher this frequency. The physical relation of this is expressed more precisely in [53]:

$$f = \frac{1}{2 \cdot n \cdot t_d} \qquad t_d = 0.52 \cdot \frac{C_L V_{dd}}{(W/L)k' V_{DSAT}(V_{dd} - V_{th} - V_{DSAT}/2)} \tag{2}$$

Frequency of a RO with n inverters having an delay of t_d each. With C_L = load capacitance, k' = transconductance, W/L = ratio between transistor width to channel length and saturation, threshold and supply voltage. [53]

To answer this question we have to consider that the minimum number of stages is limited by the operating speed of the measuring counter, which in turn is determined by the technology node [53]. If the frequency is to high, a counter will probably miss rising edges and cause systematic errors. On the other hand, we cannot excessively increase the amount of stages until we gain a proper frequency because we have to consider the area overhead. If we imagine a RO Sum PUF with a 64 bit challenge, we have to create $128 \cdot n$ stages. Furthermore, we have to be able to distribute the stages over the whole IC, which will further limit the maximum amount of stages.
Related to this parameter, we also have to consider the applied sample time. This will determine how long a counter will be incremented by an oscillator until we stop the measurement, which is typically expressed in clock cycles. This process will quantise an analog signal into a discrete one, hence we have to choose a sufficient minimum sample time to avoid undersampling and loosening resolution. For example, if we measure some ROs with a close frequency range around 50 MHz ± 5 MHz by using only five clock cycles this will cause a quantisation error of 5 MHz (50/(2·5)). This means that each RO between 45 MHz and 55 MHz will be expressed with the same counter value and hence make them indistinguishable according to [19]. To minimise this error to roughly 0,1 MHz, we have to apply 250 clock cycles like shown in the following equation stated by [19].

$$e_{measure,max} = \pm \frac{f_{clk}}{2 \cdot n_{clkcyc}} \qquad n_{clkcyc} = \frac{50 \text{ MHz}}{2 \cdot 0.1 \text{ MHz}} = 250 \text{ clkcyc} \tag{3}$$

(a) Maximum quantisation error based on system clock f_{clk} and sample time n
(b) Solved equation to find minimum clock cycles required for a 0.1 MHz error

As we can see, ROs with an approximate equal frequency require more time until their small differences will emerge and may appear equal when this time is to low. On the other hand, the measurement deviation is steadily increasing by extending the sample time [53], while long sample times require larger counters to mitigate overflows, which results into bigger hardware footprints.
Based on these considerations we started to implement our own ring oscillators on a Cyclone IV FPGA (A) using different parameters. The result of this evalu-

ation task can be seen in Figure 15 and 16. As one might expect the oscillation frequency, which is proportional to the counter value, greatly decreases by increasing the amount of inverter stages. Furthermore, slower oscillators require more time to overflow the counter and hence can be used to reduce the overall area consumption of the design. The spreading of the graph shows additionally, how oscillation differences will steadily emerge when using larger sample times (wider spread) in comparison to shorter sample times (lower spread).

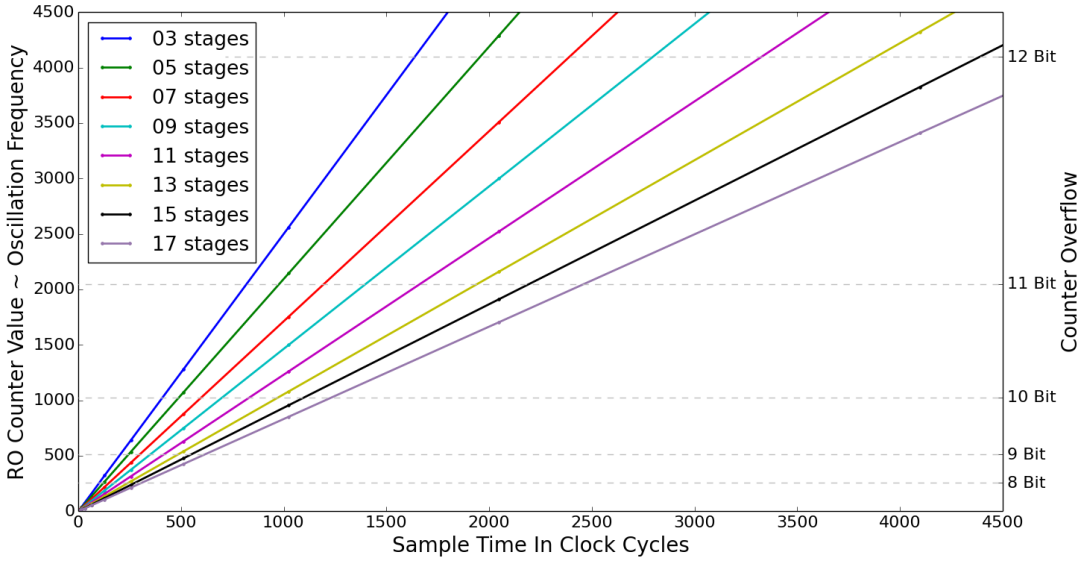

Figure 15: Relationship between RO stages and sample time quantity (Stages placed vertically in one LAB column)

Additionally, we evaluated the deviation behaviour of the created ROs for 100 repetitions in Figure 16. As we can see, a sample time lower than 200 clock cycles will be quite error free, while more than 4000 cycles cause strong variances in the measurement. Moreover, a large inverter chain keep the noise lower in general, while small chains cause significantly more noise.

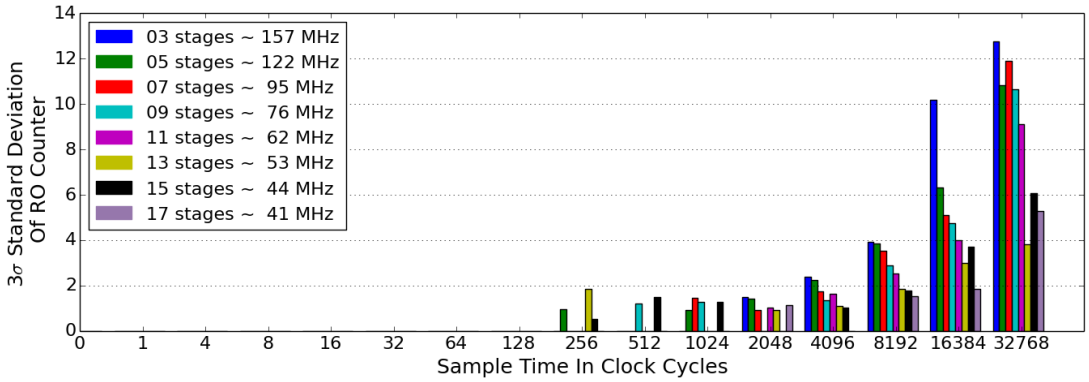

Figure 16: Standard deviation for different RO sizes and sample times

4.3.2 Placement

Another key factor for RO performance, is the placement of the inverter stages on the FPGA. Regarding our design requirements in Chapter 3, we want to distribute them as much as possible over the whole IC to establish a high sensitivity for locally confined effects. How this sensitivity is related to the distance between ROs and a source of distortion can be seen in Figure 17 elaborated by [53].

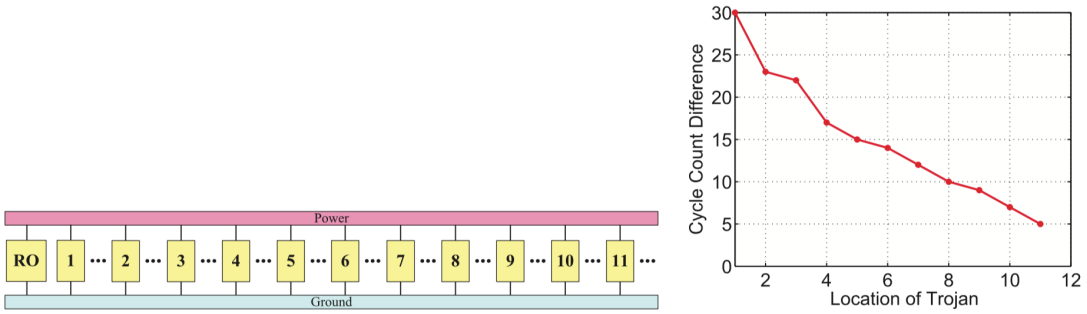

Figure 17: (a) Setup with fixed RO location and different Trojan distances. (b) Decreasing influence caused by Trojan translation. [53]

According to [53], the distribution of oscillators will further make it inherently more difficult for an adversary to remove or tamper with the components. And lastly, a well-balanced distance between the ROs is furthermore necessary to keep the influence of neighbouring ROs small. Based on these observations, we started to implemented a few ROs using different placement strategies by ourself. Hereby we not only want to distribute each of the ROs, we even want to spread the single RO inverter stages over the device. According to [49] long interconnects between these stages are good victims for crosstalk effects and interconnect variations and hence promise better environmental sensitivity.
For the placement on the Cyclone IV FPGA (A) we can allocate 1539 LABs (Logic Array Blocks) structured into 33 rows and 48 columns (excluding several LABs not accessible because they implement hard wired logic). Using the Quartus II IDE, we can either rely on the auto placement or do it manually.

- Auto-placement: Using the Quartus auto placement is the worst option to choose. It causes that the single ROs become unpredictable regarding their frequency. In greater detail, some will oscillate far beyond 200 MHz, some stay lower than 50 MHz and others will be somewhere between. Considering this behaviour as a foundation for the RO Sum PUF, single ROs will strongly dominate the response and hence infringe the requirement of equal participation stated in Chapter 3.1. In addition to that, we couldn't instruct Quartus to distribute the ROs neither the single stages on the FPGA, which is certainly another obstacle.

- Manual Placement: Using the manual placement strategy each single design element can be placed individually within the Quartus Assignment Editor.

To simplify this task we created some python scripts which generate placements according to a specific placement strategy. The best approach we could discover thereby is the line-by-line placement strategy.

Line-by-line placement can be done in a vertical or horizontal way and is simply done by placing each stage of a RO into consecutive LABs. In doing so, it seems somehow unproblematic to spread the stages by skipping intermediate LABs, as long as we do it equally for any stage. Certainly we have to avoid line overflows, which means we shouldn't place some stages of a RO at the end of a line and continue this placement in the next. That is why our placement tool is always calculating the available LABs based on tuples of the chosen RO size and place it only in lines containing sufficient space. Regarding the vertical or horizontal approach we favour the vertical way because it have the benefit that unassignable lines can be easily skipped.

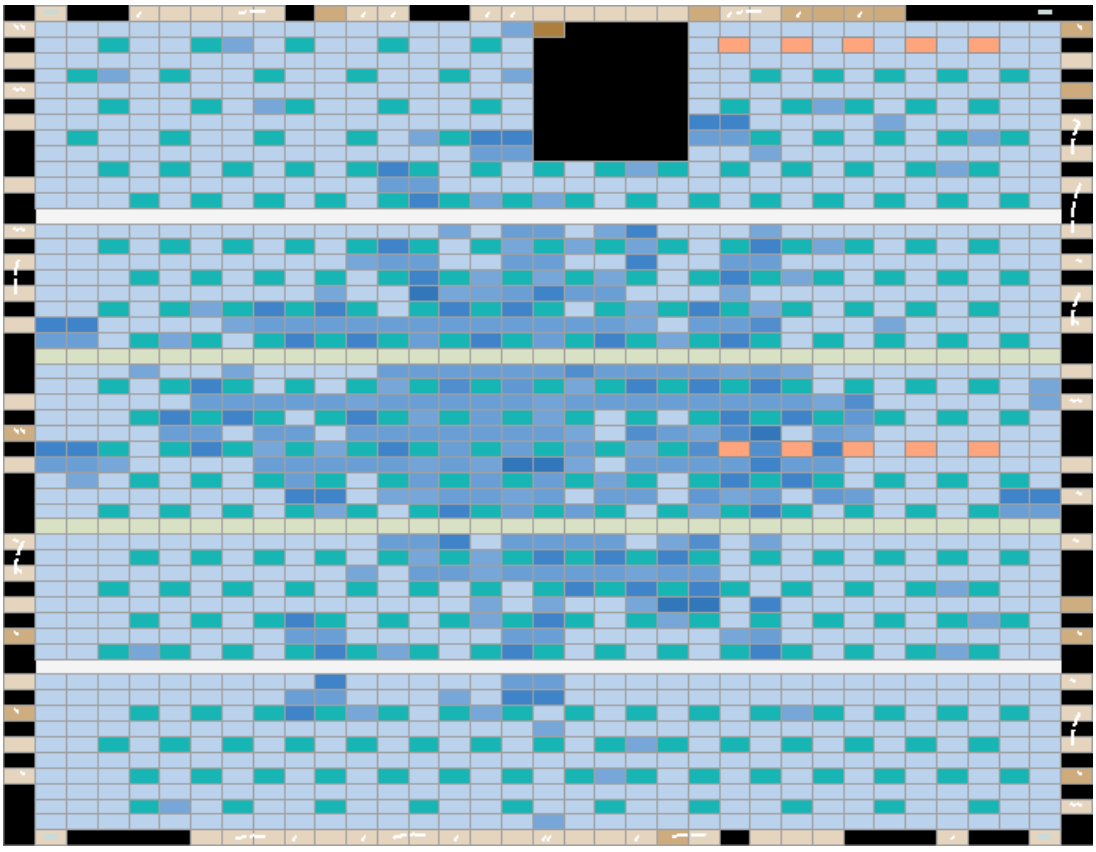

Figure 18: Screenshot of Quartus II Chip Planer Tool, showing line-by-line placement of 64 ring oscillators (green LABs), further logic for RO Sum PUF and communication (blue LABs), as well as unused LABs (light blue LABs). (Image rotated by 90 degree clockwise)

4.3.3 Optimisation

The following section will discuss some changes we have done to optimise the overall PUF performance. One advancement we could discover is especially related to the RO placement. Like already introduced, to compute the response of the PUF, we have to calculate the difference between several RO pairs (Chapter 2.6.5). This is done in our implementation by comparing the first RO with the second, the third with the fourth one and so on. If we would now naively apply the line-by-line strategy on the sorted list of ROs, we might place each related RO pair into approximately the same area of the FPGA. However, this is bad for detection sensitivity because local effects may disturb a RO pair in the same direction and hence may neutralise the effect in the difference more likely. To mitigate this and additionally mix the RO effects among each other, a far better approach will shuffle the placement or even better maximise the distance between RO pairs. To keep it simple, our placement calculation will divide the set of possible tuples in halves and chose the assignments in an alternating fashion from these sets and apply it to the sorted list of ROs. While doing so, we can ensure that RO pair are always placed apart from each other by approximately half of the maximum possible distance. For a better visualisation of this relationship, we have highlighted the position of the first and the second RO in Figure 18, which will constitute the first RO difference pair of the PUF.
Another advancement is recommended by [19], which states that advanced disabling mechanisms will improve the overall PUF quality. Indeed, this is true for simple RO PUF because neighbouring ROs will not influence each other. In case of the RO Sum PUF we could transfer this idea by calculating each RO separately, but in doing so we may miss local effects which are occurring just for a short moment in a single location. Furthermore, mixing the effects of neighbouring ROs might perhaps compensate the missing connection between ROs not directly linked in the computational model of the RO Sum PUF. Applied in a meaningful sense, each RO may this way influence the ROs next and will somehow create a network of dependencies comparable to the advanced evaluation model of [53]. From this point of view, downscaling the PUF seems to be a better solution then disabling when trying to minimise the influence of neighbouring ROs.

4.3.4 RO Sum PUF

Concluding the previous evaluations, by using a 5 stages ROs and apply the advanced line-by-line placement strategy we can produce stable oscillators within 125 $\pm$ 10 MHz and further allow a meaningful distribution of the ROs. An appropriate sample time will lie in the range between 64 to 256 clock cycles, which will yield a low deviation and an acceptable quantisation error ($0{,}4 \leq e \leq 0{,}01$). Furthermore, this sizing is comparable to the solution used by [53] for the ring oscillator networks.
Based on this elaboration we are now able to build a proper RO Sum PUF. The implementation itself consists out of 5 modules which are linked and controlled by

a state machine to initially reset all components, triggering any subprocess and finally latch the response until reset again. To build a RO Sum PUF supporting a 64 bit challenge, we have to create 128 ROs. Our implementation is using ROs with 5 stages followed by a 15 bit counter. The sample time can be configured via parameters and is set by default to 128. According to our previous observations in Chapter 4.3.1, this seems to be a good compromise in terms of stability and area consumption. Furthermore, if we estimate a RO slope like shown in Figure 15 (130 MHz RO $\approx$ 2.1), we can apply far more clock cycles than necessary ($\approx \frac{2^{15}}{2.1}$) until an overflow will occur with a 15 bit counter. After sampling, the counter values will be processed by 64 modules responsible to compute the difference, while the following stage of 64 multiply modules will alter the sign of these values according to the challenge applied. The response is computed by a large adder module, which is implemented imperatively and sum up one value per clock cycle because a naive implementation of this computation (64 additions in parallel) will require much space and caused time dependent errors. Additionally, this is a signed computation and we have to declare any input wire explicitly as signed to ensure a faultless computation.
Indeed, the chosen design and sizing must not represent the optimal solution. For instance, one might use a more intelligent adder (e.g. carry-lookahead) or other architectures, to yield faster computations and reduce area consumption. But so far, this implementation satisfy our needs for experimental purposes. Moreover, it can be can be easily scaled up/down depending on the available resources of the FPGA. To do so, one can simply remove/add a basic blocks like shown in Figure 19 to reduce/enlarge the challenge space and hence the required space.

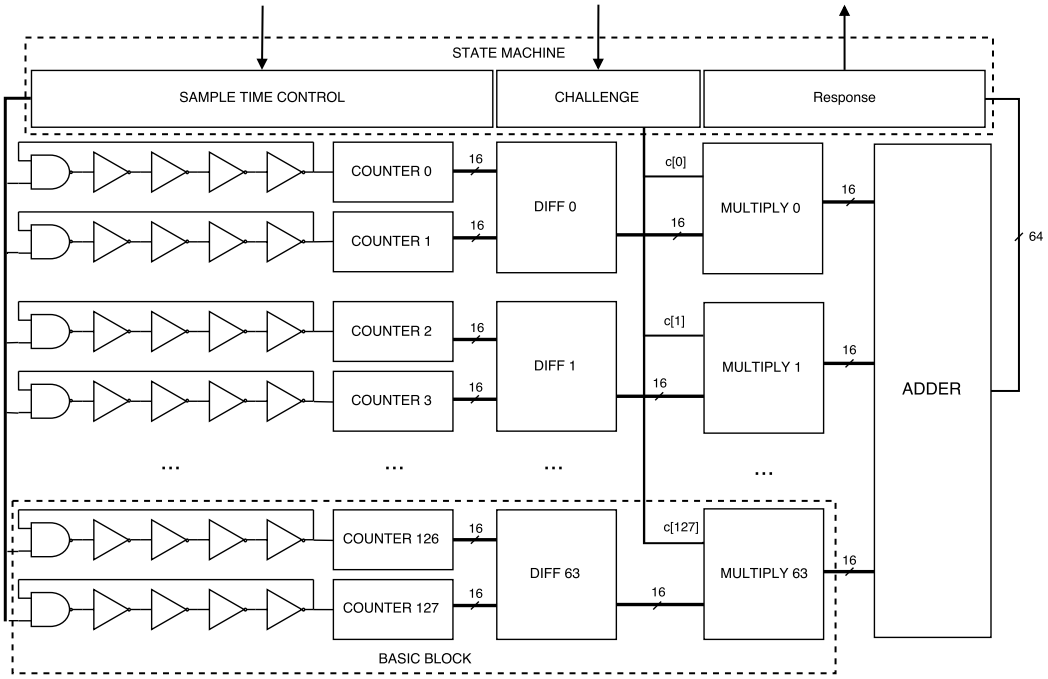

Figure 19: Overview of the RO Sum PUF implementation

4.3.5 Resource Utilization

To evaluate the resource utilisation of this implementation, we have elaborated the IDE placement report for multiple sizes of the RO Sum PUF. Obviously doubling the elementary logic will cause a quite linear grow in the utilised resources. The overall utilisation is made up by round about 13% for ROs, 35% for counters, 15% for multiplier plus differential components and finally 20% for the adder.

Metric	32 ROs	64 ROs	128 ROs
Logic Cells	2030	3962	7838
Logic Registers	604	1116	2140
LUT-Only LCs	1418	2825	5662
Register-Only LCs	21	49	92
LUT/Register LCs	591	1088	2084

Table 1: Resource utilisation of RO Sum PUF implementation

4.3.6 IDE Configuration

Related to the creation and placement of ROs, we want to additionally point out how important a proper IDE configuration will be to create a working solution. Thereby we typically configured a balanced optimisation mode, but in general we have to suppress most compiler optimisation's manually.
To do so, we have activated the register duplication and retiming prevention options (Project > Settings) in Quartus II IDE. This will prevent the compiler from creating a copy of registers to improve routability and minimise the wiring and furthermore prohibit to move or modify the combinational logic to improve certain data path delays.
Furthermore, the compiler will additionally try to optimise the design, which is helpful in some cases, but not in general. From compiler perspective, a ring oscillator is actually nothing else than a combinatorial loop without any useful functionality. Because of that, it will try to optimise the design and remove its logic which we must prevent. To do so, we can explicitly add a keyword after the output of an instantiated component. In particular, this is done differently for Xilinx and Altera IDEs, in the following way:

```
wire ro0_out /* synthesis KEEP */; // Quartus syntax
(* KEEP = "TRUE" *) wire ro0_out;  // ISE syntax
ring_oscillator ro0(
  .R_in(ro_enable),
  .R_out(ro0_out));
```

5 Evaluation

The following section will give a close look at the performance of our implementation like previously decribed. In greater detail, we will review several experiments we have created and present their outcomes and our expectations. Furthermore, we will also explain assumptions we have taken and discuss the validity of our experiments.

5.1 General Setup

In general our evaluation will cover multiple attack scenarios, which are related to threats introduced in Chapter 2.4 and use cases of actual vendor implementations presented in Chapter 2.7.1. Indeed, there are so many possible variations of single attack types that we cannot cover all, hence we will focus on very basic ones to show the overall performance and allow a prediction for more complex ones. In greater detail we will have a closer look how the PUF will react to:

- Hardware Trojan insertion
- System frequency manipulation
- Supply voltage manipulation
- Laser based scanning and fault injections
- Temperature changes

To structure these experiments we have respectively created a unique Quartus II project containing the basic hardware implementation previously described. Based on that, any required modification (inserting Trojans, down-/upscale, etc.) can easily be done in the separate project without altering the overall implementation. All relevant data created during the experiments can be found in the dedicated Git repository: `https://gitlab.sec.t-labs.tu-berlin.de/jfietkau/TamperEvidentPUF.git`

Beside that, multiple parameters will be kept constant during the evaluation and will be valid unless otherwise stated: For sake of simplicity and comparability, all experiment will use the same set of 100 challenges, which are randomly generated. The training phase is normally done for 50 rounds and follows the procedure described in Chapter 4.2.1. Furthermore, the experiments will be evaluated with the methodology presented in Chapter 3.2, using a 20% detection threshold to make differences more clear and to establish a common reference value. Moreover, all non-temperature related experiments will be done under room temperature condition. Related to our implementation, the placement will be done with the line-by-line strategy introduced in Chapter 4.3.2. We will configure the sample clock with 64 clock cycles because it has yield the best results in previous evaluations. The system is normally driven by a internal 50 MHz oscillator and powered over USB.

5.2 Trojan Insertion

One of our first observation related to all experiments was, that each new synthesis of the implementation will yield a modified bitstream. This is certainly caused by optimisation and placement varieties managed by the proprietary compiler. Indeed, the PUF is very sensitive to such kinds of modification, which implies that any data recorded during an experiment is always tied to its related bitstream file. Considering that, even if the same code is synthesised again, the resulting bitstream will alter the intrinsically linked response with such an magnitude that previously recorded databases are meaningless afterwards. Hence, altering the source code is not an option for an attacker because of the resulting bitstream modification. Indeed, he might try to reverse and manipulate an extracted or stolen bitstream file for Trojan injection. Another attack might misuse the partial reconfiguration possibilities recently introduced by FPGA vendors, allowing to partially alter the device configuration even while operating. This way someone could try to change a small portion of the device and keep the necessary PUF elements intact. Without concerning the high effort of such an attack, an adversary might find a way to inject a Trojan even if he do it already during fabrication. Hence, in the following we will assume that an attacker already have placed a Trojan, which can be activated by some external or internal signals or conditions.
For the Trojan detection evaluation we created several additional setups for the Cyclone IV FPGA (A) to figure out which PUF density is required to reveal Trojans randomly placed on the FPGA. Hence we configured multiple sizes of the RO Sum PUF covering 32, 64, 128 ROs and validated their detection performance. In fact, less ROs will yield lower placement density and less intra RO distortion, but indeed also decrease the monitored area, while more ROs will increase distortion and coverage. For an initial evaluation we added some other oscillators to the configurations, which can be activated externally and represent a first, simple Trojan model. ROs are pretty small, can be assigned to a single LAB and yield a high impact when activated, which may influence the PUF strong enough to cause a detectable change in the response. For each setup, we repeatedly create a database of the unbiased device with round abound 40 iterations and started the monitoring tool afterwards. Like expected, the active Trojans will force the responses to leave their normal operational range and indicate a change of the physical integrity. On closer observation, the smallest PUF version (32 ROs) seems to yield the most stable behaviour, while larger PUFs result typically into higher response deviation possibly caused by the intra RO distortion. Moreover, the smallest version does already show a sufficient Trojan detection capability, which will be further evaluated in the following.
Based on this observation we created a first setup, trying to examine the relationship between Trojan distance and detection ratio. The farthermost point where we can place a Trojan can be found at the margin of the FPGA, like presented in Figure 20. Hence, we assigned a Trojan to each of the five positions shown, with decreasing distance to one of the PUF ROs.

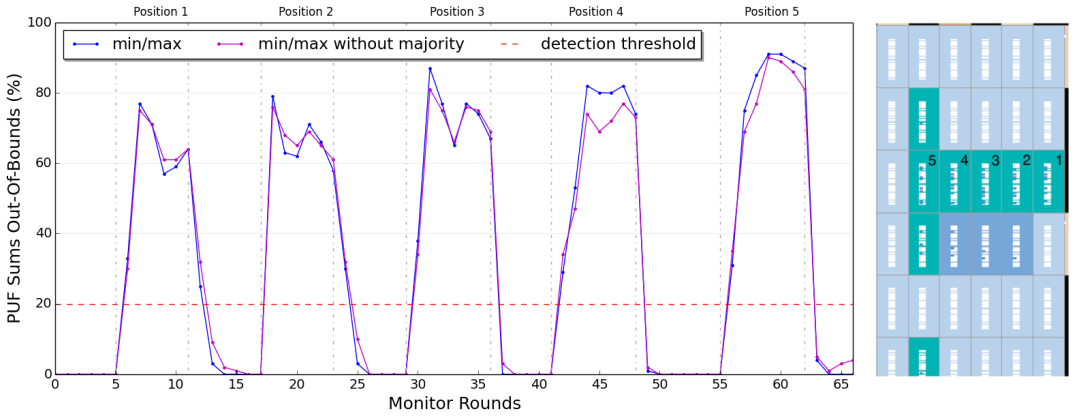

Figure 20: Monitor record for different Trojan distances (RO Trojan)

After configuration, we created the unbiased database and started the monitor application as usual. Next to some initial rounds, we activated the first Trojan at the farthermost point (Position 1, +4 LABs) and disabled it some seconds later. Like that, we did it for each of the single Trojans with decreasing distance to the PUF until we have reached the last one, which is finally a direct neighbour of the PUF RO (Position 5, +0 LABs). As we can see in Figure 20, depending on the distance of the Trojan the out of bounds ratio will increase, while the overall influence of RO based Trojans is strong in general. Furthermore, placing a Trojan within the area covered by a PUF will cause quite different changes not only correlated to the distance. ROs are slightly unequal by nature and hence a Trojan close to a dominating RO can change the response much stronger as a similar Trojan next to a non-dominating one, like we can also see in the computational model of the RO Sum PUF:

$$S = \sum_{i=0}^{n-1} (c_i * 2 - 1) \cdot (f_{2i} - f_{2i+1}); c_i \in \{0, 1\}$$

Assuming a trivial PUF like n = 1 with c = 1_2:

$$S = f_0 - f_1 + f_2 - f_3$$

and adding a Trojan factor t influencing one RO or another:

$$\begin{aligned} S_1(t) &= f_0 \cdot t - f_1 + f_2 - f_3 \\ S_2(t) &= f_0 - f_1 + f_2 - f_3 \cdot t \end{aligned} \tag{4}$$

now assume unequal ROs like: $(f_0, f_1, f_2, f_3) = (10, 11, 12, 13)$

$$S_1(t) = 10 \cdot t - 11 + 12 - 13 = 10 \cdot t + 36$$

$$S_2(t) = 10 - 11 + 12 - 13 \cdot t = 13 \cdot t + 33$$

And we can see, that trojan influence depends also on how strong the affected RO is participating.

Furthermore a Trojan might affect multiple ROs at the same time and hence its influence will also be mingled by their individual influence.

To estimate a more realistic scenario we created a new Trojan model, containing an LFSR functionality. This will cover a shift register with some additional feedback taps that is clocked by the system frequency. Like the RO Trojan we can equally activate it externally to introduce switching activity temporary and observe the PUF behaviour. The size of this Trojan was furthermore adjusted to fit completely within one LAB, so we can easily assign it on the device. According to the Quartus II fitting report one of these Trojans will use 15 logic cells and 14 logic registers at all. To measure the impact of this Trojan type, we placed five of them at the right margin of the FPGA. Apart from the new Trojan model the configuration is retained like before. During monitoring we successively activated only one, two, three, four and finally five Trojans simultaneously and monitored their influence on the PUF. As we can see in Figure 21, one or even two active LFSR Trojans will not cause a change strong enough to reach the detection threshold. However, when enabling three or more, we can see a significant change

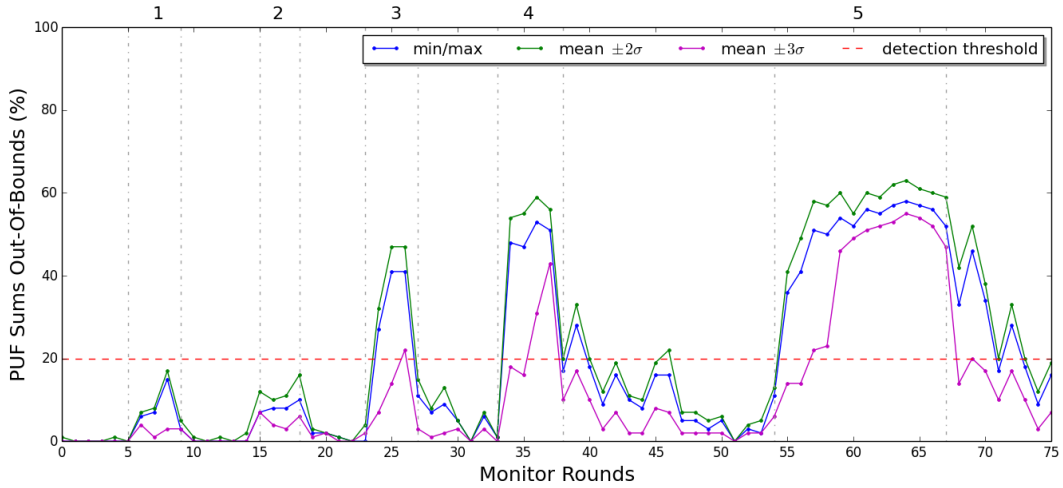

Figure 21: Monitor record for different Trojan sizes (LFSR)

in the out of bounds ratio of approximately 50% and beyond, depending on the evaluation method used. The new Trojan model yields thereby a more unequal amplitude, which quivers a lot and looks different each time. This might be caused by the varying switching activity of the LFSR. Furthermore, we can see that the slope of an LFSR Trojan is quite lower that the slope introduced by the formerly used RO based Trojan. More generally spoken, a realistic Trojan have to operate quite some time until the threshold is reached and need a decent size to be detectable.

For a last evaluation we will take into account how [53] evaluated the purposed architecture of ring oscillator networks for Trojan detection. They implemented Trojans with different sizes (2-20 Gates, 2-16 FlipFlops) covering comparator and shift register functionalities. The Trojans were placed at different points of the device, which furthermore contains a basic AES implementation. This AES module will simply represent an asset which requires protection and will

introduce some basic distortion produced by normal device operations. This configuration is finally evaluated within simulation and on a real FPGA under room temperature conditions by [53].
Related to that, we created a similar experiment: We configured the Cyclone IV FPGA (A) with a PUF covering 32 ROs, like done in previous evaluations. Additionally we added a basic AES-128 implementations, which simply execute a dummy encryption process in an infinite loop. Furthermore we slightly dis-

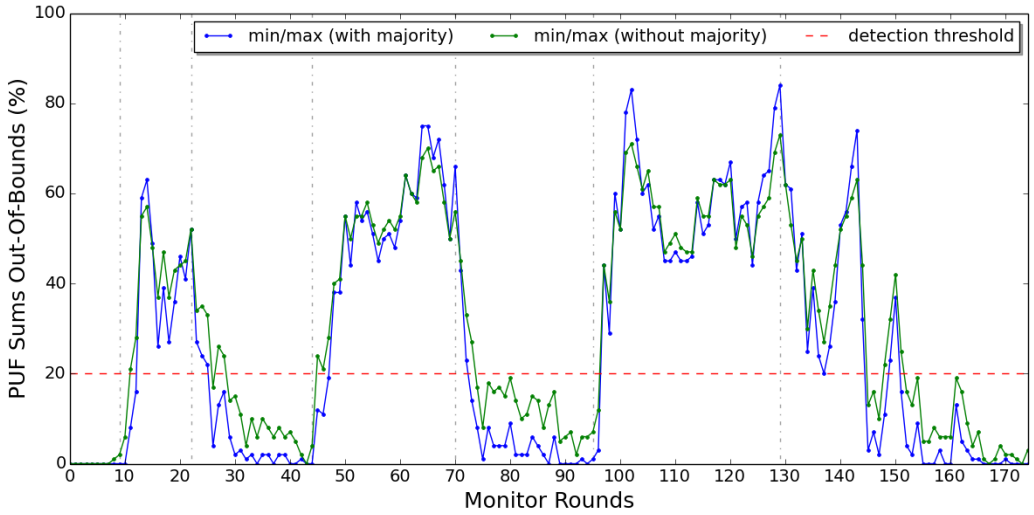

Figure 22: Monitoring an AES-128 implementation while 3 times activating 5 LFSR Trojans. A single round have an approx. duration of 8 seconds.

tributed 5 malicious LFSR Trojans in the upper left quarter of the FPGA, using 5 LABs in total (Detailed placement shown in Appendix). As usual we have created a database for the configuration and started a monitoring process afterwards. As we can see, the Trojans can be clearly detected and showing their characteristic behaviour. Like before we have observed that the Trojans will sometimes cause a much longer impact on the PUF as expected. Even after disabling the malicious activity it can be detected, sometimes up to 10-15 rounds after, which seems to increase depending on the duration of Trojan endurance. Possibly, this might be caused by the intra RO distortion, which could mean that Trojan related changes will propagate even on ROs not directly affected and hence requires much time to react and abate.

Conclusion

Concluding our observations, we can argue that a significant changes can be observed in the RO Sum PUF response when activating malicious logic elements which are not part of the normal switching behaviour. Depending on size and placement, this impact will differ a lot. So far, our experiments do not cover many types and sizes of Trojans, but a more granular evaluations of these parameters can be found in [53]. Furthermore, the approach still works when some basic

operational switching activity is introduced because the PUF will incorporate this normal activity within its intrinsics. Related to that, we have to ensure that a majority of the normal operation logic is active once during the creation of the database to prevent false positives. The easiest way to activate large parts of the overall design, might be to execute some dynamic test cases which are normally created during application development. To prevent detection an attacker might try to distribute the Trojan logic in a quite clever way to circumvent a strong influence by finding uncovered spots on the chip or decelerate the Trojan logic, but this might require reversing of the design which will at least increase effort and costs. Furthermore, the solution is limited to active Trojans, because a non-activated Trojan can not cause any change and hence cannot be revealed until it gets active.

5.3 Fault And Glitching Attacks

5.3.1 Active System Clock Manipulation

In glitching attack scenarios the adversary will try to take over control of the system clock or supply voltage. Like already discussed in Chapter 2.4, a typical attack only occurs for some clock cycles to introduce a data misread or an instruction miss. The RO Sum PUF will at least require 64 clock cycles to measure a single response, hence such a short attack will not be noticeable easily. Another related scenario is differential fault analysis, presented in Chapter 2.4. As shown in [22], such an attack can be used to extract key material even from unknown cryptosystems. Indeed in booth scenarios the attacker have to repeat the attack very often until he found a point of interest during execution or collect enough data to analyse. Especially if he is unaware of the particular design, such an attack is normally based on a trail and error principle. The detection of such an attack is rather trivial with our solution, as long as the clock is directly attached to the PUF. Because the PUF will use the clock frequency to measure the sample time, even very small changes will stretch or contract this and hence strongly change the final response value.
To imitate a clock attack scenario we have created a new setup containing a RO Sum PUF with 64 ROs. Furthermore we have assigned the clock driving the PUF sampler (usually connected to the internal 50 MHz oscillator) to a I/O pin of the device. This pin can hence be driven by an external frequency generator, which is configured to produce a square-wave clock signal with 3.3 V amplitude, 1.65 V offset and a frequency within 0 - 80 MHz. As we can see in the monitoring record in Figure 23, by altering the system frequency by ± 10 MHz already ≈ 90% of the responses will leave their origin boundaries. The more we move away from the origin clock frequency of 50 MHz, the stronger will be the change. Hence, overclocking by multiples of the origin clock or interrupting the clock will yield strong and instant changes in the PUF response.

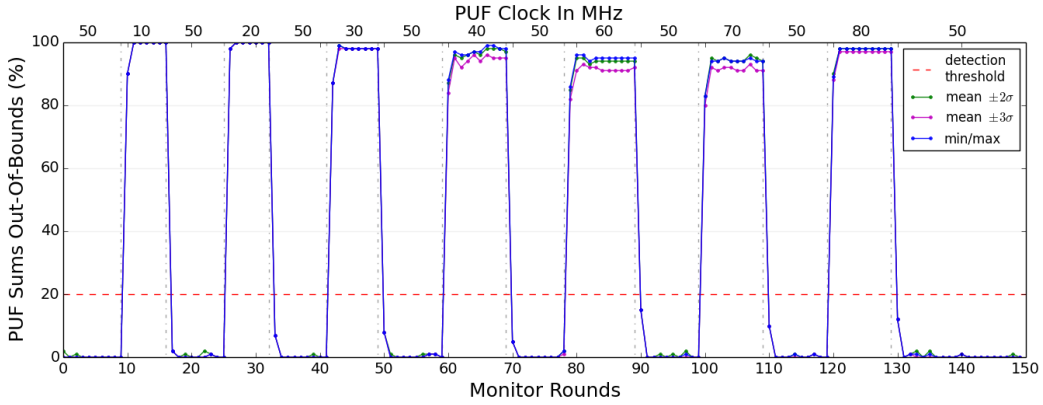

Figure 23: Monitor record for direct clock changes.

To further evaluate this behaviour, we have measured some additional response data while applying selected frequencies. As we can see in Figure 24, the single

response values evolve inversely proportional to the frequency applied, which can be reasoned by imaging the frequency influence on the sample counter. A high frequency increment the sample counter faster and the measurement will be stopped earlier. Thereby a response value is decreased by half of its size, when doubling the frequency and vice versa.

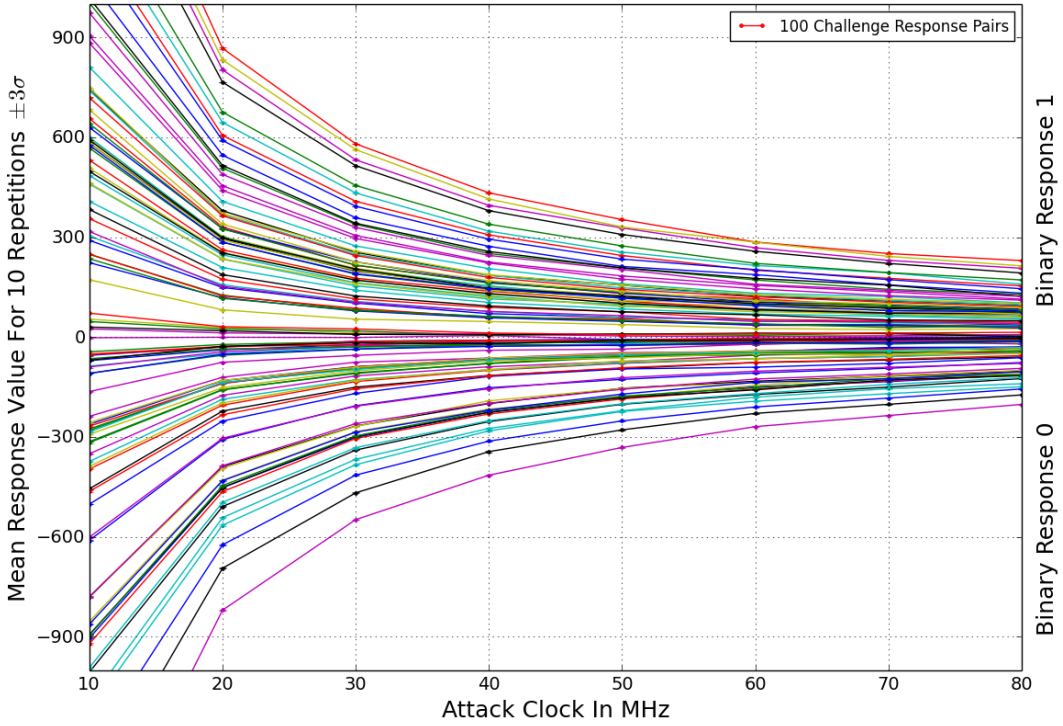

Figure 24: Relationship between system clock and PUF response

Small valued responses are thereby not simply caused by challenges with an even distribution of ones and zeros, as we assumed initially. A close look at the data will show that equal distributed challenges having likewise a large response value as unequal ones having small values. A much better explanation might be that relatively small responses are related to a challenge which balance the unequal ROs much better than others.

$$response = \frac{k_c}{frequency}$$
$$\text{E.g. for challenge c = 9D0A724A with data: (10, 1740), (20, 868), (30, 581), (40, 433), (50, 353), (60, 286) ,(70, 251), (80, 230)} \quad (5)$$
$$k_{9D0A724A} \approx 17536.25$$

5.3.2 Passive System Clock Manipulation

Another scenario we have evaluated are passive clock attacks, which are necessary to consider because today's ICs will contain multiple clock networks to drive their

logic. Furthermore, it might be imaginable that an adversary will disconnect the target component from the origin clock and attack it isolated from the rest of the system. Hence, we have to assume that some components are clocked by another oscillator as the PUF.
To evaluate such a situation we have created another setup, containing a basic AES-128 module which might be such an isolated target. Hence, this module will be the only component that is driven be the external frequency generator, while the rest of the system is clocked by the internal 50 MHz oscillator. Unfortunately, our frequency generator does only support a maximum of 80 MHz, hence we will simply drive the AES with 10 MHz for a normal operation condition and apply multiples of this for an attack. As we can see in Figure 25, an indirect clock attack does not influence the PUF strong enough and generally yield an average amplitude below the detection threshold. Not until 70 MHz, meaning a seven-fold of the origin clock signal, the detection threshold is reached by the min/max method. Hence, such an scenario seems more unrealistic to detect.

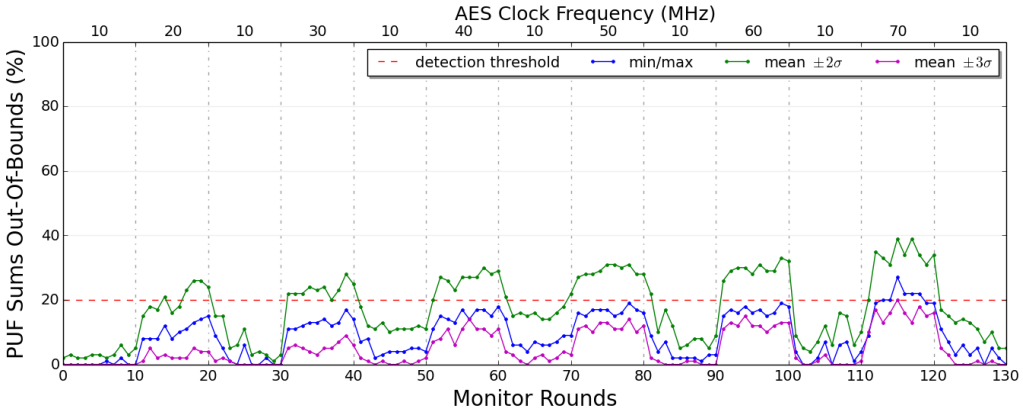

Figure 25: Monitor record for indirect clock attacks.

5.3.3 Supply Voltage Manipulation

Equally to the previous attacks, faults can be introduced by altering the supply voltage of the device. Similar to the previous discussion, a very short attack duration will remain undetected because of the PUF sample time. However, an adversary can try to operate the device out of the specified range repeatedly or permanently to create faulty computations or search initially for attackable points in execution. To model such an attack we have to modify the device because the previously used Nano DE0 board will not allow us to take over control of the supply voltage easily. First, we need to physically access the device and take over control of the power supply line. Furthermore, the device contains a capacity trying to compensate occurring differences of the power source and will bias our measurements. For sake of simplicity, we will switch to an already prepared custom board embedding a smaller version of the Cyclone IV FPGA (B). This board is generally operated by an external frequency generator and

power supply, which directly allow us to set the voltage level to arbitrary values without any modification.
The FPGA configuration we have chosen, contains the smallest PUF version using 32 ROs in total, which has shown good results before and can be easily placed on the much smaller device. Using that, we have measured the response behaviour, between 1.0 V and 1.35 V with intermediate steps of 0.05 V. Under normal conditions the device will be operated with 1.2 V.
While running the monitoring application, like shown in Figure 26, we can clearly see the strong impact of voltage changes on the response values. On smaller changes, like ± 0.05 V more than 60% of the responses will be immediately out of bounds, independent of the applied evaluation method. The stronger the modulation will be, the stronger and more reliable will be the detectable change. Some observable uncertainties might be introduced by activation/deactivation of the selected condition because we cannot instantly create the voltage level of choice on the power supply.

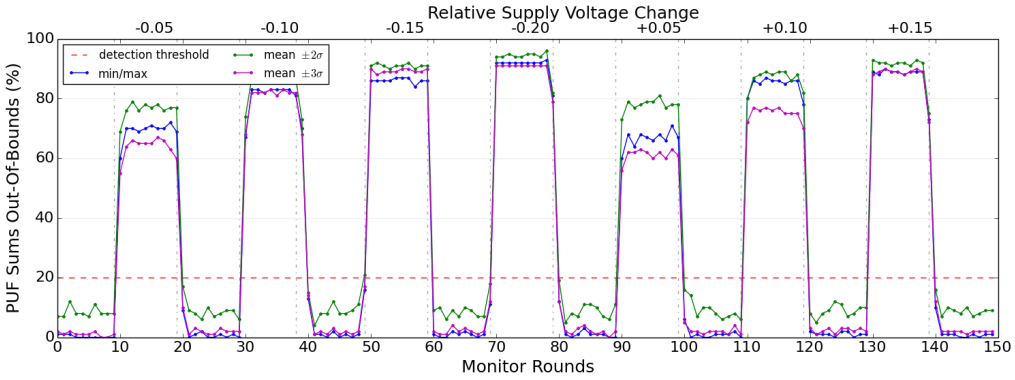

Figure 26: Monitor record with different voltage changes 1.2 ± 0.15 V.

Similar to previous evaluations, we have individually recorded the PUF behaviour under different conditions multiple times. As we can see in Figure 27, the response value grows directly proportional to the applied voltage level. The foundation for this relationship can be found in [49]. When increasing the voltage driving a ring oscillator we will accelerate the RO, because the single inverter delay t_d will be decreased and hence the overall frequency increase, like shown in Equation 2 and Figure 35. A faster RO results into higher counter values, which finally influence the response of the PUF.

Conclusion

To conclude our observations, the detectability depends highly on the attack type. In general, a very short glitch will not be seen by observing the PUF response because the timing resolution is to low. A meaningful change for onboard monitoring will require $\frac{n}{2} \cdot t_{sampletime}$ clock cycles to yield approximately 50% suspicious responses by using n challenges and assuming a permanent and ideal

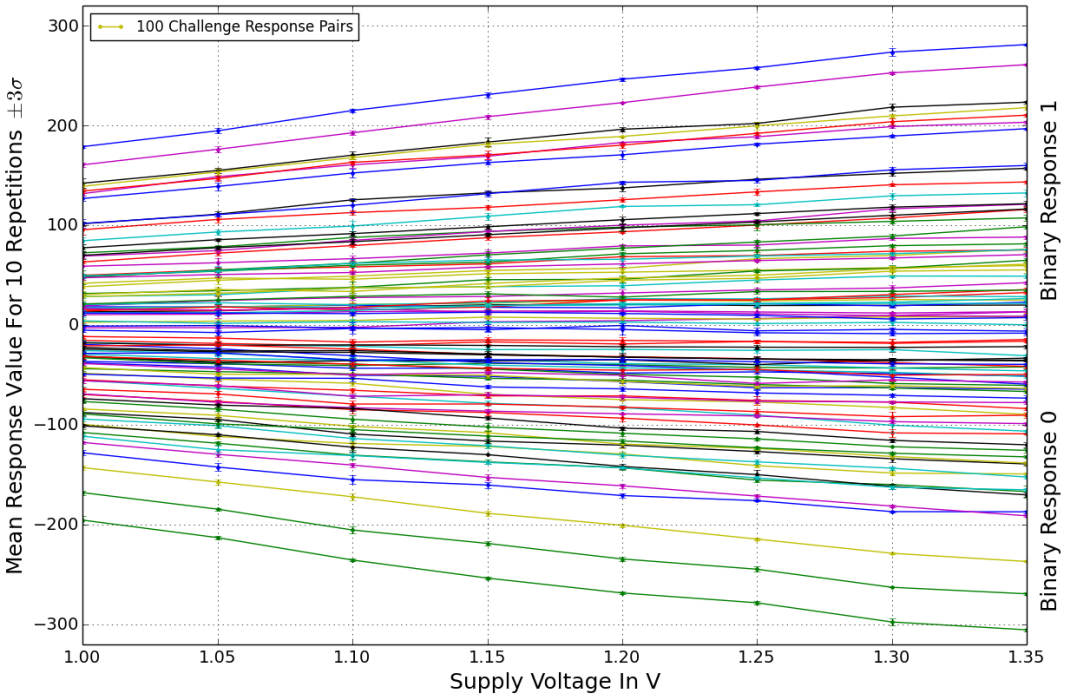

Figure 27: Relationship between PUF response and supply voltage level.

measurement. On the other hand, longer, repeated or even permanent modification will introduce significant response changes for clock as well as voltage manipulations. In booth scenarios, the impact on the PUF response is related to the challenge applied. Responses far away from zero, does typically show stronger changes under biased conditions as responses close to zero. Hence, for a monitoring scenario like proposed in this work, we might archive a higher sensitivity to these effects when favouring challenges with responses far away from zero. Similar to this, a crafted set of challenges might also improve the PUF stability for traditional applications, because responses far from zero require strong environmental changes to alter their binary representation than responses close to zero. Furthermore the detection might be mitigated by detaching a target component or when the system is using multiple voltage and clock networks by design. Hence, to increase the attack effort, we should ensure that PUFs and valuable components are driven by the same clock and power source, which can be enforced by design rules. What we did not cover in our evaluation so far, is a combination of voltage and frequency attacks. Related to our findings, an attacker might try to decrease the frequency to increase the PUF responses, while simultaneously decrease the voltage to balance the suspicious change in total.

5.4 Laser Scanning Attack

Another scenario we want to cover in our evaluation are laser based attacks. Lasers are typically used for two types of attacks: fault injection and device imaging. A fault injection attack will try to provoke an error at a particularly element of the design by shooting a short, high energy pulse at the target location. Such an attack may often introduce permanent changes, for instance by modifying a single look up table (LUT) element of an FPGA to change its functionality or flip single bits in memory (Chapter 2.4). To find the location of such an element and more generally to reverse engineer the overall architecture of the device, an attacker can use a laser scanning microscope to create high resolution images of the whole device. This is possible, because bulk silicon is transparent to special kinds of infrared light. Hence, by using this technique, attackers can see several parts of the metal layers without removing the IC package. In general, these scanning processes will take several minutes and require that the laser will iterate the area multiple times.
To evaluate such kinds of attacks, we have configured the Cyclone IV FPGA (B) with our RO Sum PUFs containing 32 ROs. The device itself is already prepared for laser experiments and will be operated by an external frequency generator and power supply. For laser stimulation we have used a PHEMOS-1000 Emission microscope, allowing us to modulate lasers of wavelength 1.3 μm and 1.1 μm on adjustable power levels. Such a power level is declared on a relative scale from 0% up to 100% and linked to the individual laser configuration. In general, we have set up the microscope to scan a large area in the center of the FPGA covering approximately a sixth of the overall area.

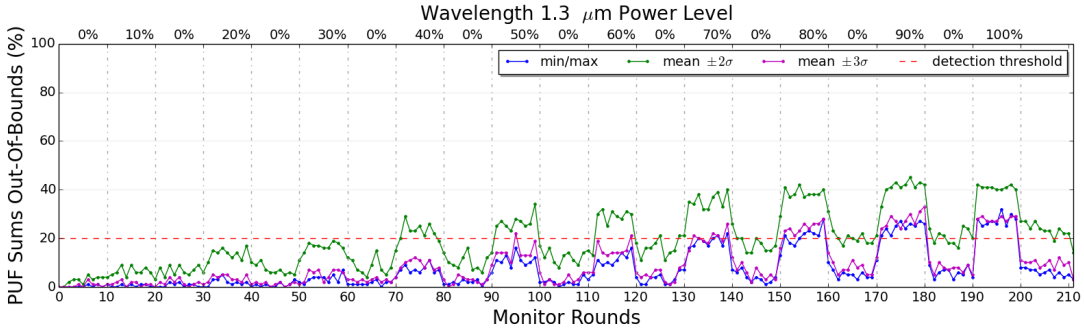

Figure 28: Monitor record of 1.3 μm laser experiment.

Before starting the experiment, we have created the database for the unbiased device covering 40 iterations as usual. During the monitoring run, we have alternately activated and deactivated the laser on different power levels. For the 1.3 μm measurement, we have used the complete power scale from 0 to 100% with intermediate steps of 10%. As we can see in Figure 28, the outline evaluation will reach the critical threshold firstly by applying a power level of 70% and beyond. For lower power levels the amplitude is relatively low compared to previous evaluations. Unfortunately, even with low power levels an adversary can already create images of the device architecture with an adequate quality.

Hence, we can detect laser scanning attempts in the given scenario, but an aware adversary can simply circumvent the detection by using only low power levels.
For the second experiment we will use the laser with 1.1 μm wavelength. From previous work we know that a power level configuration beyond 50% might inject faults into the device. Because we cannot estimate the effect of such an power level in our implementation, we will not exceed these level to prevent a possible corruption of our measurement data. Besides that, this experiment will use the same configuration, database and environmental conditions like used for the 1.3 μm laser and voltage experiments.

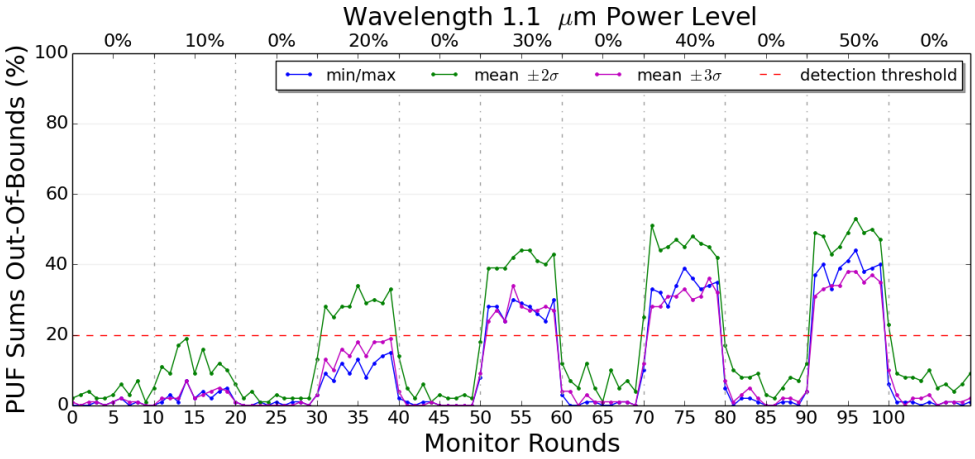

Figure 29: Monitor record of 1.1 μm laser experiment.

As we can see in Figure 29, already a low power level will cause interesting uncertainties in the PUF response, which are detectable with a proper evaluation method. At least by applying a 30% power level and more, we can clearly see a significant and increasing influence of the laser stimulation.
The reason for the occurring changes relying basically on the same principle like that for voltage transitions. In general a laser, interpreted as a photon stream will transport additional energy into the device components and hence accelerate a ring oscillator hit with these photons. The amount of energy will thereby depend on the chosen wavelength and number of photons like generally expressed in the photon energy equation:

$$E = \frac{n \cdot h \cdot c}{\lambda} \tag{6}$$

Number of photons n, Planck constant h, speed of light c and wavelength λ (assuming vacuum)

Generally speaking, with increasing power level we will higher the amount of photons and hence increase the transmitted energy, like we can clearly see in our monitoring record. Similar to this, we can observe that a higher frequency (shorter wavelength, e.g. 1.1 μm) will increase the energy transmitted while lower frequencies (longer wavelength, e.g. 1.3 μm) will decrease it, which is a reasonable

explanation for the occurring differences we can see between Figure 28 and 29. Finally, the laser might also cause thermal effects, but under consideration of the sharp slops created by activation and deactivation of the laser, this influence might be negligible.

Conclusion

To conclude our observation, we can argue that special kinds of lasers can be detected with RO Sum PUFs and might allow to mitigate attacks. This does especially apply for unaware attacker, using bad laser configurations with much power or short wavelengths. Certainly, this detection can be circumvented by lowering the power level and use longer wavelengths like 1.3 μm. Furthermore, the coarse time resolution of the PUF might allow an attacker to simply slow down the scanning process iterations and hence lowing the laser impact to mitigate the detection.

5.5 Temperatur Manipulation

The final experiment will evaluate the temperature behaviour of the PUF and try to address the threat of temperature attacks. While doing so an adversary might try to heat up or cool down the surrounding temperature of the device to operate it in areas which are typically out of the specified operational conditions. According to [4], commercial FPGAs, like used by us, can be operated securely within 0°C up to 85°C. More expensive version, targeting industrial application will typically have a larger operational range covering -40°C up to 100°C. Heating up the device beyond this range might allow an adversary to induce faults in the operation, like shown in [26]. Instead, by freezing a device one might be able to read volatile memory data caused by data remanence effects [7]. Considering such possibilities, a secure device have to monitor the surrounding temperature and react to malicious attempts.
Because the temperature behaviour for the RO Sum PUF is not originally given by [32], we will initially determine the overall response behaviour for different temperatures. Thereby we have configured the RO Sum PUF with 64 ROs on the Cyclone IV FPGA (A). To take control of the environmental temperature surrounding the device we are using a laboratory oven allowing us to create constant temperatures within and beyond the device specifications. In our experiment we have recorded the PUF response for temperatures between -20°C and 60°C with a sample resolution of 5°C. The temperature deviation lie within ± 1 °C and was monitored by a measurement unit, which is part of the overall control unit of the laboratory oven. For each sample point we have collected the response of 100 random challenges with 10 repetitions each. Based on that data we have calculated the average PUF response and its associated standard deviation (3σ).

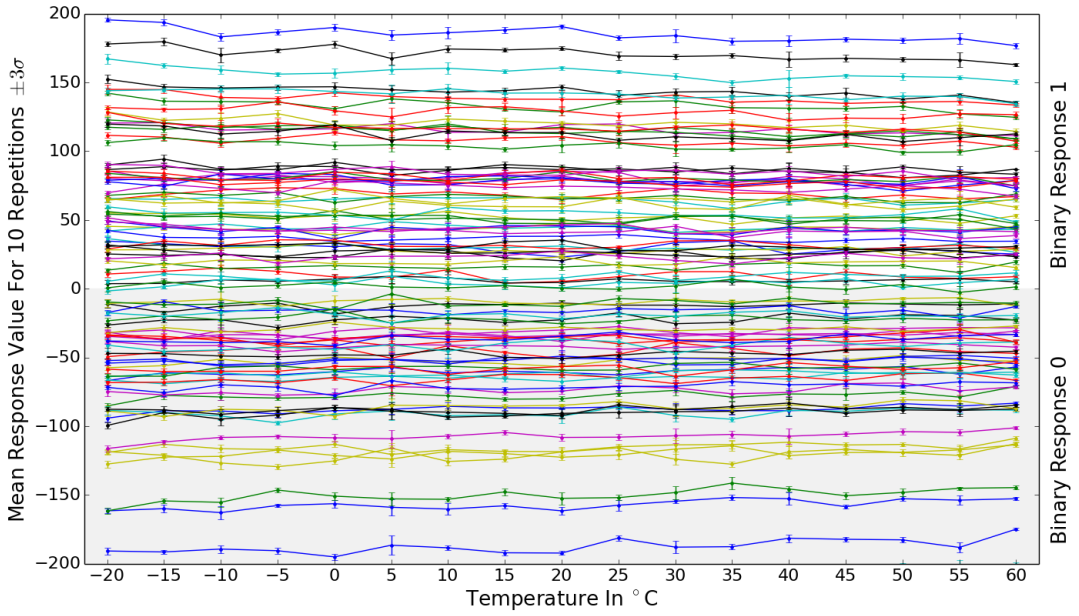

Figure 30: RO Sum PUF temperature characterisation

When observing the result, visualised in Figure 30, we can clearly see that the PUF responses does not have a direct relation to the environmental temperature. Considering a binary output, the RO Sum PUF will provide a very good temperature characteristic, which is really important for PUF applications in general. Like already discussed in Chapter 2.6.5, this behaviour is caused by the differential component which compensates static changes of individual ROs pretty well.
However, our application design is relying on the real value output of the PUF, which indeed shows slight changes and deviations. On a closer observation like done in Figure 31, we can see that the temperature transition change the response in a non-linear, random way. Indeed, these changes can be strong enough to force a response out of the origin monitoring boundaries created within another temperature condition. Certainly, such a temperature transition is not considered in our solution so far, so even a natural environment temperature change might yield some suspicious responses. To model this behaviour adequately, we can simply extend the deviation boundaries of our database collection by some temperature factor (e.g. mean $\pm$ $3\sigma \cdot t$). Unfortunately a larger deviation range might cause problems, considering the detection capabilities in previous evaluations. Especially the Trojan and laser sensitivity will rely only small changes in the response behaviour and does not yield such a strong impact like introduced by voltage and direct clock changes.

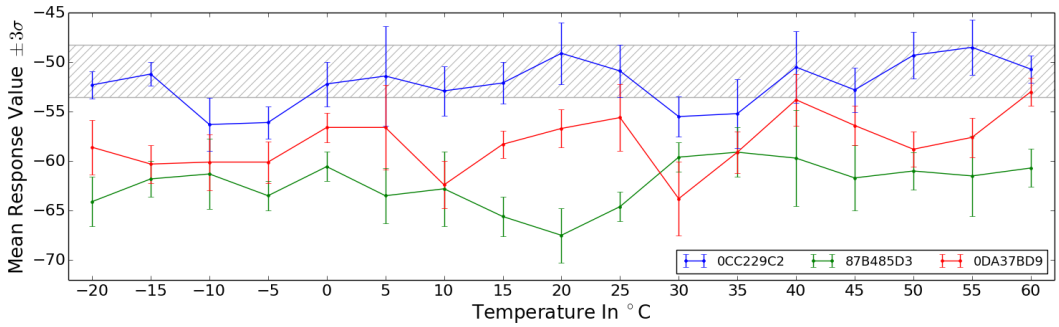

Figure 31: Close observation of 3 example challenges, and an example range which could represent a valid condition range when training is done at 25°C.

Another solution we can consider are temperature compensated ring oscillators like presented in [52]. They might show a much more stable behaviour regarding the temperature, but cannot be done on FPGAs and require full-custom ASIC design. An even better idea will be the characterisation of the PUF over a large range of normal operational temperatures. Indeed, this will cause more development overhead, but will also introduce new traits. As previously stated, the responses will change in some unpredictable and ambiguous way, but indeed it seems to be unique for each temperature. Hence, a single response will not allow us to deduce a statement about the surrounding temperature. Instead, we will use multiple CRPs and try to correlate their unique behaviour with the well defined temperature characterisation we have done before. To do so, we have

created some additional measurements with a loosely defined temperature conditions of roughly -5°C, 10°C, 25°C and 35°C. For the correlation we have taken each measurement, whose temperature will be determined and simply compare it with the temperature database samples. For the comparison we are using the mean squared error (MSE), which will estimate the average of the squares of the errors between two response sets for one temperature T_1 and another T_2. A small MSE will mean a good conformity to the comparison data set, while a large MSE will signal strong differences in between. Hence, after computing the MSE for each of the 17 possibilities, we can search for the smallest one and deduce the best estimation of the temperature surrounding the device.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\text{Response}_{n,T_1} - \text{Response}_{n,T_2}) \tag{7}$$

To make the visualisation more intuitive, we have inverted the MSE yielding large values for small differences and vice versa, along with a normalisation of these values between [0, 1] by using feature scaling. As we can see, the sample temperatures will greatly correlate with the related temperature measurement done previously. Generally speaking, any of our sample measurements can be uniquely mapped to the related PUF behaviour at a decent temperature point.

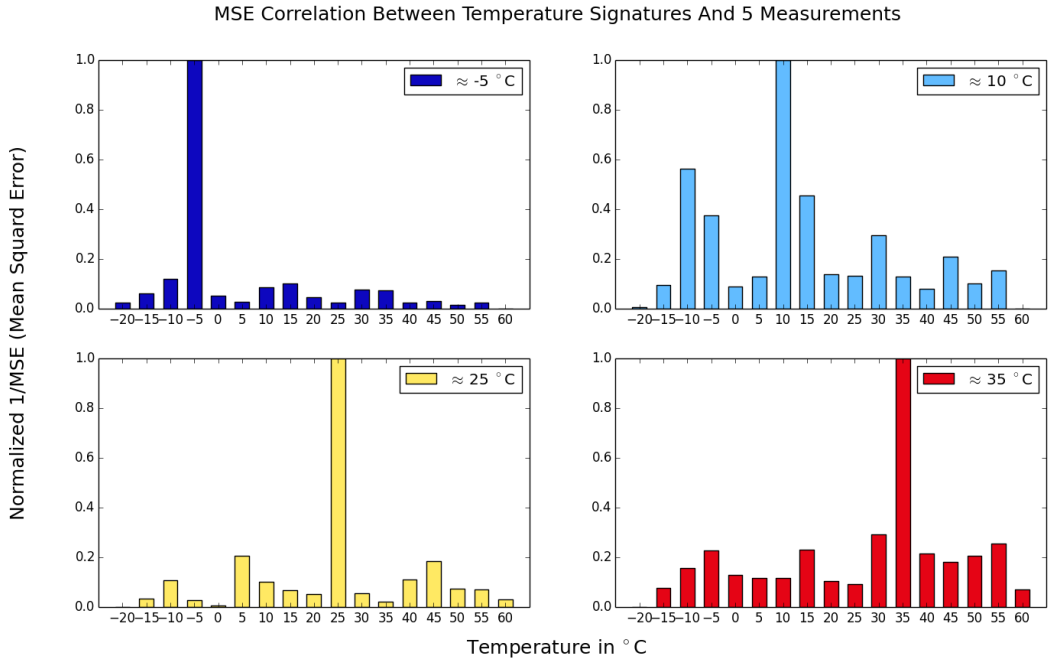

Figure 32: Correlating temperature measurements with the well defined database covering 17 temperature measurements recorded previously.

Conclusion

Concluding our observations, the binary response of the RO Sum PUF has shown a very stable temperature behaviour over a large temperature range. This ability

is very important for traditional PUF applications because we can flawlessly derive a key from the PUF independent of the surrounding temperature.
However, when observing the real valued response, we will see a slight drift within a decent range and without a clear direction. Unfortunately this cannot be easily used to determine the surrounding temperature and furthermore has to be considered in our solution. To do so, we simply enlarge the response deviation until the temperature variance is covered, which will unfortunately reduce the sensitivity for occurring attacks. To mitigate this, a more sophisticated approach was presented requiring a unique PUF characterisation for different temperatures.
Based on such an extended database, we can simply validate the current PUF behaviour against the best fitting one from the database. Indeed, this will introduce a larger development overhead, but allow us to keep the actual effect sensitivity and furthermore the ability to measure the environmental temperature sounding the device and determine malicious temperatures ranges.

5.6 Threats To Validity

Finally, we will discuss the validity of the results presented and problems during the experiments. Firstly, we can state that the observable effects suit our expectations and all of them could be reasoned to an underling principle. With respect to these principles and the observed changes, we can further argue that the recorded behaviour is mainly caused by the designated modifications. Hence, other influences (e.g. temperature) having only a negligibly impact on our measurement. Additionally, we have ensured the integrity of measurement data by using a checksum generated on the FPGA and validated within software.
The biggest issue we had during our experiments is caused by the quality of the RO Sum PUF, because we couldn't find the optimal way to implement it. To try the best, we considered and applied the design rules recommended by [19] and additionally evaluated the behaviour on our own. However, when creating the final bitstream the compiler will always introduce slight varieties we do not comprehend entirely and yield PUFs of different qualities. The main reasons for this might be the placement and interconnection variability of the ROs, which cannot be controlled more precisely and even though their ideal declaration will be complicated on proprietary hardware. Since we couldn't resolve this issue we have created a workaround, by simply redoing the synthesis until a satisfying configuration is found. Thereby, the quality can be rated by observing the total amount of boundary enlargements for single rounds during enrolment. If the value is steadily decreasing and stay below $\approx 5\%$ after some rounds (e.g. 20-50 iterations), a good solution is found. Otherwise, the configuration will be unstable and may never satisfy a suitable threshold. When ignoring this issue, we will observe an odd behaviour while monitoring the device. For instance by deactivating a malicious effect, we will observe that some responses require a long time to come back to their origin range or even change permanently. This will still allow to detect a possible attack, but might not allow to completely return into the initial state.

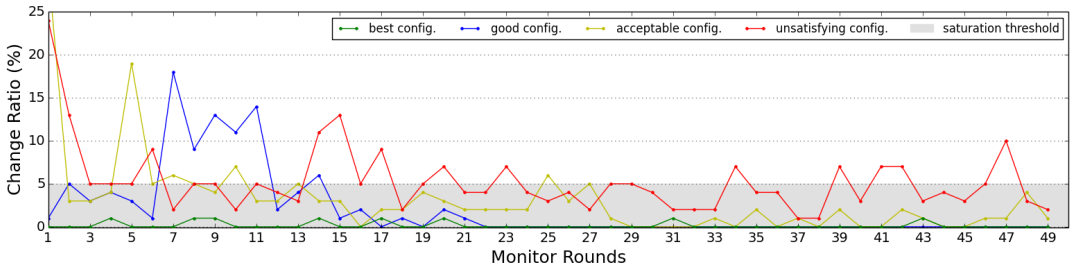

Figure 33: Enrolment behaviour for different bitstreams

For our experiments we have typically done 3 - 5 repetitions of the synthesis process which finally yield configurations with a steady change ratio and low response deviation at all. The created bitstreams for the described experiments can be found in the repository and must be used for a meaningful reproduction of our results. An entire and improved solution for this issue might be found by further optimisation of the implemented ROs or by realising them with full-custom ASIC design. However, so far the workaround satisfy our needs and yield RO Sum PUFs of high quality.

6 Conclusion and Future Work

In the last chapter we will give a short summary of our findings and present the milestone we have achieved within this thesis. Additionally we will discuss open problems and possible topics for future work.

6.1 Summary

At the beginning of this thesis, we have elaborated the various ways how secure devices can be attacked to reveal their inner secrets. By using non-, semi- or fully-invasive techniques an adversary can steal confidential data and harm asset owners in various ways, which is a big issue for the IC industry and its customers. The practical relevance and market value of this was discussed in Chapter 2 and can further be observed by recent efforts of device vendors, trying to integrate protection mechanisms against these attacks. Unfortunately, the proposed countermeasures will only address the problems in traditional ways, which is not sufficient by considering sophisticated attacks and their evolution yielding more automation and decreasing costs. Related to that, a more sustainable approach were presented in Chapter 2.6, which is using physically uncloneable functions linked to intrinsic properties of a device to change the way how secrets are stored in future. Indeed, this PUFs will introduce a new level of security and prevent several know attack types, but they can not prevent fraud in general. To overcome these limitations we have introduced a novel solution in Chapter 3, able to actively monitor the physical integrity of a device with an intrinsically linked PUF that cannot be spoofed or tampered with. Certainly this will not work out of the box with all types of PUFs, hence we initially derived the requirements for such an architecture and compared the best candidates related to that. The most promising one we could narrow down is the RO Sum PUF, which shows the best properties in terms of scalability, routability, participation and lastly a good observation capability though its real valued output. Furthermore, it satisfy all properties necessary for origin PUF applications, like authentication. Based on this architecture, we have elaborated effective evaluation methods and presented a generic architecture for monitoring a device in practice. Thereby we have considered online and offline capabilities and discussed their benefits, differences and limits in Chapter 3.3. To test the overall idea, we implemented a simplified prototype containing a scaleable PUF hardware implementation and software side evaluation components. The chosen tools, technologies, parameters and major issues during implementation are presented in Chapter 4. Based on that system, we created several experiments to prove our presented concept. In doing so, we observed the influence of Trojan insertions, system clock and supply voltage changes as well as laser and temperature based attacks. A comprehensive analysis and visualisation related to all these experiments is given in Chapter 5.

6.2 Conclusion

Concluding our findings, we can argue that all evaluated scenarios will cause an observable change by observing the real valued PUF output. The strongest impacts can be seen by attacks targeting the voltage supply and system clock because they will directly effect the evaluation logic of the RO Sum PUF. Trojan activation and occurring laser attacks will equally show strong changes, but generally require a sensitive and stable PUF implementation. The biggest issue we have to deal with is the low temporal resolution of the PUF, which is caused by the necessary sample time. Unfortunately, by exploiting this an advised attacker might be able to circumvent the detection capability for several cases. This is especially true for glitching attacks, which only occur for some clock cycles and hence are out of scope for the RO Sum PUF so far. Apart form this, also other attacks will happen which are not considered here and also new attack types will be revealed which cannot be considered today. With respect to this and to our evaluation the purposed idea might never be complete and furthermore has to deal with false positive and false negative errors.
From this point of view, it might be the best to combine an intrinsic solution with traditional countermeasures to integrate an additional security layer into the overall security solution. The actively monitored PUF allow us to detect attacks not covered by PUFs originally and hence will help us generate a more sound evaluation about the current device integrity by using both systems hand in hand. Additionally the intrinsic monitor can detect fraud and manipulation even if traditional countermeasures are already bypassed or spoofed, what cannot be done easily with an intrinsic solution. This will at least increase the effort, costs and skills to attack the device and hence minimise the risk of economic meaningful attacks. Besides that, the PUF can still be used for its origin purposes, e.g. authentication and secure key storage and hence promote the idea to a comprehensive security solution covering multiple threats. The major advantage of the proposed approach is the relatively small development overhead, when a satisfying PUF is already part of the projected security design. Its components can be reused and just have to be placed in the right way. Beyond that, only some further evaluation logic have to be added to handle the monitoring. A major tradeoff of the approach, is the additional power consumption introduced by the large usage of ring oscillators. The magnitude of this will strongly depend on the PUF size and the evaluation period used for monitoring, which should be selected according to the required security level. With respect to this discussion, the purposed solution might mainly address solutions with high security demands like secure cryptoprocessors, smart cards and similar devices. Especially the presented online validation capabilities might be able to introduce a new level of trust in applications which require remote verification of the device integrity.

6.3 Future Work

In this thesis, we haven't presented a market-ready solution of our vision, but we discussed and elaborated the practical and theoretical foundation for this. However, in future work one might find additional ways to improve the performance on a computational or architectural level.
For instance, one improvement we can imagine to address the low temporal resolution, might be to evaluate multiple challenge in parallel based on the same RO data each. This way a short occurring effect can influence several CRPs and cause a detectable change during monitoring for short effects. Indeed, this will introduce additional costs in terms of power and area consumption, but might allow us to solve one of our biggest issues. Besides that, one will find a more sophisticated recombination function for the PUF component targeting a stronger or comparable sensitivity for malicious effects as the one purposed by [32] we have used. Furthermore, even other PUF architectures can be found or developed, which might satisfy the given requirements and allow to closely monitor the physical integrity. One of the most promising alternatives we have investigated is the BR-PUF [11] and the PE-PUF [49], which indeed cover interesting properties, but were not as convincing as the RO Sum PUF.
Furthermore, the detection capability might be extended by applying new methods to evaluate the current PUF behaviour. Some interesting approach is presented in [53], using principal components analysis to separate data recorded for non-influenced and influenced oscillator networks. Another way is introduced in [36], where a supervised learning model is applied using support vector machines able to classify data of different influential situations. Currently, it is unclear if such a complex evaluation will allow fast enough decisions during monitoring, but even simple methods might benefit from those evaluations. Possibly, some challenges with unique properties can be found and used to evaluate single effects much better than before like discussed in Chapter 5.3.3. Apart from that, we can also improve the evaluation logic on an algorithmic level. For example, by starting an evaluation with a large set of challenges and dynamically focus on the most suspicious ones when malicious effects occur.
Finally all these steps will require further experiments to achieve a better understanding of limits and applications of active intrinsic monitoring. Equally, many other attack scenarios could be evaluated in practice, for instance to prove the resistance against a combination of attacks trying to leverage the detection capability of our solution. Besides that, also several hardware related issues, remarked in Chapter 4.3, can be resolved by testing the approach with custom ASIC technology.

Bibliography

[1] Ahmad-Reza Sadeghi, David Naccache. *Towards Hardware-Intrinsic Security, Foundations and Practice*. Springer-Verlag Berlin Heidelberg, 2010.

[2] Alexander Schloesser, Dmitry Nedospasov, Juliane Kraemer, Susanna Orlic, Jean-Pierre Seifert. Simple photonic emission analysis of AES. *Journal of Cryptographic Engineering*, 2013.

[3] Altera. Security Supervisor IP (SSIP) for Secure and High Assurance Systems. Brochure, `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/po/ssip_military_ss_6.pdf`, 2014.

[4] Altera. Cyclone IV FPGA Device Family Overview. `https://www.altera.com/en_US/pdfs/literature/hb/cyclone-iv/cyiv-51001.pdf`, 2016.

[5] Amir Moradi, Alessandro Barenghi, Timo Kasper, Christof Paar. On the Vulnerability of FPGA Bitstream Encryption against Power Analysis Attacks. *CCS '11*, pages 111–124, 2001.

[6] Amir Moradi, David Oswald, Christof Paar, Pawel Swierczynski. Side-Channel Attacks on the Bitstream Encryption Mechanism of Altera Stratix II Facilitating Black-Box Analysis using Software Reverse-Engineering. *Proceeding FPGA '13 Proceedings of the ACM/SIGDA international symposium on Field programmable gate array*, pages 91–99, 2005.

[7] N. A. Anagnostopoulos, S. Katzenbeisser, M. Rosenstihl, A. Schaller, S. Gabmeyer, and T. Arul. Low-temperature data remanence attacks against intrinsic sram pufs. 2016. `http://eprint.iacr.org/2016/769`.

[8] Artemios G. Voyiatzis, Dimitrios N. Serpanos. Active Hardware Attacks and Proactive Countermeasures. In *ISCC '02 Proceedings of the Seventh International Symposium on Computers and Communications*, 2002.

[9] Tom Caddy. *Encyclopedia of Cryptography and Security*. Springer US, Boston, Massachusetts, 2011.

[10] Charles Herder, Meng Day Yu, Farinaz Koushanfar, Srinivas Devadas. Physical unclonable functions and applications: A tutorial, 2014.

[11] Qingqing Chen, György Csaba, Paolo Lugli, Ulf Schlichtmann, and Ulrich Rührmair. The bistable ring PUF: A new architecture for strong physical unclonable functions. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2011*, pages 134–141, 2011.

[12] Clemens Helfmeier, Christian Boit, Dmitry Nedospasov, Jean-Pierre Seifert. Cloning physically unclonable functions. In *Proceedings of the 2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013*, pages 1–6, 2013.

[13] Clemens Helfmeier, Dmitry Nedospasov, Christopher Tarnovsky, Jan Starbug Krissler, Christian Boit, Jean-Pierre Seifert. Breaking and Entering Through the Silicon. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security*, CCS '13, pages 733–744, New York, NY, USA, 2013. ACM.

[14] D. Samyde, S. Skorobogatov, R. Anderson, J. J. Quisquater. On a new way to read data from memory. In *Proceedings - 1st International IEEE Security in Storage Workshop, SISW 2002*, pages 65–69, 2003.

[15] Dai Yamamoto, Masahiko Takenaka, Kazuo Sakiyama, Naoya Torii. Security Evaluation of Bistable Ring PUFs on FPGAs using Differential and Linear Analysis. In *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, 2014.

[16] Debdeep Mukhopadhyay, Rajat S. Chakraborty. *Hardware Security: Design, Threats, and Safeguards*. Chapman and Hall, October 29, 2014.

[17] Dmitry Nedospasov. *Security of the IC backside*. PhD thesis, Technische Universität Berlin, 2015. Available online at `http://users.sec.t-labs.tu-berlin.de/~nedos/Nedospasov_Thesis.pdf`.

[18] Dmitry Nedospasov, Shahin Tajik, Jean-Pierre Seifert. Lecture notes in Hardware Security. TU Berlin, Chair for Security in Telecommunications, Summer term 2015.

[19] Dominik Merli, Frederic Stumpf, Claudia Eckert. Improving the Quality of Ring Oscillator PUFs on FPGAs. In *Proceedings of the 5th Workshop on Embedded Systems Security*, New York, NY, USA, 2010. ACM.

[20] Drimer, Saar. Security for volatile FPGAs. *Rapport technique UCAM-CLTR-763, University of Cambridge*, 2009.

[21] Ed Peterson. Leveraging Asymmetric Authentication to Enhance Security-Critical Applications. Technical report, 2015.

[22] Eli Biham, Adi Shamir. The next Stage of Differential Fault Analysis: How to break completely unknown cryptosystems. 1996.

[23] Fatemeh Ganji, Shahin Tajik, Jean-Pierre Seifert. *Let Me Prove It to You: RO PUFs Are Provably Learnable*, pages 345–358. Springer International Publishing, 2016.

[24] Flavio D. Garcia, David Oswald, Timo Kasper, Pierre Pavlidès. Lock it and still lose it – on the (in)security of automotive remote keyless entry systems. *Proceedings of the 25th USENIX Security Symposium, 2016*, 2016.

[25] Hagai Bar-El, Hamid Choukri , David Naccache, Michael Tunstall, Claire Whelan. The sorcerer's apprentice guide to fault attacks. In *Proceedings of the IEEE*, volume 94, pages 370–382, 2006.

[26] Michael Hutter and Jörn Marc Schmidt. The temperature side channel and heating fault attacks. In *Lecture Notes in Computer Science*, 2014.

[27] IEEE Spectrum. The Hunt for the Kill Switch. Available online at `http://spectrum.ieee.org/semiconductors/design/the-hunt-for-the-kill-switch`, 2008. Accessed: 2016-08-20.

[28] Jean-Jacques Quisquater, David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. *Smart Card Programming and Security*, pages 200–210, 2001.

[29] Jeroen Delvaux, Ingrid Verbauwhede. Fault injection modeling attacks on 65 nm arbiter and RO Sum PUFs via environmental changes. *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1701–1713, 2014.

[30] Job de Haas. Side Channel Attacks and Countermeasures for Embedded Systems. Presentation, Black Hat USA, August 2007.

[31] Karine Gandolfi, Christophe Mourtel, Francis Olivier. Electromagnetic Analysis : Concrete Results. *Cryptographic Hardware and Embedded Systems — CHES 2001*, pages 251–261, 2001.

[32] Meng-Day Yu, Srinivas Devadas. Recombination of Physical Unclonable Functions. *35th Annual GOMACTech Conference*, pages 1–4, March 2010.

[33] Dominik Merli, Dieter Schuster, Frederic Stumpf, and Georg Sigl. Semi-invasive EM attack on FPGA RO PUFs and countermeasures. In *Workshop on Embedded Systems Security*, pages 1–9, 2011.

[34] Microsemi. EnforcIT Security Monitor. Brochure, September 2015.

[35] Mohammad Tehranipoor, Farinaz Koushanfar. A survey of hardware trojan taxonomy and detection. In *IEEE Design and Test of Computers*, 2010.

[36] Nima Karimian, Fatemeh Tehranipoor, et al. Genetic Algorithm for Hardware Trojan Detection with Ring Oscillator Network (RON). *2015 IEEE International Symposium Technologies for Homeland Security (HST)*, 2015.

[37] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Advances in Cryptology — CRYPTO '96*, 1996.

[38] Paul C. Kocher, Joshua Jaffe, Benjamin Jun. Differential Power Analysis. *Advances in Cryptology — CRYPTO '99*, 1666:388–397, 1999.

[39] Phuong Ha Nguyen. Efficient Attacks on Robust Ring Oscillator PUF with Enhanced Challenge-Response Set. DATE '15 Design, Automation and Test in Europe Conference, 2015.

[40] R. Anderson, M. Bond, J. Clulow, S. Skorobogatov. Cryptographic Processors-A Survey. *Proceedings of the IEEE*, 94(2):357–369, 2006.

[41] Roel Maes. *Physically Unclonable Functions: Constructions, Properties and Applications*. Springer-Verlag Berlin Heidelberg, 2013.

[42] Sergei P. Skorobogatov. *Semi-invasive attacks - A new apporach to hardware security analysis*. PhD thesis, University of Cambridge, 2005. Available online at `http://www.cl.cam.ac.uk/TechReports/`.

[43] Sergei Skorobogatov, Christopher Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *Lecture Notes in Computer Science*, volume 7428 LNCS, pages 23–40, 2012.

[44] Shahin Tajik, Dmitry Nedospasov, Jean-Pierre Seifert, Clemens Helfmeier, Christian Boit. Emission analysis of hardware implementations. In *Proceedings - 2014 17th Euromicro Conference on Digital System Design, DSD 2014*, pages 528–534, 2014.

[45] Shahin Tajik, Heiko Lohrke, Fatemeh Ganji, Jean-Pierre Seifert, Christian Boit. Laser Fault Attack on Physically Unclonable Functions. In *FDTC 2015, Saint Malo, France*, 2015.

[46] Stefan Mangard, Elisabeth Oswald, Thomas Popp. *Power Analysis attacks: Revealing the secrets of smart cards*. Springer US, 2007.

[47] Stephen M. Trimberger, Jason J. Moore. FPGA Security: Motivations, Features, and Applications. *Proceedings of the IEEE, August 2014*, pages 98–117, 2014.

[48] Steve Trimberger. Security in SRAM FPGAs. *IEEE Design and Test of Computers*, 24(6):581, 2007.

[49] Xiaoxiao Wang and M. Tehranipoor. Novel Physical Unclonable Function with process and environmental variations. *Design, Automation and Test in Europe (DATE), 2010*, pages 1065–1070, 2010.

[50] Xilinx. AXI Hardware ICAP - Product Description. `http:%www.xilinx.com/products/intellectual-property/axi_hwicap.html%overview`. Accessed: 2016-08-20.

[51] Xilinx. Info Sheet Security Monitor IP (SECMON). Brochure, `http://www.xilinx.com/support/documentation/product-briefs/security-monitor-ip-core-product-brief.pdf`, 2015.

[52] Xuan Zhang, Alyssa B. Apsel. A Low-Power, Process-and- Temperature-Compensated Ring Oscillator With Addition-Based Current Source. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2011.

[53] Xuehui Zhang, Andrew Ferraiuolo, Mohammad Tehranipoor. Detection of Trojans Using a Combined Ring Oscillator Network and Off-Chip Transient Power Analysis. *ACM Journal on Emerging Technologies in Computing Systems 9, 3, Article 25*, September 2013.

Appendix

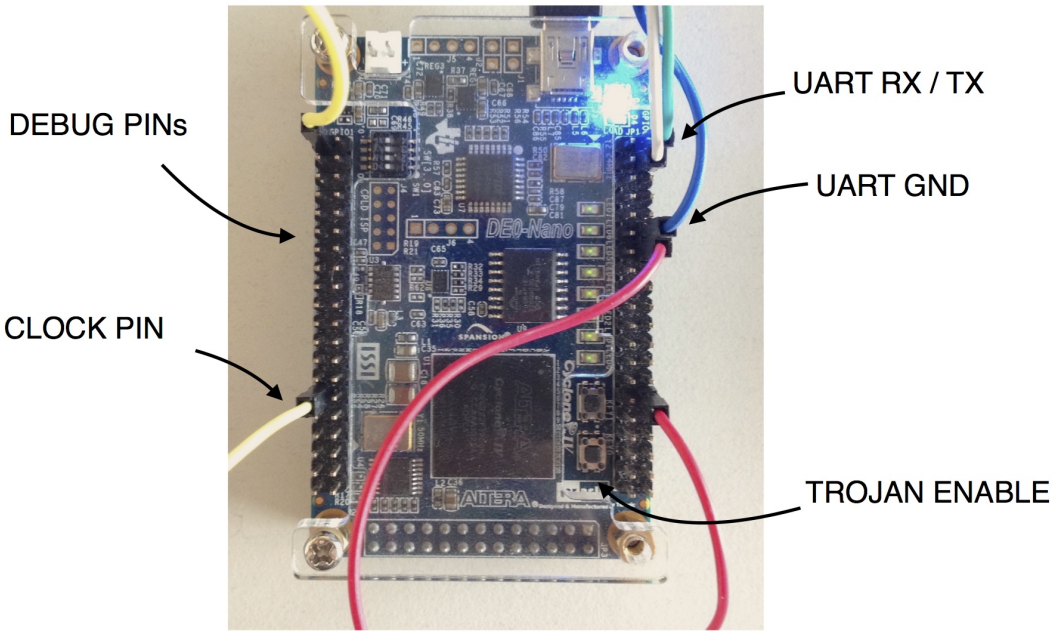

Figure 34: Wiring for the DE0-Nano board

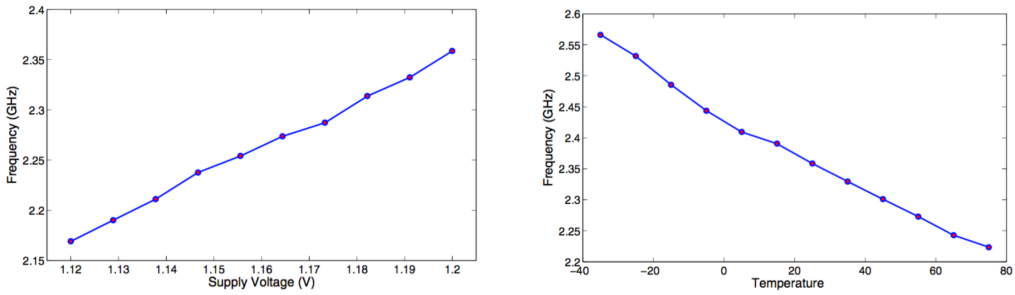

Figure 35: Relationship between RO frequency and Voltage / Temperature [49]

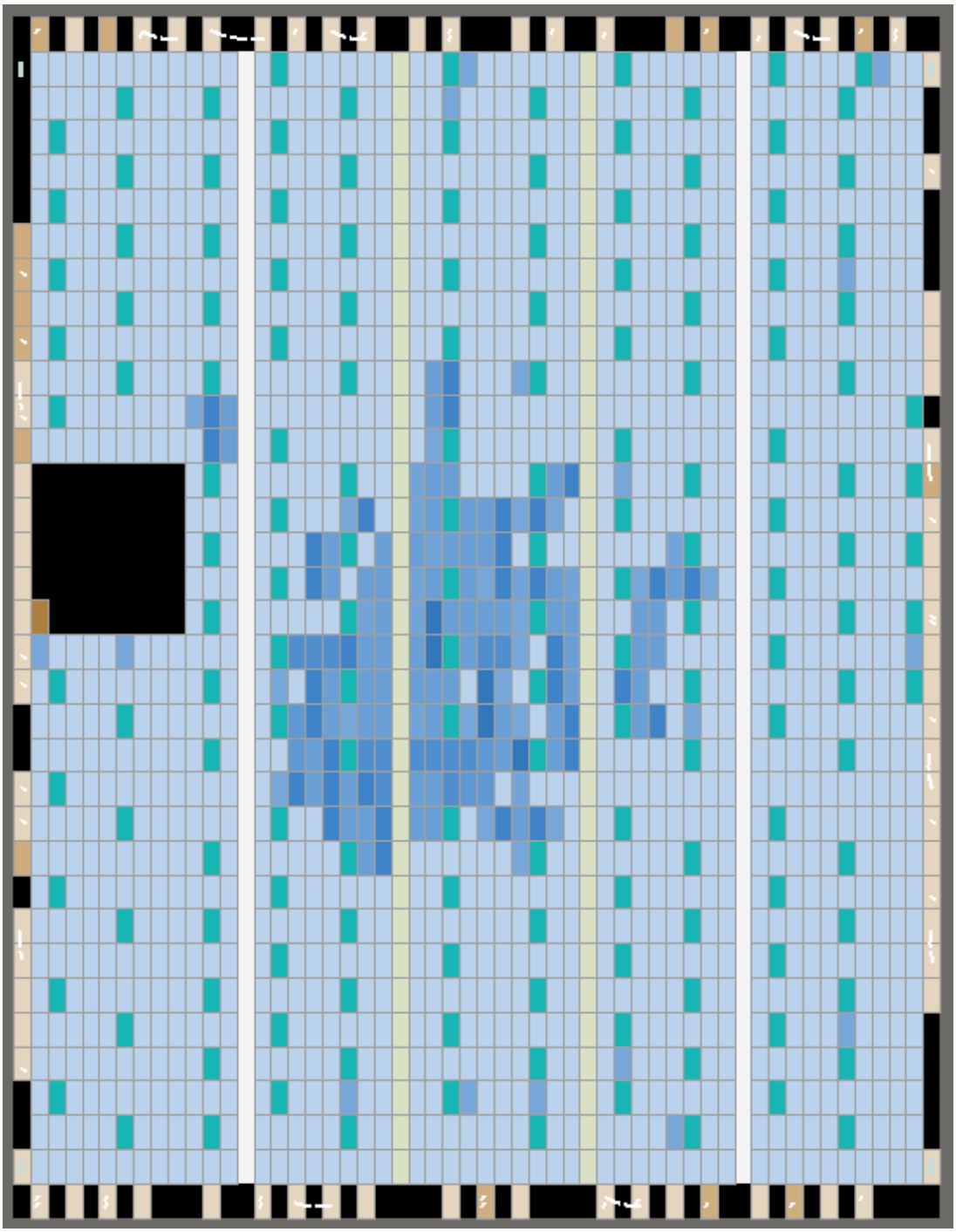

Figure 36: Screenshot from Quartus II Chip-planner for RO Sum PUF implementation with 32 ROs using the line-by-line placement strategy.

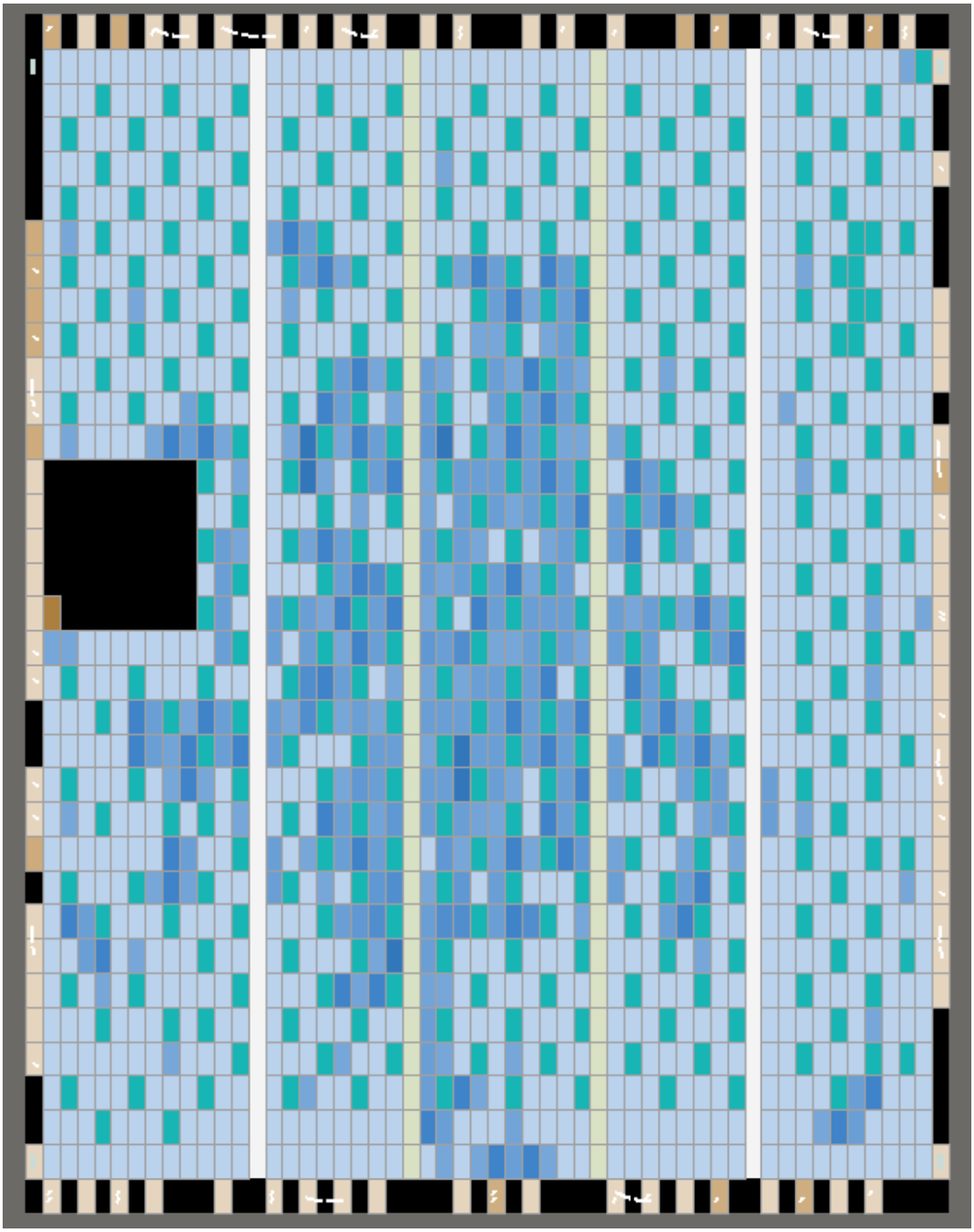

Figure 37: Screenshot from Quartus II Chip-planner for RO Sum PUF implementation with 64 ROs using the line-by-line placement strategy.