FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Remote Side-Channel Attack on AES
Student:	René Gál
Supervisor:	Dr.-Ing. Martin Novotný
Study program:	Informatics
Branch / specialization:	Computer Engineering 2021
Department:	Department of Digital Design
Validity:	until the end of summer semester 2024/2025

Instructions

Become familiar with the AES cipher, correlation power analysis, and remote attacks on the AES cipher. Implement a sensor in the FPGA to measure power consumption and repeat a selected remote attack. A suitable target platform may be, for example, the Zynq platform.

Electronically approved by prof. Ing. Hana Kubátová, CSc. on 6 February 2024 in Prague.

Bachelor's thesis

REMOTE SIDE-CHANNEL ATTACK ON AES

René Gál

Faculty of Information Technology
Department of Digital Design
Supervisor: Dr.-Ing. Martin Novotný
May 16, 2024

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Gál René. *Remote Side-Channel Attack on AES*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

List of Tables

List of code listings

I would like to thank my supervisor, Dr.-Ing. Martin Novotný for his time, guidance and ever-present optimism. My sincere gratitude goes towards MSc David Spielmann, for his willingness and helpful advice regarding the RDS. I also extend my heartfelt gratitude to my friends for their emotional support, and to Matej specifically, for always being on call to listen to my complaints. I also want to thank my family for their support throughout my studies and my grandma for her pastry.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 16, 2024

Abstract

This thesis explored the replicability of a remote side-channel attack. Initially, an analysis was conducted on current FPGA on-chip power sensors and their approaches to executing remote side-channel attacks. Subsequently, the routing delay sensor was identified as a suitable candidate and deployed on the Digilent Basys 3 (Xilinx Artix-7) board. Using this sensor, measurements were carried out, enabling a successful remote power analysis side-channel attack on the last round of the AES-128 cipher, which was evaluated using correlation power analysis. Thus, the replicability of a remote attack aimed at obtaining a secret key in the presence of the routing delay sensor has been confirmed. Furthermore, all Python scripts used throughout were made openly available in the attachments.

Keywords remote power analysis, remote side-channel attack, hardware trojan, FPGA, routing delay sensor, on-chip sensor, correlation power analysis, AES, Basys 3

Abstrakt

Táto práca skúmala replikovateľnosť vzdialeného útoku postrannými kanálmi. Na začiatku sa vykonala analýza súčasných senzorov spotreby na čipoch FPGA a ich prístupov k uskutočňovaniu vzdialených útokov postrannými kanálmi. Následne sa ako vhodný kandidát identifikoval routing delay senzor, ktorý sa nasadil na vývojovej doske Digilent Basys 3 (Xilinx Artix-7). Pomocou tohto senzoru sa vykonali merania, ktoré umožnili úspešnú vzdialenú odberovú analýzu postranným kanálom na posledné kolo šifry AES-128, ktorá sa špecificky vyhodnotila pomocou korelačnej odberovej analýzy. Potvrdila sa tak replikovateľnosť vzdialeného útoku zameraného na získanie tajného kľúča v prítomnosti routing delay senzoru. Všetky použité pomocné programy v jazyku Python boli okrem toho voľne sprístupnené v prílohe.

Klíčová slova vzdialená odberová analýza, vzdialený útok postranným kanálom, hardvérový trojský kôň, FPGA, routing delay senzor, senzor na čipe, korelačná odberová analýza, AES, Basys 3

List of abbreviations

AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
AWS	Amazon Web Services
CLB	Configurable Logic Block
CPA	Correlation Power Analysis
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DSP	Digital Signal Processing/Digital Signal Processor
FF	Flip-Flop
FPGA	Field Programmable Gate Array
GCC	GNU Compiler Collection
HRDS	Horizontal Routing Delay Sensor
HW	Hardware
LUT	Look-Up Table
RDS	Routing Delay Sensor
RO	Ring Oscillator
RPA	Remote Power Analysis
RTL	Register-Transfer Level
SSD	Solid-State Drive
SW	Software
TDC	Time-To-Digital Converter
TDL	Tapped Delay Line
VHDL	Very High Speed Integrated Circuit Program Description Language
VITI	Voltage Induced Time Interval Sensor
VRDS	Vertical Routing Delay Sensor

Chapter 1
Introduction

In a world of distributed cloud computing, field-programmable gate arrays (FPGAs) are finding widespread use throughout data centers. Their ability to adapt the hardware for a different purpose by simply uploading a new bitstream allows them to accelerate varying workloads on a single platform. This plays an essential role for many companies, as they save time, space, and money by not having to use as many application-specific integrated circuits. The highly parallel architecture and low power consumption of FPGAs also make them suitable for use in neural network hardware, where they are already being used by Microsoft or Baidu. Companies such as Amazon (AWS) and Microsoft (Azure) also provide rentable FPGA fabric to customers. This creates an obvious attack vector for adversary tenants interested in stealing encrypted information.

Power analysis side-channel attacks have been known since the late 1990s. Their main idea is to uncover secret information through observed variations in the power consumption trace of a chip. Traditionally, these power analysis attacks were performed only with physical access to the device, as they were dependent on (often expensive) high sampling rate measuring instruments, such as oscilloscopes.

The recent emergence of remote side-channel attacks poses a newfound threat, as physical access to the fabric is no longer necessary. Since the sensor used for collecting power traces in this type of attack is directly operating from within the FPGA fabric, there is also no additional associated cost regarding the attacker. Additionally, with the novelty of these methods and their indistinguishability from ordinary use of the chips, they are proving to be hard to detect. Thus, there is a need to further explore the topic, since only with a good understanding of the attacks can countermeasures be implemented. Hence, the following goals were formulated.

The first goal of this thesis is to become familiar with the AES cipher and traditional side-channel attacks, to get acquainted with correlation power analysis, and to implement it. A further goal is to research remote side-channel attacks and choose the most feasible one. Afterward, the work deals with replicating a selected remote attack on a known platform. An attack using a sensor first introduced by Spielmann et al. [1] was chosen.

The rest of this thesis is organized as follows: Chapter 2 is dedicated to explaining the main concepts used throughout the thesis. Analyzing existing approaches to remote attacks and choosing an appropriate one is dealt with in Chapter 3. Chapter 4 explains the steps taken to replicate the chosen attack, elaborates on the attack method, and presents the partial and final results. Chapter 5 reflects on the work done and proposes ideas for future work and the final Chapter 6 summarizes the work and achieved results.

Chapter 2
Preliminaries

The following chapter explores key concepts essential for understanding the thesis's core objectives. It introduces most of the main elements necessary for the thesis's main goal: a platform, a cipher, an attack method, and an instrument. While the platform, cipher, and attack method are discussed in detail within this chapter, the instrument—a power consumption sensor—is introduced in Chapter 3. This groundwork is crucial for providing readers with the context needed to comprehend the subsequent analysis and findings more easily.

2.1 FPGA Use Cases

At first glance, field-programmable gate arrays may seem as if they were merely the training version of ASICs; their higher price especially, may induce the perception, that they are not very applicable in the industry. However, FPGAs are not useful solely for educational purposes and for training future ASIC designers. They are widely utilized in low-volume production operations, such as specialized testing hardware in automotive [2], the space industry (satellites) [3], the defense industry (radars, weapons systems, electronic warfare) [4] etc., where investing in the manufacture of ASICs would be financially unwise and sometimes outright nonsensical. Especially, given that these industries additionally capitalize on the remote programmability of FPGAs—it would be ill-advised to send a multi-million dollar creation into orbit, without the ability to remotely repair and upgrade its firmware. Semiconductor companies recognize this demand as well, hence, there are the likes of XQR Kintex™ UltraScale FPGAs and RT PolarFire® FPGAs, which are dedicated space-grade radiation hardened FPGA families [5, 6].

Better yet, high-volume production is not devoid of them either—they have been previously deployed in consumer electronics: the Samsung Galaxy S5 and the iPhone 7 have both included a low-power Lattice FPGA within their motherboards. Samsung's SSD, marketed as SmartSSD, incorporates the AMD Kintex™ UlltraScale+ FPGA line, of which the objective is to free up the loads on host hardware, for it to rather handle other tasks [7]. In the near future, with the trend of locally-ran neural networks, it would not be of much surprise, if FPGAs returned to the smartphone sphere, to further accelerate these tasks.

Moreover, due to the end of Moore's law and the breakdown of Dennard scaling, data centers are transitioning from homogeneous and CPU-oriented systems towards heterogeneous architectures incorporating GPUs[1] and FPGAs [8, 9]. Microsoft extensively employs FPGAs within its data centers, deploying them for a wide array of task, spanning from web searches to network cryptography and machine learning [10]. Similarly, Baidu leverages FPGAs within its data centers to accelerate deep neural networks [11].

[1] Graphical Processing Units

Consequently, cloud service providers offer rentable multi-tenant FPGAs for customers. In their interest, FPGAs also hold many amiable features, such as:

- highly parallel architecture
- energy efficiency
- versatility
- simple programmability

2.2 FPGA Architecture

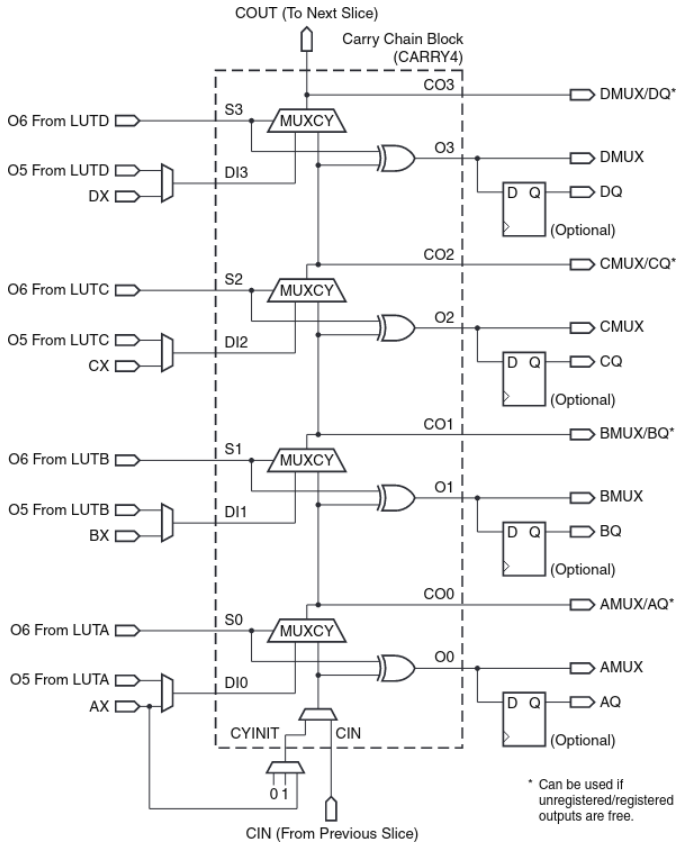
An FPGA generally comprises of:

- Configurable Logic Blocks (CLBs)
- Programmable interconnect[2]
- On-chip memory
- Special purpose logic blocks
- I/O blocks

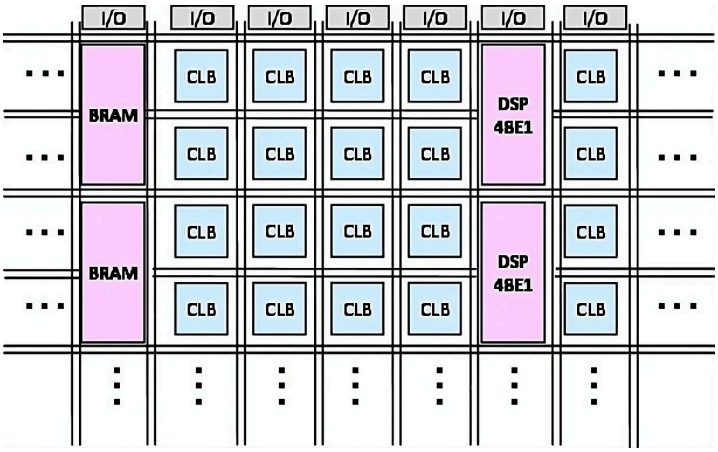
The CLBs are highly flexible logic cells, further composed of FPGA slices, which are in turn composed of logic elements, such as Look-Up Tables (LUTs)[3], Flip-Flops (FFs), and fast carry logic (a variation of a carry look-ahead adder). They are arranged in a two-dimensional grid and are interconnected by the programmable interconnect. The edges of this grid are occupied by I/O blocks, intended for peripheral communication.

The Digilent Basys 3 (XC7A35T-1CPG236C) [12], used in this thesis, contains two slices per CLB, with each slice containing 4 LUTs, 8 FFs, and fast carry logic (there is also the SLICEM, which contains a superset of logic elements, namely DRAM[4] and shift registers). However, there are also more specialized slices. An example would be the Digital Signal Processing (DSP48E1) slice, which contains a pre-adder, a 25x18 multiplier, an adder, and an accumulator. [13] An important takeaway of this section, for our purposes, is the internal structure of the slices and the observation that the internal FPGA grid is heterogeneous (observable in Figure 2.2).

[2] Also called programmable routing resources or routing interconnect.
[3] LUTs serve as logic function generators.
[4] Distributed Rapid Access Memory



■ **Figure 2.1** Internal structure of an FPGA slice, with the fast carry logic (CARRY4) highlighted [13]



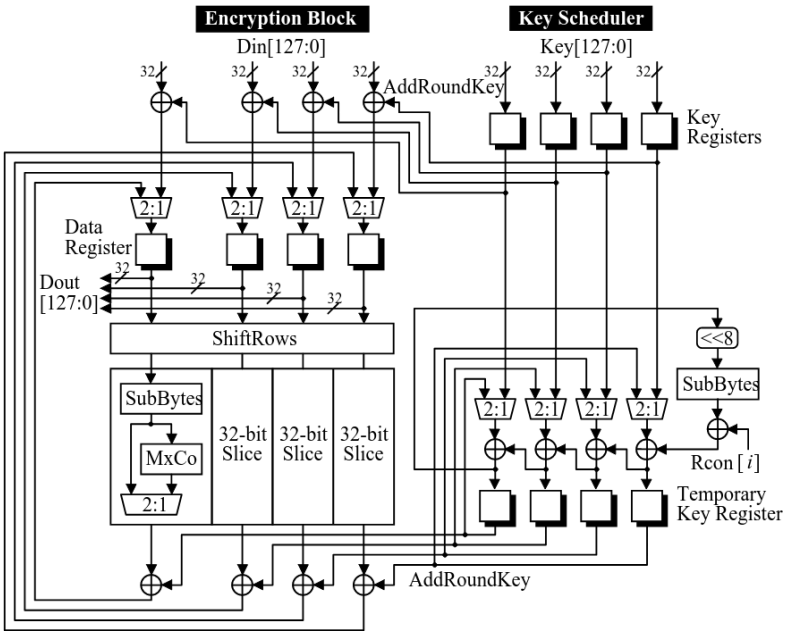
■ **Figure 2.2** Internal structure of an FPGA—a heterogeneous grid of logic blocks, with the column structure differing from the row structure [14]

2.3 Advanced Encryption Standard

Originally Rijndael, chosen as the Advanced Encryption Standard in November of 2001, is a block cipher with a block size of 128 bits and key sizes of 128, 192, and 256 bits, consisting

of 10, 12, and 14 rounds respectively. Each round is comprised of a substitution layer (SubBytes[5]), two linear mixing layers[6] (ShiftRows and MixColumns), and an XOR with the round key (AddRoundKey).

Throughout the operations of each round, the key scheduler generates round keys from the initial encryption key. The initial encryption key is transformed to produce a set of unique round keys, enhancing encryption strength. This process involves byte substitution, cyclic shifting, and mixing operations, ensuring each round key contributes to encryption complexity. The number of rounds determines the number of round keys generated. An important thing to note is, that the key scheduling algorithm is invertible, thus, by getting hold of the last round key, the key scheduling can be reversed and the initial encryption key can be trivially acquired. [15, 16]



■ **Figure 2.3** Openly available AES-128 hardware module performing encryption rounds in parallel [17]

In this thesis, the focus was on the 128-bit version, implemented in hardware, where each round is applied in a single processor cycle and on every byte of the plaintext simultaneously. The open-source AES-128 module created by the Aoki Laboratory of the Tohoku University [17] was subjected to an attack later in Chapter 4.

▶ Note 2.1. *When talking about an encryption key in this thesis, the initial encryption key, before any key scheduling occurs, is meant.*

2.4 Side-Channel Attacks

The taxonomy of hardware attacks is quite broad, hence, for the necessary context of this thesis, non-invasive ones will be briefly described. Paul Kocher's work [18], published in 1996, played a crucial role in the formalization and widespread recognition of side-channel attacks in modern cryptography, emphasizing their significance. Side-channel attacks do not target the mathematical properties of cryptographic algorithms but rather their implementations, bypassing the theoretical strength of the underlying algorithms.

[5] Also referred to as an SBox.
[6] The last round contains only a single one.

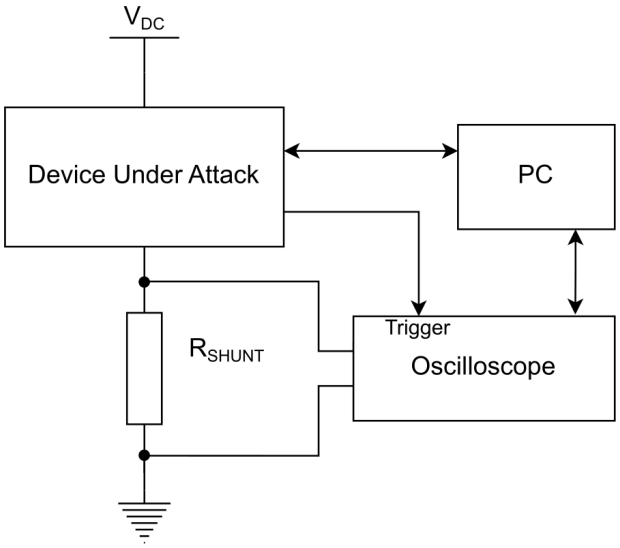
The implementation of a cipher can leak secret information in various ways. Timing attacks, for instance, focus on the time taken to complete steps of an algorithm based on processed data. Electromagnetic radiation (EMR) attacks involve reading the EMR radiated from a device, exposing information about its internal operations. In addition to timing and EMR, other side-channel attack methods include power analysis, which inspects the power consumption patterns of a device during cryptographic operations, as well as temperature and acoustic attacks, which exploit variations in temperature or sound emitted by the device during computation.

Fault-injection attacks work by introducing deliberate malicious faults into the target device, to bring it into a set of states from which private internal information can be acquired. There are multiple fault-injection techniques, for example (power) supply attacks, clock attacks, heating attacks, and radiation attacks. [19] Each of these methods provides insights into vulnerabilities in cryptographic implementations that adversaries can exploit.

2.4.1 Power Analysis

In power analysis attacks, the attacker attempts to reveal secret information stored inside the device, based on its power consumption. This targeted information is typically a secret key used for a cryptographic algorithm. Power analysis attacks work because the cryptographic devices' power consumption depends on the executed cryptographic algorithms' intermediate values.

The conventional measurement setup, depicted in Figure 2.4, assumes physical access to the cryptographic device and consists of an oscilloscope measuring the voltage drop of the attacked device over a shunt resistor connected to the ground. In experimental conditions, the attacked device usually transmits a *start encryption* signal to the oscilloscope. The measured power trace is then processed in a computer, using methods such as simple power analysis, differential power analysis, the later mentioned correlation power analysis (see Section 2.5), or others.



■ **Figure 2.4** A typical setup for a power analysis—an oscilloscope measuring voltage over a shunt resistor in the ground path of the cryptographic device [20]

2.4.2 Remote Power Analysis

The remote power analysis attack (RPA) is a novel approach to the conventional power analysis attack. Schellenberg et al. [21] extended the use of time-to-delay converters (see Subsection 3.3.1.1) to power variation measurements and presented the first RPA attack reported in the

literature. Resembling the power analysis, RPA focuses on measuring small power variations within a device in time.

However, instead of relying on physical access to the device and using dedicated high sampling rate power measuring instruments (oscilloscopes), RPA specifically relies on a power consumption sensor constructed using the already available logic resources within an FPGA. As is commonplace with regular power analysis attacks, RPA ordinarily employs correlation power analysis to acquire the secret key from the measured power traces. It is also typical, to use some kind of a side-channel evaluation metric, such as key rank, signal-to-noise ratio, or guessing entropy (as is also the case in this thesis), to more precisely assess the success of the attack [1, 22].

Remote attacks also introduce some challenges. First, there is the challenge of capturing ciphertexts, which is shared with traditional power analysis attacks. However, what is different, is, that during a conventional power analysis attack, the attacker has physical access to the device and therefore has no trouble accessing the measured power traces. In RPA, this can become more challenging in some cases, though in the threat model assumed for this thesis (see Subsection 3.2.1), where the attacker has control of the FPGA fabric with the on-chip sensor, this should not be a problem. Further, with the reduced sensitivity of the power monitors used for RPA, compared to an oscilloscope, noise can become a significant limitation (as is later illustrated in Chapter 3). There is also the issue of proximity, as constraining oneself to a part of an FPGA distant from the attacked circuit can make the transient voltage fluctuations harder to capture. This is predominantly the case for large FPGAs used in data centers [22, 1].

2.5 Correlation Power Analysis

The concept of correlation power analysis (CPA) was originally introduced by Brier et al. [23], it is based on a power consumption model of the running device at some point in time. This analysis is based on certain secret bits and input bits (plaintext) that change for each power trace. In the case of AES, the moment most utilized to perform the attack is after the first SBox (see Figure 2.5), where the linearity between the plaintext and the key is broken and with the finest granularity (the key bytes are not yet mixed). Similarly, the attack can be targeted before the last SBox to obtain the last round key (see Figure 2.6).

The most commonly used consumption models are the Hamming distance between two relevant values or the Hamming weight of a particular value. These models are then correlated with actual power consumption measurements (which are therefore a requisite for using this approach) using the Pearson correlation coefficient. [24] Notable differences between these attack variants are highlighted in table 2.1.

Table 2.1 High-level differences between CPA attack variants; the encryption key refers to the master key before any key scheduling occurs

CPA variant	First round attack	Last round attack
Input	Plaintext + Traces	Ciphertext + Traces
Output	Encryption key	Last round key
Model	Hamming weight	Hamming distance

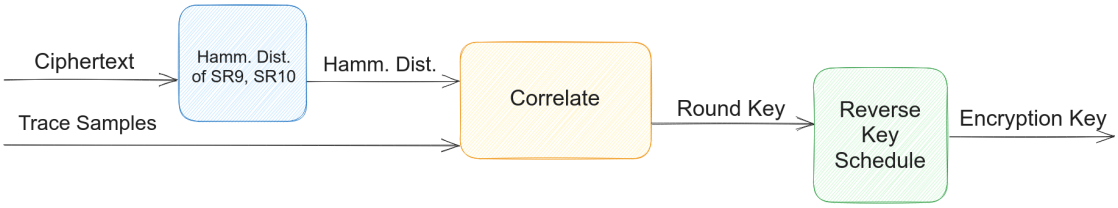


Figure 2.6 CPA attacking the last round of AES—ciphertexts and power traces are at the input, with the last round key at the output

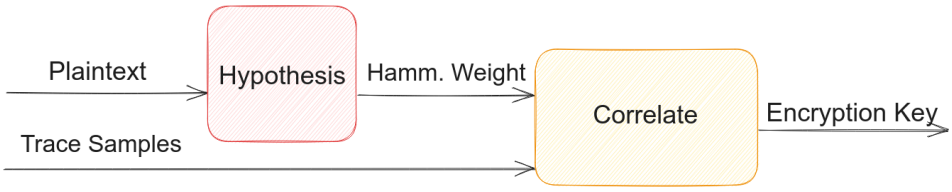


Figure 2.5 CPA attacking the first round of AES—plaintexts and power traces are at the input and the output (if successful) is the encryption key straightaway

The attack on the first round is fairly straightforward—key hypotheses are made by combining (XOR) each plaintext with key guesses, followed by the calculation of the Hamming weight of these hypotheses. These hypotheses are then correlated with the power consumption traces. On the other hand, the last round attack slightly complicates the process, since it is necessary to calculate the Hamming distance of the last two state registers, that is, the state register after the 9th round and 10th round (which is the ciphertext itself). To acquire values corresponding to the same byte of the processed data in both state registers, first, it is necessary to reverse the ShiftRows operation on the ciphertext for the processed byte. This yields the first value, of which the Hamming distance will be calculated. Next, to acquire the second value (the state register after the 9th round), it is required to, first, reverse the AddRoundKey operation (using all of the key guesses respectively) for the given ciphertext byte, and second, inverse the SBox for it. This is illustrated in the Python code 1. The last thing to note is, that the Hamming distance of two values is equal to the Hamming weight of those two values XORed together.

```
def hamm_distance(ciphertext_row: np.array, byte_idx: int, keyguess: int):
    byte_idx_shifted = ShiftRowInverse[byte_idx]
    state10 = ciphertext_row[byte_idx_shifted]

    AddRoundKeyByte = keyguess ^ ciphertext_row[byte_idx]
    state9 = SBoxInverse[AddRoundKeyByte]

    return hamm_weight(state9 ^ state10)
```

Code listing 1 Hamming distance calculation of state registers 9 and 10 in the last round attack of CPA

2.6 Guessing Entropy

In the context of a side-channel attack, guessing entropy characterizes the amount of the remaining work of the attacker when the attack fails to reveal the correct key [20]. Originally defined as:

▶ Definition 2.2. *The lower bound to the average number of successive guesses, required with an optimum strategy until one correctly guesses the value of a discrete random variable X [25].*

In practice, it is calculated as the expected position of the correct key within an array of guesses, sorted according to a value of a chosen distinguisher[7], in a descending order, where the first position represents the most probable key candidate, and the last position represents the least probable candidate [20].

Due to the purely experimental nature of this thesis, the encryption key is known at the time of the attack, therefore its expected position is also precisely established. The calculation is done for each of the key bytes, and then the arithmetic average of the calculated values is declared as guessing entropy. As for the distinguisher, the Pearson correlation coefficient was chosen for this thesis, as its calculation is already performed for the purposes of the CPA.

[7] Distinguisher $D^k(o_{x_1}, \ldots, o_{x_q}; x_1, \ldots, x_q)$ is defined as an absolute value of the statistic that is used to distinguish the correct key during the attack [20].

Chapter 3
Analysis

Initially, the common FPGA configurations will be examined (see Section 3.1). It is crucial to understand them to identify their specific vulnerabilities and to be able to carry out an attack on them.

Next, the various threat models that exploit these vulnerabilities will be delved into (see Section 3.2). Scenarios involving attacks between FPGAs themselves and attacks targeting the CPU from a compromised FPGA will be explored.

Following up, the focus will be shifted to the available approaches towards remote attacks themselves in Section 3.3, specifically to the sensors used to carry the attacks out. Various design concepts will be examined, including, but not limited to, Tapped Delay Line (TDL) and its variants, as well as frequency counters.

Finally, Section 3.4 will delve into routing delay sensors— a specific class of sensors that uses FPGA's routing multiplexers and the routing interconnect as its main element. Different types of routing delay sensors will be explored, such as Vertical Routing Delay Sensor (VRDS), Horizontal Routing Delay Sensor (HRDS), and the particular Routing Delay Sensor (RDS), which disposes of the tapped delay line. The ultimate aim of this chapter, following an in-depth examination of on-chip power consumption sensors, is to identify the most suitable one for the purposes of this thesis.

3.1 Platforms

The following section explores the common FPGA configurations, including standalone FPGAs, SoC FPGAs, and soft-core CPUs. The main focus is on their use cases and vulnerabilities, as well as the differences between them.

3.1.1 Standalone FPGAs

The widely known and deployed variant of an FPGA board is a standalone FPGA, also referred to as a discrete FPGA. Data centers—where FPGAs are finding widespread use, thanks to their highly parallel architecture, programmability, and energy efficiency—already sit on a large amount of processing power thanks to high-end server-grade CPUs, and therefore do not have the need for SoC boards with both a dedicated CPU and an FPGA. The standalone FPGA is used for tasks that require high parallelism, such as machine learning, data processing, and encryption [10, 11].

3.1.2 SoC FPGAs

FPGA fabric may be integrated into a single SoC that includes other components such as general-purpose cores and GPUs, as in Xilinx's Zynq Series or Intel's SoC-FPGAs. These SoC architectures are likely to be deployed for mobile and embedded systems thanks to their smaller footprint and lower power consumption.

The SoC FPGA is a more complex system than the standalone FPGA, as it includes a processing core, which can be used for control algorithms, user interfaces, or even for encryption. This introduces a new vulnerability, as the power delivery network of the FPGA fabric is shared with the CPU core, which can be exploited by an attacker to breach the encryption key [26].

3.1.3 Soft-Core CPUs

While highly parallel tasks in the cloud are accelerated using FPGAs, developers still sometimes rely on the general computing power of CPUs, for example in some user event-driven control algorithm applications, where the development and maintenance is considerably easier in software than in hardware. This creates a need for a different solution from dedicated server-grade CPUs or even SoC chips within the same board, as the inter-chip communication of these architectures introduces too much latency. As a solution to this, soft-core CPUs, such as AMD's MicroBlaze [27], Intel's Nios [28] or the open-source PicoRV [29] and ZipCPU [30] prove themselves useful for designers. [31]

In practice, these are IP cores instantiated directly within the FPGA fabric, that communicate with the rest of the FPGA hardware normally using the AXI or the open-source (usually hobbyist) Wishbone bus.

Even though there are available and used countermeasures in place, to prevent unauthorized access between IP cores, the soft-core CPU shares the vulnerability of multi-tenant FPGAs (see Subsection 3.2.1), since it is instantiated within the attacked FPGA fabric and so the power delivery network is shared as well.

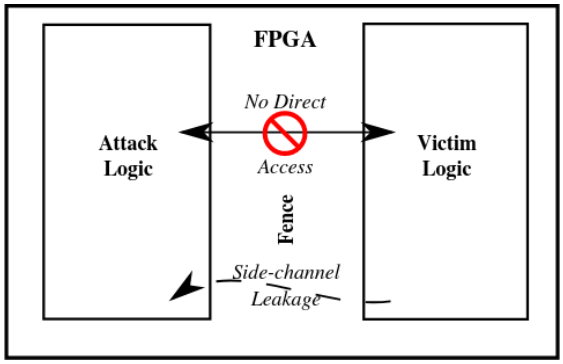
3.2 Threat Models

The following section will explore the various threat models that exploit the vulnerabilities of the aforementioned FPGA configurations. The main goal is to become acquainted with possible attack scenarios and the potential vulnerabilities of the FPGA configurations.

3.2.1 Multi-Tenant FPGA

Also known as an FPGA-to-FPGA attack or an intra-FPGA attack, this threat model describes a situation in which an FPGA is deployed in the cloud and shared among multiple users (usually thanks to virtualization techniques). Each user shares a portion of the FPGA fabric and has access to external interfaces through dedicated FPGA logic, deployed by the service provider [32].

An important factor concerning side-channel attacks is, that in such cases the users share a power delivery network of the chip through which leakage occurs (see Figure 3.1). The victim encrypts data using a cipher's hardware implementation and sends the ciphertexts over a public channel, which is observable by the adversary. The adversary also needs access to a channel used for offloading the measured power traces for further analysis (and subsequent breaching of the secret key).



■ **Figure 3.1** Multi-tenant FPGA where the victim deploys a legitimate design while the attacker deploys a malicious one, both within their respective fabric, separated by a 'fence' of unused configurable logic blocks [26]

3.2.2 FPGA-to-CPU Attack

This case refers to a model where the attacker deploys a malicious design onto an FPGA fabric that shares its power delivery network with a victim core where a software crypto algorithm is running. Possible real-life examples of this model can be: an SoC FPGA (see Subsection 3.1.2), a soft-core CPU running within a compromised FPGA fabric (see Subsection 3.1.3), or even a compromised standalone FPGA that shares the same board as other processing cores performing encryption tasks. The attacker's goal is to breach the encryption key of the victim core by measuring the power variations within the board's power delivery network.

3.3 Power Consumption Sensors

The fundamental concept that is used across on-chip power consumption sensors is constant: they measure the signal propagation delay caused by voltage fluctuations within the circuit's power delivery network, which are in turn caused by a sudden increase in switching activity from synthesized logic [33, 34]. Some achieve this using Ring Oscillators (ROs), while others opt for a tapped delay line or routing resources. The quantified signal propagation delay is then used to infer the secret key of the victim core in remote power analysis attacks.

Ring oscillators are used in the power monitor based on frequency counters by M. Zhao [26], whereas tapped delay line is used in Time-to-Digital Converters (TDCs) [33], the Voltage Induced Time Interval Sensor (VITI) [22], and both the HRDS and VRDS introduced by Spielmann et al. [1]. Generally, TDCs have a higher sensitivity, while the others require fewer place and route constraints and are therefore easier to deploy.

A special case of these sensors is the routing delay sensor (RDS)—a sensor that utilizes routing resources of the FPGA, such as routing multiplexers and interconnect. It does not require as many constraints as TDCs while achieving better results than ring oscillator-based power monitors. As is shown in [1], even though the TDC is more sensitive to voltage fluctuations, the RDS comes out on top in terms of efficiency. To be precise, the average number of power traces required to obtain n bits of the key is lower, which is among the most interesting properties in the context of an attack.

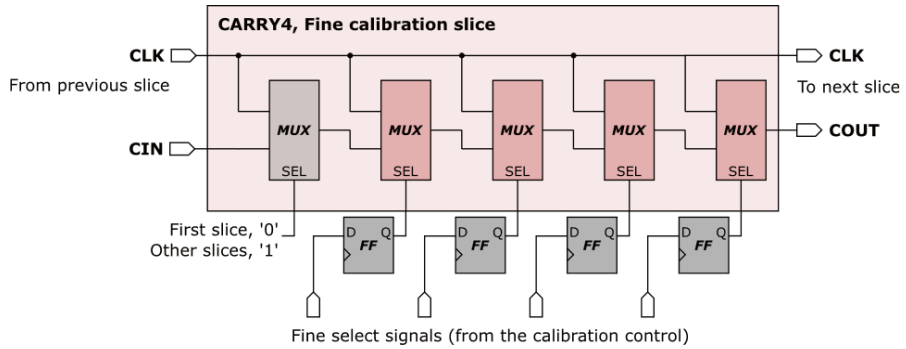
3.3.1 Tapped Delay Line-Based Sensors

This category of sensors utilizes a line of delay elements, where between each delay element lies a register used for sampling the distance a signal has propagated through the delay line.

A typical sensor based on this principle comprises of the following: an initial delay line, the tapped delay line, and sampling registers. Some extra logic for self-calibration can also be present (as is the case for VITI). Tapped delay line-based sensors represent state-of-the-art sensors for high sensitivity and high-resolution power consumption measurements. The choice of a delay element depends on the sensor's requirements, such as sensitivity, area, and calibration, and environmental constraints .

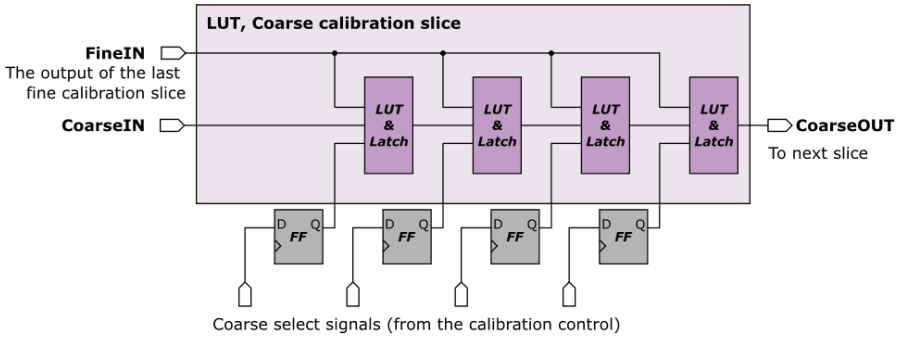
The initial delay line (see *init* in Figure 3.5) serves two purposes: saving the area of the sensor and calibration. Firstly, the signal delay does not change enough to affect the complete delay line. Having the logic to tap the entire delay line would therefore not serve any purpose. Secondly, the real delay of the logic elements the sensor consists of is not known at design time. To account for this, some calibration has to take place, either through manual addition (or subtraction) of delay elements [21], or by—the more flexible—software calibration [35, 1] of the number of delay elements. The whole point of this ordeal is to a) ensure the clock signal reaches the observable delay line within the clock cycle and b) the tapped delay line does not become saturated. In other words, the clock signal can not overshoot the tapped delay line and neither undershoot it [21].

In practice, the initial delay is implemented using a combination of coarse delay elements (see Figure 3.3), and fine delay elements (see Figure 3.2) [21, 33]. As the names would suggest, the fine delay elements provide finer control of the calibration process and are typically implemented using a series of fast carry logic elements—the CARRY4 primitives (see Figure 2.1) [36]. On the other hand, the coarse delay is usually made up of 'slower' elements[1], such as look-up tables (LUTs) [37] and latches. The key objective, when deciding on the type of delay element to be used, is to try to equally balance the final precision of the calibration process and to minimize the area used up by the sensor.



■ **Figure 3.2** Fine initial delay line slice [31]

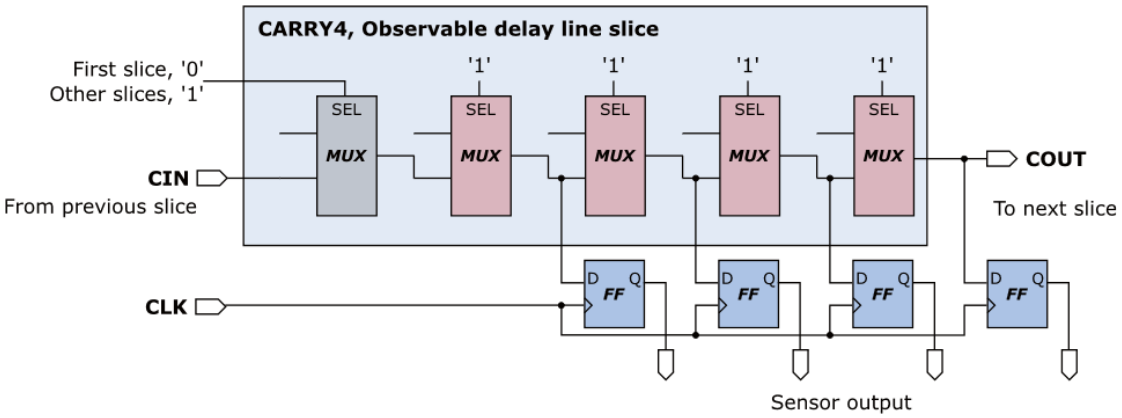
[1]Ones that introduce a longer delay into the signal.



■ **Figure 3.3** Coarse initial delay line slice [31]

The tapped delay line (see *Delay line* in Figure 3.5), is where the actual variations in signal delay are reflected and sampled. Between each delay element within the tapped delay line lies a register used for measuring the distance the *1* has traveled. Some implementations approach this as a thermometer code, with extra logic in the form of a priority encoder, to account for 'bubbles' of *0*s within the line [38]. Others process the register values as a Hamming weight—the number of logic *1*s in the full width of the output registers [35].

The delay elements within the tapped line are either implemented using a series of fast carry logic elements—the CARRY4 primitives (see Figure 3.4), as is the case for the TDC (see Subsection 3.3.1.1), by a series of LUTs, as in the case of VITI (see Subsection 3.3.1.2), or by routing resources in routing delay sensors (see Subsection 3.4). This difference arises mainly from two different properties of these sensors: the TDC targets high sensitivity, which asks for more, fast elements, while VITI, which came after the TDC, sacrifices the higher sensitivity and favors ease of deployment. The particular case of routing delay sensors aims for a certain balance.



■ **Figure 3.4** Tapped delay line, also called the observable delay line and its implementation using CARRY4 elements, common for the TDC [31]

3.3.1.1 Time-to-Digital Converters

Simply put, TDCs are commonly used to measure the time interval between a start pulse and a stop pulse. They transform analog time intervals into a digital output and subsequently find an application in many fields, such as high energy and particle physics, for time of flight measurement and others [39]. They can also be used as low-cost on-chip temperature sensors, supply voltage sensors [40], or even for glitch attack detection [41]. In the context of a remote attack, the start and stop pulse are of the same origin—the clock signal. The pulse that triggers the registers

arrives phase shifted. The other pulse that is sampled for propagation delay travels through the delay line.

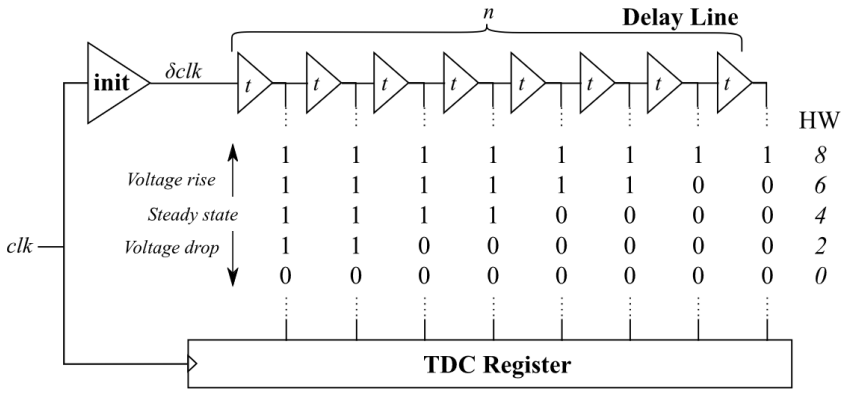


Figure 3.5 Schema of a TDC sensor using Hamming Weight of the registers as the final output [35]

A significant disadvantage of the TDC is its placement constraints. For consistency in the spacing between the delay elements, it is necessary to place them using the `RLOC` constraints [42] in combination with VHDL's `generate` statements [43]. This constraint ensures consistent spacing between delay elements along the lengthy delay line, which spans multiple FPGA slices. Due to the structure of the Configurable Logic Blocks (CLBs) within an FPGA[2], it is also necessary to constrain the placement of the TDC vertically across an FPGA column.

3.3.1.2 VITI

The design concept of the VITI sensor aims to achieve the following objectives:

- maximize portability
- minimize footprint
- avoid:
  - combinational loops
  - long carry chains
  - latches

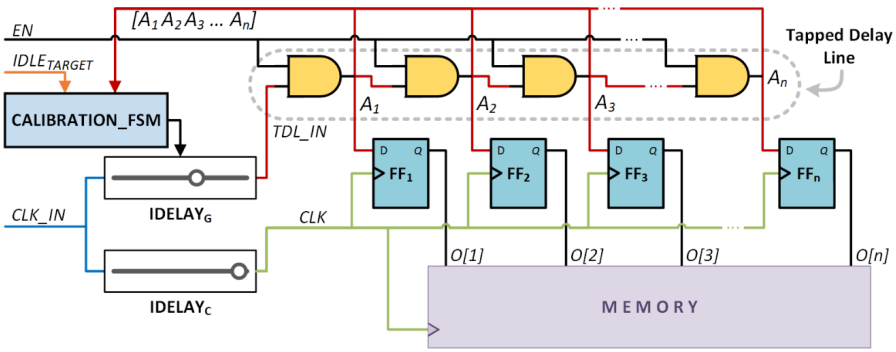
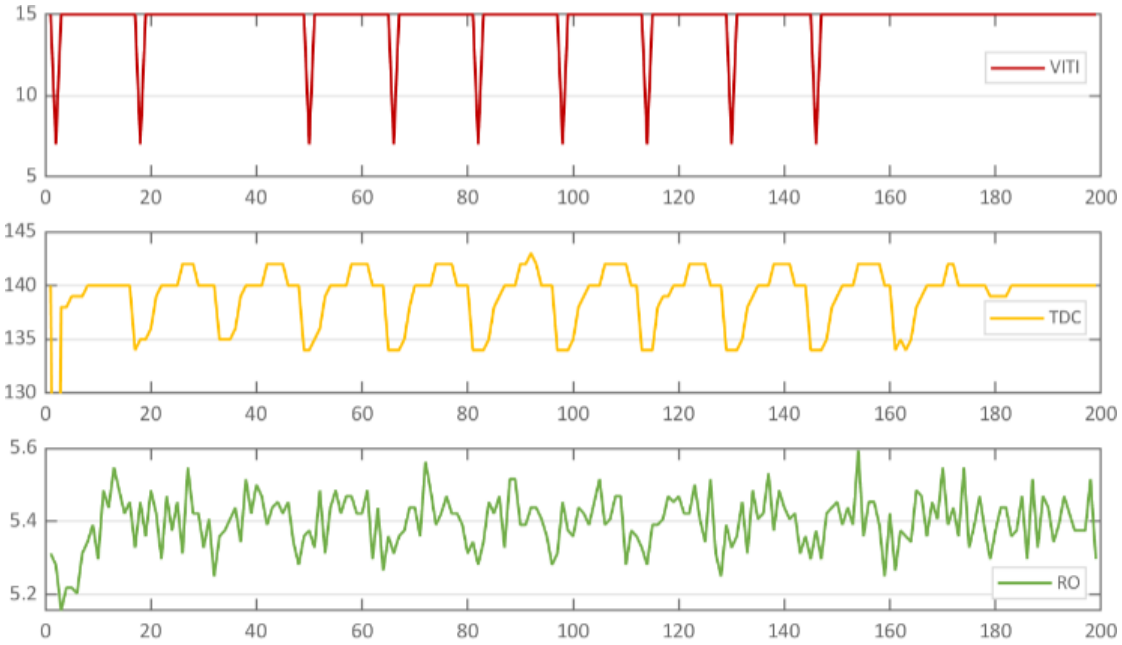


Figure 3.6 The VITI sensor uses LUTs in its tapped delay line [22]

[2]More specifically the structure of the carry chain within a slice.

VITI uses adjustable delay elements, flip-flops, and LUTs in the tapped delay line (Figure 3.6). Reasons for having these constraints are the following: VITI tries to be deployable in environments where detectable ring oscillator structures (see Subsection 3.3.2) are forbidden (such as one that would comply with the proposed design rule checks of Sugawara et al. [44]), and bulky carry chain structures of the TDC do not fit into the area-constrained FPGA fabric.

A noteworthy characteristic of VITI is its self-calibration capability, which allows it to adapt to varying situations, such as increased power consumption, temperature changes, or inconvenient placement in relation to the attacked circuit. In addition, it assures that the sensor can be easily moved from one device to another, making it highly portable. Udugama et al. [22] also claim, that VITI consumes 1/16th of the area compared to RO sensors and 1/4th compared to the TDC of Schellenberg et al. [21]. In order to achieve all of these features, VITI sacrifices resolution and sensitivity (Figure 3.7), primarily because of the usage of LUTs in its tapped delay line. Their research shows, the use of VITI by an attacker can result in the partial or complete compromise of the secret key of an AES-128 hardware module.

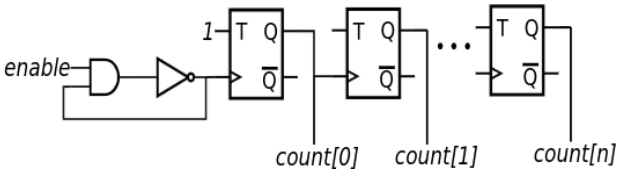


■ **Figure 3.7** VITI power trace compared to the TDC of [21] and the RO of [45]—VITI exhibits reduced resolution and sensitivity [22]

3.3.2 Ring Oscillator-Based Frequency Counters

Generally, a ring oscillator consists of an odd number of inverters in series, with an AND gate such that the output of the last inverter is fed back into the input of the AND gate, while the other input of the AND gate is an *enable* signal. Though research has shown, that such composition is dependent on temperature variations and also provides lower resolution [46, 47]. Zhao et al. have therefore opted for a design (Figure 3.8), which consists of a single inverter and an AND gate, with the intention of improving the previously mentioned characteristics [26].

The RO circuit clocks a counter that increments every oscillation period. As the ring oscillator oscillates much faster than the system clock, the counter is constructed as a chain of T-Flip-Flops (TFF) to eliminate slow carry chains. Alongside it runs a reference clock, triggered by the FPGA system clock. When the reference clock counter reaches a predetermined cycle count (C_{RefCLK}), the RO counter is disabled and its cycle count is read (C_{RO}). The RO frequency (f_{RO}) is then



■ **Figure 3.8** A single instance of a ring oscillator-clocked counter [26]

calculated so:

$$f_{RO} = C_{RO} \times \frac{f_{RefCLK}}{C_{RefCLK}} + \varepsilon \tag{3.1}$$

Where ε is the quantization error introduced by the phase difference between the two clock pulses. Consequently, due to the structure of the RO, we know that the oscillation frequency of the RO is inversely proportional to the time that a signal takes to propagate twice around the circuit—a measure of signal propagation delay. [26]

Their attack targeted both a standalone FPGA (see Subsection 3.2.1) and an SoC FPGA (see Subsection 3.2.2) hosting the RSA cipher. They have used 20 RO circuits (Figure 3.8) in multiple configurations. First, evenly distributed throughout the FPGA using place and route constraints, then physically separated from the encryption hardware, and then with no place and route constraints.

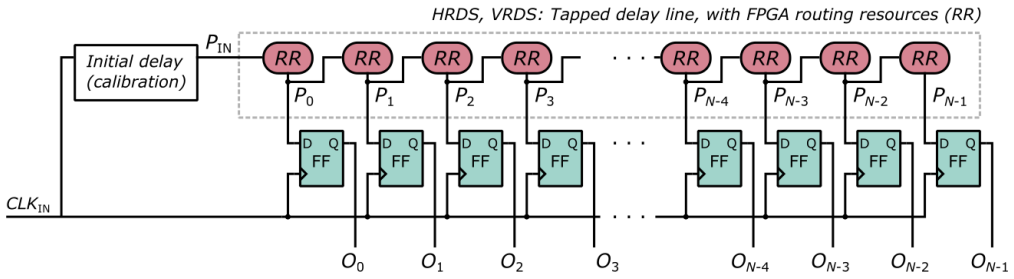
Regarding the intra-FPGA attack, when using power viruses to simulate interference and other switching activity on the chip, they required five times their original amount of power traces. However when the power consumption of the power viruses exceeded the power consumption of the RSA core, their filtering algorithm for the power traces stopped working and no longer recognized the RSA operations from the noise, paralyzing the attack efforts.

In conclusion, the RO-based approach is easier to implement across many FPGAs as it does not require the careful customization and placement constraints the delay line-based TDC sensor requires, however, it provides low sensitivity of transient voltage fluctuations. Research has also shown, ring oscillator structures are relatively easy to detect using bitstream checking techniques [48]. Furthermore, the detectable combinational loops (the heart of a ring oscillator) are already prohibited on commercial cloud services, as they can also be used as power viruses that damage the FPGA. [49]

3.4 Routing Delay Sensors

Attempting to merge the best characteristics of both approaches to the on-chip sensors—the ease of deployment of frequency counters and VITI, together with the sensitivity of TDCs—Spielmann et al. [1] build on the findings of Ahmed et al. [50]. Their research finds, that although the LUT-dominated paths of the FPGA were affected by the impact of power supply voltage on the signal propagation delay the most, routing multiplexers in the FPGA interconnect were affected as well. Motivated by these findings, Spielmann et al. [1] designed three variants of Routing Delay Sensors.

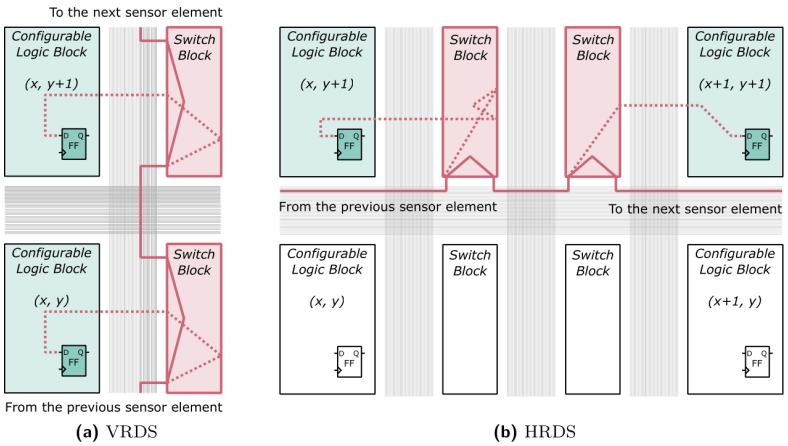
While the first two of them (HRDS, and VRDS) are primarily used for testing and showcasing the potential of routing multiplexers, the third one is the final and the most improved design of their work. A feature worth highlighting is the openly available nature of all routing delay sensor designs and experiments.



■ **Figure 3.9** RTL level diagram of both the VRDS and HRDS, with the CARRY4's of the TDC replaced by routing resources [1]

3.4.1 VRDS and HRDS

Both the HRDS and VRDS stick to the tapped delay line (see Figure 3.9), therefore their general design is very similar to the TDC. The main difference being, that they do not use CARRY4's, but the FPGA interconnect (best illustrated in Figure 3.10). An important observation for the HRDS is, that due to the structure of FPGA rows, which (contrary to the columns) also host BRAMs[3], DSPs[4], etc. [51], the wires between CLBs that do neighbor with BRAMs and DSPs are longer than wires between directly neighboring CLBs (see Section 2.2). This causes some routing resource blocks to have higher delay than others, which consequently reduces the sensitivity of this sensor variant compared to the VRDS.

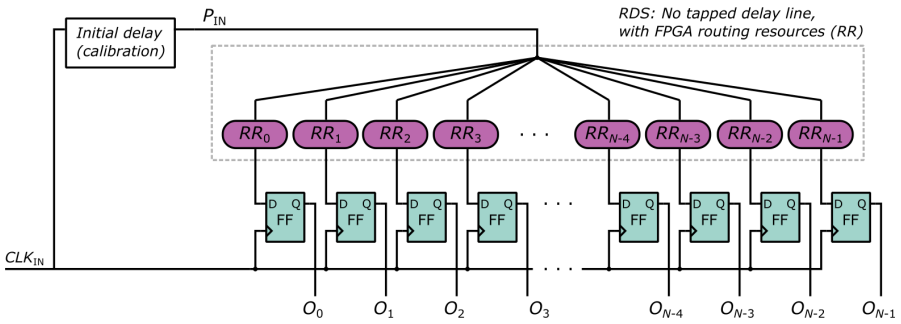


■ **Figure 3.10** FPGA interconnect used in the tapped delay line of the VRDS and HRDS [1]

3.4.2 RDS

The final routing delay sensor variant, simply called the RDS, features a new design modification, as it gets rid of the tapped delay line[5]. Instead, it connects each routing resource individually and lets the default router build the connections and the default placer decide on the locations of flip-flops in the output register, for it to help find suitable paths for the router (see Figure 3.11). This ultimately leads to a very simple implementation, where only the delay elements (LUTs, latches, CARRY4s) of the initial delay line need to be constrained. Finally, the

[3]Block Random Access Memories
[4]Digital Signal Processors
[5]The initial delay line, with its necessary placement constraints, remains.



■ **Figure 3.11** The final routing delay sensor—RDS—disposes of the tapped delay line [1]

RDS uses the Hamming weight of the output register as the power consumption representation.

Contrary to the ring oscillator-based power monitor and TDC of Schellenberg et al. [21], only a single instance of the RDS is necessary to be deployed for a successful attack, which further simplifies the design. All of the routing delay sensor variants have been deployed and tested on the Alveo U200 datacenter card, demonstrating a successful attack in a simulated real-world scenario.

A specific intricacy of the RDS is its calibration process, which is unlike any other previously mentioned sensor. Since the RDS does not feature a tapped delay line, it is no longer possible to tell where the clock edge precisely lands. Instead, the RDS calibration targets high variance in the power trace, specifically aiming for a high amount of toggling bits in the output register at the time of a voltage drop. [1]

Even though software calibration is an improvement over the manual adjustment of the number of elements in the initial delay line of early TDCs, the calibration process presents a drawback in comparison to the VITI sensor. The VITI sensor is self-calibrated, with the calibration occurring entirely in hardware. Whereas in the case of the RDS, the host PC software drivers are responsible for the calibration—monitoring the output register variance and calibrating the amount of coarse and fine delay elements in the initial delay line. This would introduce a complication in a real-world attack.

3.5 Discussion

The main goal of this chapter was to explore the available sensors for remote power analysis attacks on FPGAs. The TDC, VITI, RO-based frequency counters, and routing delay sensors were examined. The RDS combines the ease of deployment of frequency counters (only challenged by VITI) with the sensitivity of TDCs, while also providing a simple implementation and calibration process. The RDS was successfully deployed and tested on the Alveo U200 datacenter card by its authors, demonstrating a possible attack in a real-world scenario. Along with VITI, the RDS designs and software were made openly available.

The sensor characteristics carrying the greatest weight were sensitivity, calibration, place and route constraints, victim hardware, and availability. The introduced power monitors were compared in Table 3.1, with the most favorable candidates being the RDS and VITI. The RDS was chosen for its high sensitivity, ease of deployment, and open-sourcedness. The next chapter will delve into the implementation of the RDS on the Basys 3 FPGA, with the aim of replicating the successful attack scenario.

Table 3.1 Comparison of sensors used for remote power analysis

Sensor	RDS [1]	VITI [22]	HRDS [1]	VRDS [1]	TDC [21]	ROPM[1] [26]
Main component	RR[2]	TDL	RR[2]	RR[2]	TDL	RO
Sensitivity	High	Low	Low	Low	High	Medium
Calibration	SW	HW	SW	SW	Manual[3]	Manual
P&R Constraints	Few	None	Horizontal	Vertical	Vertical	None
Victim HW	FPGA	FPGA	FPGA	FPGA	SoC/FPGA	SoC/FPGA
Availability	Open	Open	Open	Open	Limited	Limited

[1] Ring Oscillator-Based Power Monitor
[2] Routing Resources
[3] Manual, although software-calibrated variants are available

Chapter 4

Method, Implementation and Results

This chapter is dedicated to detailing the implementation of the RDS attack, including the methodology employed and the resulting outcomes. The chapter follows a structured approach: it begins with an overview of the RDS components, followed by the implementation of the CPA script. Subsequently, the setup of both the board and host PC are explained. This is followed by a description of the power trace measurement process and the accompanying helper scripts. The replication of the attack is then outlined, concluding with the anticipated outcomes and the actual results achieved.

4.1 RDS Components

The original RDS project is generally well-documented. It implements the sensor and conducts experiments on three different boards, the Basys 3, Sakura X, and Alveo U200 datacenter card. I worked with the Basys 3 board, but the general project structure for each board is shared. The following text explains each component, specifically for Basys 3. The project is structured as follows:

- RTL hardware descriptions
- C software drivers
- CUDA C++ CPA code
- Python helper scripts

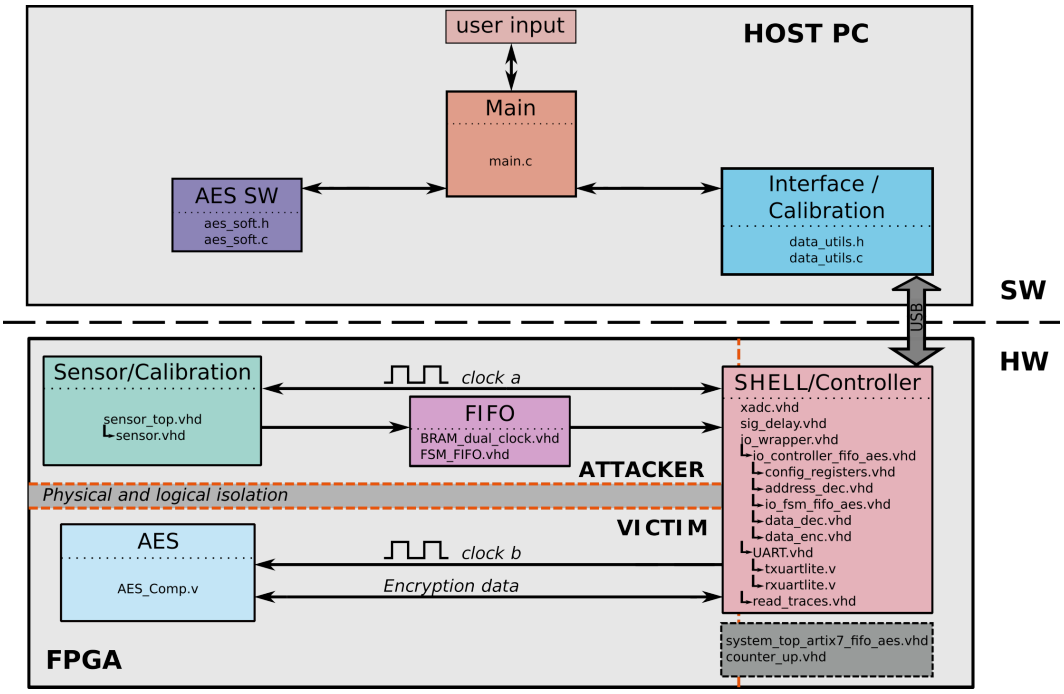
4.1.1 RTL Hardware Descriptions

system_top_artix7_fifo_aes.vhd serves as the top-level module of the project, coordinating interactions between its components. *AES_Comp.v* is a module dedicated to performing AES with a 128-bit key length, taking plaintext and key inputs, and producing ciphertext output. *io_wrapper.vhd* acts as the primary controller of the system, managing communication between the host and the system components, including AES encryption and sensor data handling. *reset_wrapper.vhd* generates reset signals for various components, including AES and sensors, based on external commands from the host PC. *xadc.vhd* implements an on-chip temperature sensor for temperature measurements unrelated to this thesis. Lastly, *sensor_top.vhd* is the

top-level file for *sensor.vhd*, which specifically implements the RDS. The architecture is best showcased in Figure 4.1.

4.1.2 C Software Drivers

The driver part comprises of: *aes_soft.c*, *data_utils.c* and *main.c*. *aes_soft.c* is simply a software AES implementation. *data_utils.c* implements the software calibration of the sensor, low-level serial interface, utility functions for parsing user input, and communication with the hardware AES module. It also implements testing of the hardware AES encryptions using the software AES. Some default values, such as the key and plaintexts, are specified in *main.c*. As expected, it primarily contains the general driver code that calls *data_utils.c functions*.



■ **Figure 4.1** RDS Basys 3 SW architecture (top), and HW architecture (bottom) [52]

4.1.3 CUDA C++ CPA Code

The provided code implements the last round attack on an NVIDIA GPU for high parallelization, as running the attack on the Alveo U200 card required multiple millions of power traces, which proved to be too slow to process on a CPU. For this thesis, there is no need for such a large amount of power traces, therefore my Python implementation of the CPA will suffice. An added bonus of using Python is the portability of the code since it does not require an NVIDIA GPU and very specific operating system requirements needed to run the CUDA toolkit.

4.1.4 Python Helper Scripts

The outputs from the CUDA program are later processed in the *calculate_keyrank.py* script. Before the CUDA code can be run though, it is needed to convert the sensor outputs to a different file format. These conversion are done using the *convert_ciphertexts.py* and *convert_traces.py*

scripts, though these conversion scripts miss a component for converting power traces to a binary file (which the *convert_traces.py* script requires).

4.2 CPA Script

Before any work with the sensors began, I had been tasked with implementing a CPA script using a platform (or a language) of my choice. One has multiple choices for such a task, for instance, MATLAB, Wolfram Mathematica, and Python. Since I am the most familiar with Python out of these three, I chose it for this thesis. Python's numerical library *NumPy* [53]—which equips Python with MATLAB-like functionality—has naturally been used generously. The script has been tested using data provided by my supervisor. I have implemented both the first round attack and later the last round attack. The *measurement.py* class has been created for representing power consumption traces to be later used with this CPA script.

4.3 Board Setup

After Vivado (the authors reference the 2018.3 version, having tried both, all has worked for me on the 2023.1 version as well), USB cable drivers and board files are installed [54], the sensor is set up using the provided Tcl script. All one needs to do is run the following in the Vivado command line:

```
cd basys3;  the full directory path is necessary
set project_name <project_name>
source create_project_AES50MHz_SENSOR200MHz.tcl
```

The Tcl script creates the project, which is then prepared for running the implementation. After running the implementation, clicking *Generate bitstream* generates the bitstream, which is then uploaded to the FPGA in the hardware manager. In my case, Digilent Basys 3 featuring the Xilinx Artix-7™ FPGA (XC7A35T-1CPG236C) was used. A thing worth mentioning is, that all works even if Vivado reports errors with the clock generator in the synthesized design window. That concludes the preparation of the hardware.

4.4 Host PC Setup

For setting up the host PC, where the sensor outputs are going to be exported, one only needs to compile the code in the *basys3/sw* directory, by running `make`. Afterward, the binaries are ready in the *basys3/sw/bin* directory, with the help message available after running `./interface -help`. The experiments for this thesis were conducted using the following command:

```
./interface -k 0 -pt 1 -t [encryption_count] -s -d /path/to/output_dir
```

This command starts an encryption with a constant key (`-k 0`), chained plaintexts[1] (`-pt 1`), specified number of encryptions/measurements (`-t [encryption_count]`), and the sensor traces saved (`-s`) to the given output directory (`-d /path/to/output/dir`). The initial key and plaintext are hardcoded in the *main.c* source code file. A notable point to highlight is, that the output directory must already exist when running the command, else the following error message will be displayed: "Error openning the output plaintexts binary!" Another thing worth mentioning is, that the command may need to be run as a superuser, in case the current user is not part of the dialout group. All my experiments were done on x86_64 systems, running Ubuntu 23.10 6.5.0-28-generic, GCC version 13.2.0.

[1] A mode where the ciphertext of the previous encryption is used as the plaintext in the following encryption, with a previously specified initial plaintext.

4.5 Power Trace Measurement

After having chosen the sensor, what I thought would be a straightforward process of replicating the objectively well-documented attack of Spielmann et al. [1], turned out to be a much lengthier and more tedious process than anticipated. Namely, the—in hindsight—trivial issue of starting the measurement. Despite an otherwise comprehensive description of steps taken to carry out the attack, at the time of my work, there was no mention of the purpose of switch 15 (SW15). Additionally, this switch's function was in a way obfuscated in its naming, as it was named the *reset* switch in Vivado's I/O planning window. After a significant amount of time debugging the host-side C code, responsible for initiating the attack through the UART bus, testing multiple Basys 3 boards, and verifying their UART (which I thought was problematic) using a dedicated UART test module, I have discovered the actual purpose of the *reset* switch (SW15)—initiating the AES encryption and the consequent power trace measurement.

Thus, an important warning is in place: *the leftmost switch on the Basys 3 board, labeled as SW15, must be in the ON position either before, or after executing the driver binary on the host PC*. Then and only then will the calibration phase start, after which the AES encryptions and power trace measurements follow (see Figure 4.2).

In the meantime, this elementary, yet for me unexpected step, has been cleared up in the documentation by the authors for future users. Another positive outcome of this has been the start of my communication with MSc David Spielmann, who has helped with other blocking issues later on.

```
Sensor trace transfer done!
Key: al a2 f7 54 18 c0 e4 c5 c6 91 d5 21 4d 22 76 ee
Plain text: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Cipher text: a5 3d 47 0a 47 e2 4b 60 85 08 fc b0 8e cf 10 87
Seconds needed to record the traces 48
Number of fails encountered: 0
```

Figure 4.2 Terminal output of the first successful encryption/measurement

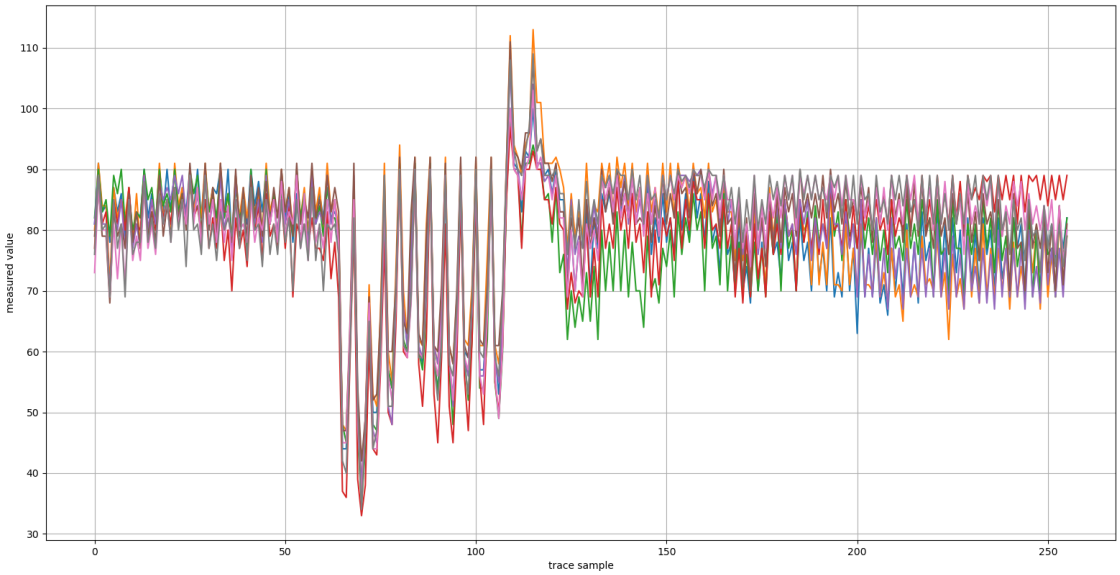


Figure 4.3 Full power consumption waveform of 8 power traces from the measurement of 150,000 traces total, with noise visible before and after the encryption

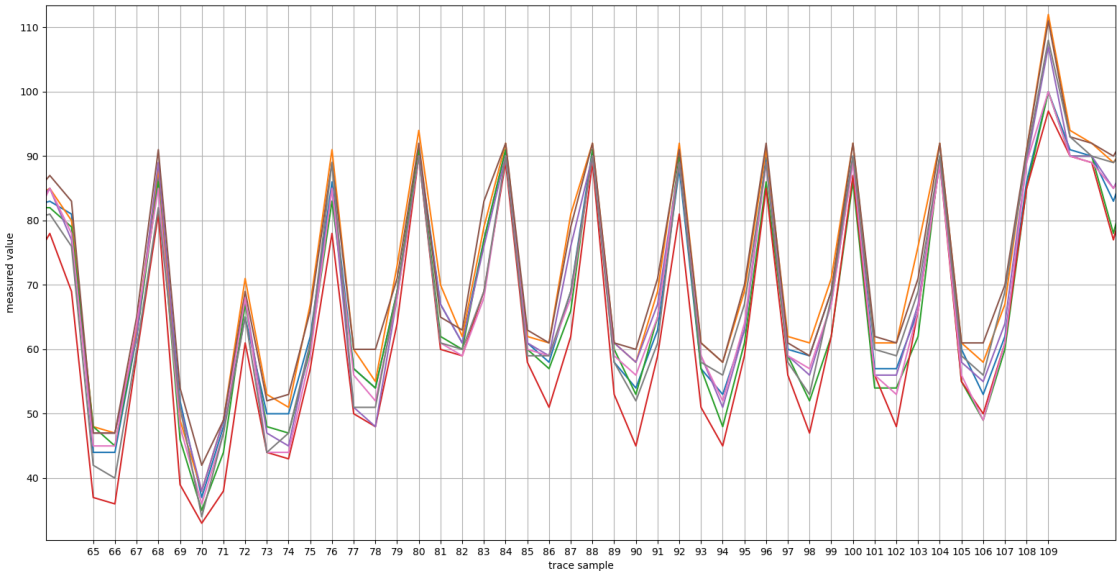


Figure 4.4 Zoomed plot of 8 power traces from the measurement of 150,000 power traces total—AES encryption rounds visible from trace 65 to 109

Figure 4.4 depicts the 10 AES rounds, with the first voltage drop visible at trace number 65. This would correspond to the first AddRoundKey and its write to the status register, which happens ahead of the 10 following AES rounds. Since the AES module runs at 50 MHz, while the RDS sensor runs at 200 MHz, each round is depicted as four trace samples. Therefore we would expect the encryption power trace to last around 44 samples. This is precisely what happens, as we can see around trace sample number 108. For completeness, Figure 4.3 depicts the full power trace of 256 samples the sensor outputs in its current configuration. Multiple measurements were conducted (some available in the attachments), for example, one of the measurements consisted

of 150,000 power traces and took 20 hours to complete. All of the measurements were recorded in a home environment at room temperature.

4.6 Helper Scripts

Even though the RDS project contained some file format conversion scripts, I decided to go for different formats and write my own. The provided scripts also do not seem to be complete. For example, the power trace conversion script already requires a binary file at its output, but only a comma-separated file is outputted by the sensor. The RDS sensor outputs the following:

- ciphertexts.bin
- plaintexts.bin
- keys.bin
- sensor_traces.csv
- ttest_fail.csv
- ttest_valid.csv

4.6.1 Output Standardization Script

For better readability, debugging and in order to use the data with my CPA script, I wrote the *standardize_rds_output.py* script, to convert the binary ciphertexts and plaintexts to a text file format with newline separated encryptions, where each encryption consists of sixteen `uint8` hexadecimal values.

The *sensor_traces.csv* file consists of 256 samples per encryption, each sample represented as a hexadecimal 256-bit value—corresponding to the RDS output register width. The script also converts this to a binary *traces.bin* file, which calculates the Hamming weight of each sample and writes it as an `uint8` value. For completeness, the script also converts the keys to a text file of hexadecimal values. The t-tests were not in the scope of this thesis, hence they have been left intact.

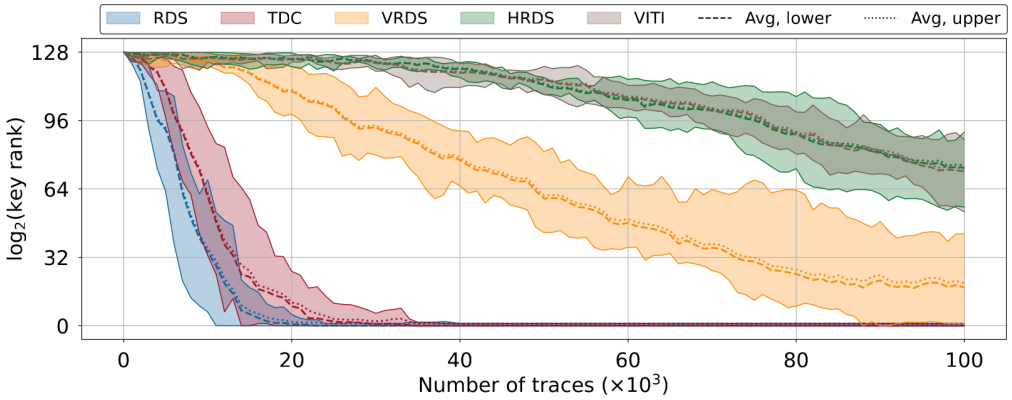
The script takes two arguments, a path to the RDS output directory (`/path/to/output_dir` mentioned in Section 4.4) and the number of encryptions. It can also be used to cut off an unfinished trace at its tail, by specifying a lower amount of encryptions than initially commanded (if the encryptions/measurements got interrupted for any reason).

4.6.2 Plotting script

To visualize the power traces there's the *plot.py* plotting script, which for debugging purposes also creates a *ham_weights.csv* file, that is simply the *sensor_traces.csv* file with the hexadecimal sensor register outputs represented in decimal Hamming weights. It takes the path to the *sensor_traces.csv* file and the number of power traces to plot as input. This script was also used for generating Figures 4.3 and 4.4.

4.7 Replicating the Attack

This section is dedicated to the final goal of this thesis. First, some expectations of attack metrics are laid out, and then my results are compared with the expectations.



■ **Figure 4.5** Key rank estimation of RDS, TDC [55], and VITI [22]—the shaded area represents the observed extremes (min, max) across all runs, whereas the dashed and dotted lines represent the upper and lower bounds of the key rank range averaged over all experiments [1]

4.7.1 Expectations

Spielmann et al. have recorded 100,000 power traces in five different runs, each time with a different key. They have used the key rank estimation as their side-channel evaluation metric. The key rank is calculated as a range and works in the following way: if an attacker has no side-channel information, then the key rank equals the entire key space, i.e., 2^{128} in the case of AES-128. Alternatively, the key rank drops to zero, when the entire key is broken. [1] Their results are illustrated in Figure 4.5—it would be expected that on average the full key would be broken with around 20,000 power traces.

4.7.2 First Attack Attempt

After the CPA script had been tested, the AES and sensor deployed on the board, and the host PC drivers ready, I recorded the first 40,000 power traces. Then I started the CPA attack using my script right away. The outcome is depicted as a terminal output, in Figure 4.6.

Immediately it is visible that something is not working correctly. With what should have been a very sufficient amount of traces, the attack did not work, with a guessing entropy of 74.5, even though a single byte of the key was found correctly (`0xB4`).

■ **Table 4.1** Attempts to reduce the noise of power consumption measurements by combining multiple measurements ran using the same encryption key and initial plaintext combination

Trace count	110,000	160,000	310,000	578,760
Guessing entropy	41	41.44	20.31	51.62

4.7.3 Merging and Higher Amount of Power Traces

Some ideas for a solution to the dysfunctional CPA were a) to record more measurements and observe how guessing entropy evolves and b) to merge power traces in order to separate the noise from the side-channel leakage. One of the running theories was, that the attack could be board dependent and my Basys 3 a manufacturing outlier.

```
Performing first round CPA using 40000 measurements.
Traces mtx shape: (40000, 256)
Byte guessing entropy: 32/256
key[0]: 0xFD, sample: 227
Byte guessing entropy: 42/256
key[1]: 0xD5, sample: 57
Byte guessing entropy: 250/256
key[2]: 0x24, sample: 66
Byte guessing entropy: 80/256
key[3]: 0x75, sample: 44
Byte guessing entropy: 107/256
key[4]: 0x4F, sample: 247
Byte guessing entropy: 93/256
key[5]: 0xF1, sample: 147
Byte guessing entropy: 0/256
key[6]: 0xB4, sample: 67
Byte guessing entropy: 15/256
key[7]: 0xF5, sample: 67
Byte guessing entropy: 254/256
key[8]: 0xF7, sample: 11
Byte guessing entropy: 64/256
key[9]: 0x96, sample: 251
Byte guessing entropy: 6/256
key[10]: 0x10, sample: 65
Byte guessing entropy: 19/256
key[11]: 0xDF, sample: 74
Byte guessing entropy: 17/256
key[12]: 0xC4, sample: 65
Byte guessing entropy: 58/256
key[13]: 0xB2, sample: 65
Byte guessing entropy: 63/256
key[14]: 0x09, sample: 194
Byte guessing entropy: 92/256
key[15]: 0xC8, sample: 66
CPA took: 51 seconds
Guessing entropy: 74.50
================================================================================
Found key: 0xFD 0xD5 0x24 0x75 0x4F 0xF1 0xB4 0xF5 0xF7 0x96 0x10 0xDF 0xC4 0xB2 0x09 0xC8
Attack success: False
```

Figure 4.6 First CPA attempt—an unsuccessful first round attack

Figure 4.7 Evolving guessing entropy using the first round attack

The results of the merging attempts are showcased in Table 4.1. As is visible, the results do not look promising as they do not represent a decreasing function. A more granular insight (although on a smaller sample size) into the evolving guessing entropy of a single continuous measurement of 150,000 power traces is shown in Figure 4.7. The results indicate that the first round approach would take hundreds of thousands if not millions of power traces, in order to break the full key, that is if the side-channel leakage at the first round would even be sufficient to

break the full key at all. Based on previous measurements, running the experiment for 1,000,000 traces would take about 130 hours. This result is highly unsatisfactory, as it has been expected that the upper bound of power traces required to break the full key would be approximately 24,000 (see Section 4.7.1).

Everything was tested from this point onwards. First, the experiments were recorded on a different Basys 3 as well—with little to no success. Second, more traces were recorded, though it now started getting quite time-consuming—150,000 traces took around 20 hours, and 310,000 traces took over 40 hours. Third, my CPA script was also double-checked, and I have compared its results to other CPA scripts used at the faculty, as well as one provided by my supervisor. All ended up similarly, with variations between them in only a few bits, though all found the wrong keys.

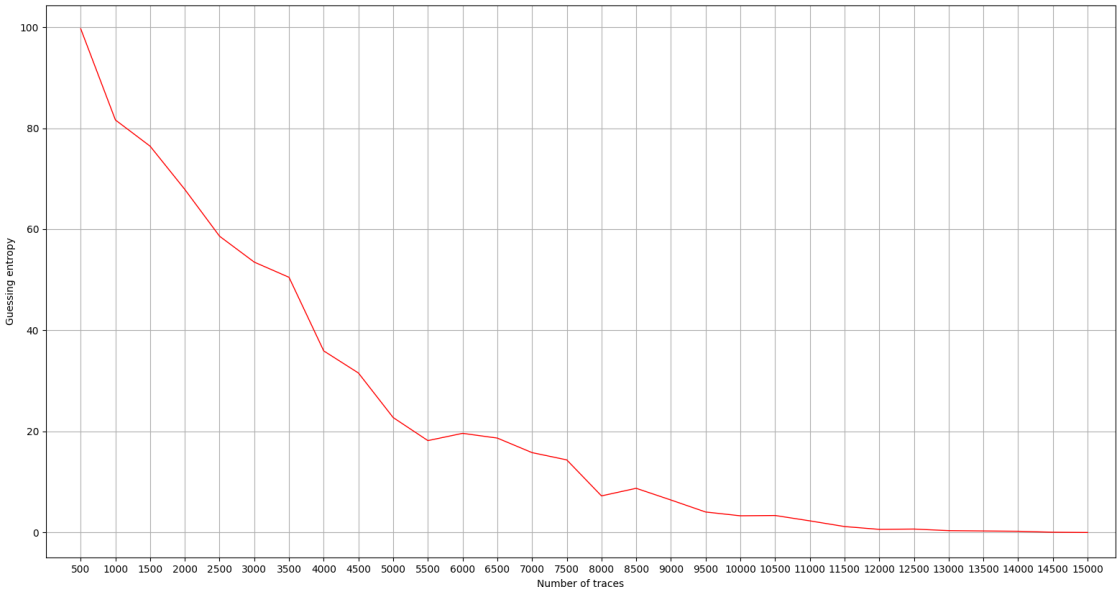
However, around this time, I found an issue with my CPA—the maximum correlation values were not looked at as absolute values, therefore high negative correlations, which could still yield a correct key guess, were not accounted for correctly. More testing has shown that the issue did not lie in this bug. After fixing it, the guessing entropy has been affected negatively.

4.7.4 Solving the CPA Issue

Having confirmed with my supervisor and MSc Spielmann that the traces looked nominal, the issue had to be on my side, in one of the scripts. On a single morning, after having looked through the original CUDA CPA code again, I realized, that instead of attacking the first round of AES, they are attacking the last round. My supervisor and MSc Spielmann suggested this on the same morning, so the issue seemed clear, and I have started implementing the last round attack.

```
Performing last round CPA using 40000 measurements.
Traces mtx shape: (40000, 256)
Byte guessing entropy: 0/256
key[0]: 0xE0, sample: 105
Byte guessing entropy: 0/256
key[1]: 0x7F, sample: 105
Byte guessing entropy: 0/256
key[2]: 0x16, sample: 105
Byte guessing entropy: 0/256
key[3]: 0xBD, sample: 105
Byte guessing entropy: 0/256
key[4]: 0xB9, sample: 106
Byte guessing entropy: 0/256
key[5]: 0xE5, sample: 105
Byte guessing entropy: 0/256
key[6]: 0x03, sample: 106
Byte guessing entropy: 0/256
key[7]: 0x46, sample: 105
Byte guessing entropy: 0/256
key[8]: 0xA2, sample: 105
Byte guessing entropy: 0/256
key[9]: 0x27, sample: 105
Byte guessing entropy: 0/256
key[10]: 0x7C, sample: 105
Byte guessing entropy: 0/256
key[11]: 0xD3, sample: 105
Byte guessing entropy: 0/256
key[12]: 0x82, sample: 105
Byte guessing entropy: 0/256
key[13]: 0x77, sample: 106
Byte guessing entropy: 0/256
key[14]: 0x42, sample: 106
Byte guessing entropy: 0/256
key[15]: 0x70, sample: 105
CPA took: 295 seconds
Guessing entropy: 0.00
================================================================================
Found key: 0x7D 0x26 0x6A 0xEC 0xB1 0x53 0xB4 0xD5 0xD6 0xB1 0x71 0xA5 0x81 0x36 0x60 0x5B
Attack success: True
```

■ **Figure 4.8** First successful attack using CPA to attack the last round of AES—notice the subkeys being different than the ones in the result, showcased as correct; this is because the subkeys are of the last round key, not the initial encryption key, bytes of which are highlighted at the bottom



■ **Figure 4.9** Plot of falling guessing entropy with incremented power traces—the full key is broken with 15,000 power traces

As per Figure 4.8, the last round CPA attack breaks the key successfully. Figure 4.9 illustrates the falling guessing entropy with the incremented power traces. The plot has been made from a single measurement, with the power traces progressively incremented by 5000 in each iteration. Breaking the key successfully while maintaining expected results, confirms the replicability of the RDS experiments.

4.8 Results

My experiments have confirmed the results of Spielmann et al. [1]. As is observable in Figure 4.9, the guessing entropy falls with the increased number of power traces, as expected. The guessing entropy of 0 (full key broken) is reached at around 15,000 power traces, which is in line with the expectations of Spielmann et al. [1] (see Figure 4.5). Thus the attack is well replicable. A thing worth mentioning is that from 11,500 power traces onwards, guessing entropy is less than 1.

The attack has been attempted at the first round of AES as well, with little success, having tried the CPA with up to 310,000 power traces. Further experiments would need to be carried out in order to find out whether breaking the full key would possible at all.

I have created the CPA script, which breaks the AES-128 key using the last round attack and it does so requiring around 15,000 power traces. As a side product, the helper scripts are able to convert the sensor outputs to a more readable format and plot the power traces while primarily being able to convert the sensor outputs to a format that can be used with the CPA script.

Chapter 5
Future Work

The RDS seems to have potential and therefore if it were used in the future, it would certainly benefit from a refactoring of the host PC driver side of the code, as it is a bit lacking in terms of readability, which consequently impacts its modularity. Additionally, it would be interesting to see how the RDS would perform in an attack on a soft-core processor, or even an SoC running a software crypto algorithm. Another interesting future research topic could be performing remote power analysis using the RDS on a different cipher, such as Serpent [56]. From a high abstraction level, Serpent features a block size of 128 bits, a 128-bit key, and a substitution box. The only notable difference seems to be that it has 32 rounds, instead of 10 as is the case for AES. All of this points to a possible remote power analysis with not so many necessary changes to the existing project.

With remote side-channel attacks came efforts for remote reverse-engineering efforts. As research shows, deep learning classifiers in combination with remote power consumption sensors have great potential to determine the executed instructions on a soft-core processor, with accuracy higher than 80% [31]. Side-channel attacks utilizing deep learning have been previously used for profiled attacks, however, they have been demonstrated to be able to assist in performing non-profiled side-channel attacks as well. [57] More research on these topics could further our understanding of the impact deep learning methods can have on the novel remote side-channel analysis methods.

Countermeasures for remote power analysis attacks are in development, such as reversing bitstreams into netlists that are analyzed for malicious design patterns [48], though these methods rely on an available bitstream reverse engineering toolchain, which few FPGA families have publicly accessible. While it certainly is not the only countermeasure available,
Glamočanin et al. have thoroughly categorized available countermeasures and concluded: "*Finding the right countermeasure—or a combination of them—remains an open problem.*" [58]

Chapter 6
Conclusion

The goals of becoming familiar with the AES cipher and correlation power analysis were met, as the correlation power analysis has been successfully implemented in Python and an attack on the first and last round of the AES cipher has been carried out. Furthermore, remote attacks have been researched, specifically remote power analysis side-channel attacks, which were thoroughly dissected and analyzed in Chapter 3.

Based on this analysis, an on-chip power consumption sensor was selected and deployed. The selected remote power analysis attack has been successfully conducted and evaluated using the previously tested CPA script. A comprehensive step-by-step procedure of steps taken has been documented, thus, proving its replicability, as the results do not differ from the original authors in any significant way. Additionally, several promising avenues for future research are suggested, which are discussed in more detail in Chapter 5.

In retrospect, it becomes evident that more efficient communication with the original authors of RDS could have sped up the replication of the attack. Nevertheless, the comprehensive documentation of the implemented attack procedures and the successful replication of results in alignment with the original authors satisfies the goals outlined at the beginning of this thesis.

Bibliography

1. SPIELMANN, David; GLAMOČANIN, Ognjen; STOJILOVIĆ, Mirjana. RDS: FPGA Routing Delay Sensors for Effective Remote Power Analysis Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*. 2023, vol. 2023, pp. 543–567. Available from DOI: 10.46586/tches.v2023.i2.543-567.
2. BARTÍK, Matěj. *FPGA Applications for Automotive* [online]. 2022-07. Available also from: https://pesw.fit.cvut.cz/2022/Digiteq.pdf. (Accessed on 05/08/2024).
3. EUROPEAN SPACE AGENCY, ESA. *The use of reprogrammable FPGAs in space* [online]. Available also from: https://www.esa.int/enabling_support/space_engineering_technology/microelectronics/the_use_of_reprogrammable_fpgas_in_space. (Accessed on 05/08/2024).
4. INTEL. *FPGA for Military Applications - Intel® FPGA* [online]. Available also from: https://www.intel.com/content/www/us/en/government/products/programmable/applications.html. (Accessed on 05/08/2024).
5. ADVANCED MICRO DEVICES, Inc. *Space-Grade Kintex UltraScale FPGA Family* [online]. 2024. Available also from: https://www.xilinx.com/products/silicon-devices/fpga/rt-kintex-ultrascale.html. (Accessed on 05/08/2024).
6. MICROCHIP TECHNOLOGY, Inc. *RT PolarFire® FPGAs | Microchip Technology* [online]. 2024. Available also from: https://www.microchip.com/en-us/products/fpgas-and-plds/radiation-tolerant-fpgas/rt-polarfire-fpgas#. (Accessed on 05/08/2024).
7. ADVANCED MICRO DEVICES, Inc. *Samsung SmartSSD* [online]. 2024. Available also from: https://www.xilinx.com/applications/data-center/computational-storage/smartssd.html. (Accessed on 05/07/2024).
8. AMAZON WEB SERVICES, Inc. *Amazon EC2 F1 Instances* [online]. 2024. Available also from: https://aws.amazon.com/ec2/instance-types/f1/. (Accessed on 05/07/2024).
9. MICROSOFT. *Azure virtual machine sizes for field-programmable gate arrays (FPGA) - Azure Virtual Machines | Microsoft Learn* [online]. 2023-03. Available also from: https://learn.microsoft.com/en-us/azure/virtual-machines/sizes-field-programmable-gate-arrays. (Accessed on 05/07/2024).
10. CAULFIELD, Adrian M. et al. Configurable Clouds. *IEEE Micro*. 2017, vol. 37, no. 3, pp. 52–61. Available from DOI: 10.1109/MM.2017.51.
11. OUYANG, Jian; LIN, Shiding; QI, Wei; WANG, Yong; YU, Bo; JIANG, Song. SDA: Software-defined accelerator for large-scale DNN systems. In: *2014 IEEE Hot Chips 26 Symposium (HCS)*. 2014, pp. 1–23. Available from DOI: 10.1109/HOTCHIPS.2014.7478821.

12. DIGILENT. *Basys 3 - Digilent Reference* [online]. Available also from: https://digilent.com/reference/programmable-logic/basys-3/start. (Accessed on 05/08/2024).

13. ADVANCED MICRO DEVICES, Inc. *AMD Technical Information Portal* [online]. 2016-09. Available also from: https://docs.amd.com/v/u/en-US/ug474_7Series_CLB. (Accessed on 05/08/2024).

14. BO, Song; KAWAKAMI, Kensuke; NAKANO, Koji; ITO, Yasuaki. An RSA encryption hardware algorithm using a single DSP block and a single block RAM on the FPGA. *IJNC*. 2011, vol. 1, pp. 277–289. Available from DOI: 10.1109/IC-NC.2010.56.

15. NIST. *Specification for the Advanced Encryption Standard (AES)* [Federal Information Processing Standards Publication 197]. 2001. Available also from: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

16. DAEMEN, Joan; RIJMEN, Vincent. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002. ISBN 3-540-42580-2.

17. AOKI LABORATORY GSIS, Tohoku University. *CAST-128 Hardware Macro Specification (Draft)* [online]. 2007. Available also from: http://www.aoki.ecei.tohoku.ac.jp/crypto/items/AESSpec2007Sep25.pdf. (Accessed on 05/08/2024).

18. KOCHER, Paul C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*. Springer, 1996, pp. 104–113.

19. PRINETTO, Paolo; ROASCIO, Gianluca, et al. Hardware Security, Vulnerabilities, and Attacks: A Comprehensive Taxonomy. In: *ITASEC*. 2020, pp. 177–189.

20. SOCHA, Petr; MIŠKOVSKÝ, Vojtěch; NOVOTNÝ, Martin. A Comprehensive Survey on the Non-Invasive Passive Side-Channel Analysis. *Sensors*. 2022, vol. 22, no. 21. ISSN 1424-8220. Available from DOI: 10.3390/s22218096.

21. SCHELLENBERG, Falk; GNAD, Dennis R.E.; MORADI, Amir; TAHOORI, Mehdi B. An inside job: Remote power analysis attacks on FPGAs. In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, pp. 1111–1116. Available from DOI: 10.23919/DATE.2018.8342177.

22. UDUGAMA, Brian; JAYASINGHE, Darshana; SAADAT, Hassaan; IGNJATOVIC, Aleksandar; PARAMESWARAN, Sri. VITI: A Tiny Self-Calibrating Sensor for Power-Variation Measurement in FPGAs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*. 2021, vol. 2022, no. 1, pp. 657–678. Available from DOI: 10.46586/tches.v2022.i1.657-678.

23. BRIER, Eric; CLAVIER, Christophe; OLIVIER, Francis. Correlation power analysis with a leakage model. In: *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*. Springer, 2004, pp. 16–29.

24. MEUNIER, Quentin L. FastCPA: Efficient correlation power analysis computation with a large number of traces. In: *Proceedings of the Sixth Workshop on Cryptography and Security in Computing Systems*. 2019, pp. 7–12.

25. MASSEY, J.L. Guessing and entropy. In: *Proceedings of 1994 IEEE International Symposium on Information Theory*. 1994, pp. 204–. Available from DOI: 10.1109/ISIT.1994.394764.

26. ZHAO, Mark; SUH, G. Edward. FPGA-Based Remote Power Side-Channel Attacks. In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 229–244. Available from DOI: 10.1109/SP.2018.00049.

27. ADVANCED MICRO DEVICES, Inc. *The MicroBlaze Soft Processor: Flexibility and Performance for Cost-Sensitive Embedded Designs (WP501) • Viewer • AMD Technical Information Portal* [online]. 2018-04. Available also from: https://docs.amd.com/v/u/en-US/wp501-microblaze. (Accessed on 02/05/2024).

28. INTEL. *Nios® V Processor for Intel® FPGA* [online]. Available also from: https://www.intel.com/content/www/us/en/products/details/fpga/nios-processor/v.html. (Accessed on 02/05/2024).

29. CONTRIBUTORS, PicoRV. *GitHub - YosysHQ/picorv32: PicoRV32 - A Size-Optimized RISC-V CPU* [online]. 2024-03. Available also from: https://github.com/YosysHQ/picorv32. (Accessed on 02/05/2024).

30. CONTRIBUTORS, ZipCPU. *GitHub - ZipCPU/zipcpu: A small, light weight, RISC CPU soft core* [online]. 2024-01. Available also from: https://github.com/ZipCPU/zipcpu. (Accessed on 02/05/2024).

31. GLAMOČANIN, Ognjen; SHRIVASTAVA, Shashwat; YAO, Jinwei; ARDO, Nour; PAYER, Mathias; STOJILOVIĆ, Mirjana. Instruction-Level Power Side-Channel Leakage Evaluation of Soft-Core CPUs on Shared FPGAs. *Journal of Hardware and Systems Security*. 2023, vol. 7, pp. 1–28. Available from DOI: 10.1007/s41635-023-00135-1.

32. TRIMBERGER, Steve; MCNEIL, Steve. Security of FPGAs in data centers. In: *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*. 2017, pp. 117–122. Available from DOI: 10.1109/IVSW.2017.8031556.

33. GNAD, Dennis R.E.; OBORIL, Fabian; KIAMEHR, Saman; TAHOORI, Mehdi B. Analysis of transient voltage fluctuations in FPGAs. In: *2016 International Conference on Field-Programmable Technology (FPT)*. 2016, pp. 12–19. Available from DOI: 10.1109/FPT.2016.7929182.

34. ÖRS, Sıddıka Berna; OSWALD, Elisabeth; PRENEEL, Bart. Power-Analysis Attacks on an FPGA – First Experimental Results. In: WALTER, Colin D.; KOÇ, Çetin K.; PAAR, Christof (eds.). *Cryptographic Hardware and Embedded Systems - CHES 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 35–50. ISBN 978-3-540-45238-6.

35. GRAVELLIER, Joseph; DUTERTRE, Jean-Max; TEGLIA, Yannick; MOUNDI, Philippe Loubet; OLIVIER, Francis. Remote Side-Channel Attacks on Heterogeneous SoC. In: BELAÏD, Sonia; GÜNEYSU, Tim (eds.). *Smart Card Research and Advanced Applications*. Cham: Springer International Publishing, 2020, pp. 109–125. ISBN 978-3-030-42068-0.

36. ADVANCED MICRO DEVICES, Inc. *CARRY4 • Vivado Design Suite 7 Series FPGA and Zynq 7000 SoC Libraries Guide (UG953) • Reader • AMD Technical Information Portal* [online]. 2023-10. Available also from: https://docs.amd.com/r/en-US/ug953-vivado-7series-libraries/CARRY4. (Accessed on 23/04/2024).

37. ADVANCED MICRO DEVICES, Inc. *LUT Primitives • Versal ACAP Configurable Logic Block Architecture Manual (AM005) • Reader • AMD Technical Information Portal* [online]. 2023-02. Available also from: https://docs.amd.com/r/en-US/am005-versal-clb/LUT-Primitives. (Accessed on 23/04/2024).

38. WU, Jinyuan. Several Key Issues on Implementing Delay Line Based TDCs Using FPGAs. *IEEE Transactions on Nuclear Science*. 2010, vol. 57, no. 3, pp. 1543–1548. Available from DOI: 10.1109/TNS.2010.2045901.

39. HENZLER, Stephan. Applications for Time-to-Digital Converters. In: *Time-to-Digital Converters*. Dordrecht: Springer Netherlands, 2010, pp. 103–113. ISBN 978-90-481-8628-0. Available from DOI: 10.1007/978-90-481-8628-0_6.

40. UENO, Miho; HASHIMOTO, Masanori; ONOYE, Takao. Real-time on-chip supply voltage sensor and its application to trace-based timing error localization. In: *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*. 2015, pp. 188–193. Available from DOI: 10.1109/IOLTS.2015.7229857.

41. ZICK, Kenneth M.; SRIVASTAV, Meeta; ZHANG, Wei; FRENCH, Matthew. Sensing nanosecond-scale voltage attacks and natural transients in FPGAs. In: *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. Monterey, California, USA: Association for Computing Machinery, 2013, pp. 101–104. FPGA '13. ISBN 9781450318877. Available from DOI: 10.1145/2435264.2435283.

42. ADVANCED MICRO DEVICES, Inc. *RLOC • Vivado Design Suite Properties Reference Guide (UG912) • Reader • AMD Technical Information Portal* [online]. 2023-11. Available also from: https://docs.amd.com/r/en-US/ug912-vivado-properties/RLOC. (Accessed on 24/04/2024).

43. ADVANCED MICRO DEVICES, Inc. *Using for-generate Statements • Vivado Design Suite User Guide: Synthesis (UG901) • Reader • AMD Technical Information Portal* [online]. 2023-11. Available also from: https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Using-for-generate-Statements. (Accessed on 24/04/2024).

44. SUGAWARA, Takeshi; SAKIYAMA, Kazuo; NASHIMOTO, Shoei; SUZUKI, Daisuke; NAGATSUKA, Tomoyuki. Oscillator without a combinatorial loop and its threat to FPGA in data centre. *Electronics Letters*. 2019, vol. 55, no. 11, pp. 640–642.

45. GRAVELLIER, Joseph; DUTERTRE, Jean-Max; TEGLIA, Yannick; LOUBET-MOUNDI, Philippe. High-Speed Ring Oscillator based Sensors for Remote Side-Channel Attacks on FPGAs. In: *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. 2019, pp. 1–8. Available from DOI: 10.1109/ReConFig48160.2019.8994789.

46. ZICK, Kenneth M.; HAYES, John P. Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems. *ACM Trans. Reconfigurable Technol. Syst.* 2012, vol. 5, no. 1. ISSN 1936-7406. Available from DOI: 10.1145/2133352.2133353.

47. FRANCO, John J. León; BOEMO, Eduardo; CASTILLO, Encarnación; PARRILLA, Luis. Ring oscillators as thermal sensors in FPGAs: Experiments in low voltage. In: *2010 VI Southern Programmable Logic Conference (SPL)*. 2010, pp. 133–137. Available from DOI: 10.1109/SPL.2010.5483027.

48. KRAUTTER, Jonas; GNAD, Dennis R. E.; TAHOORI, Mehdi B. Mitigating Electrical-level Attacks towards Secure Multi-Tenant FPGAs in the Cloud. *ACM Trans. Reconfigurable Technol. Syst.* 2019, vol. 12, no. 3. ISSN 1936-7406. Available from DOI: 10.1145/3328222.

49. AMAZON WEB SERVICES, Inc. *aws-fpga/ERRATA.md at master · aws/aws-fpga · GitHub* [online]. 2021-10. Available also from: https://github.com/aws/aws-fpga/blob/master/ERRATA.md. (Accessed on 05/06/2024).

50. AHMED, Ibrahim; SHEN, Linda L.; BETZ, Vaughn. Optimizing FPGA Logic Circuitry for Variable Voltage Supplies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2020, vol. 28, no. 4, pp. 890–903. Available from DOI: 10.1109/TVLSI.2019.2962501.

51. FAROOQ, Umer; MARRAKCHI, Zied; MEHREZ, Habib. FPGA Architectures: An Overview. In: *Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*. New York, NY: Springer New York, 2012, pp. 7–48. ISBN 978-1-4614-3594-5. Available from DOI: 10.1007/978-1-4614-3594-5_2.

52. SPIELMANN, David; GLAMOČANIN, Ognjen; STOJILOVIĆ, Mirjana. *GitHub - mirjanastojilovic/RDS: FPGA routing delay sensors for effective remote power analysis attacks* [online]. 2024-03. Available also from: https://github.com/mirjanastojilovic/RDS. (Accessed on 05/13/2024).

53. CONTRIBUTORS, NumPy. *NumPy documentation — NumPy v1.26 Manual* [online]. 2022. Available also from: `https://numpy.org/doc/stable/`. (Accessed on 05/11/2024).

54. DIGILENT. *Installing Vivado, Xilinx SDK, and Digilent Board Files - Digilent Reference* [online]. [N.d.]. Available also from: `https://digilent.com/reference/programmable-logic/guides/installing-vivado-and-sdk`. (Accessed on 05/13/2024).

55. GNAD, Dennis RE; NGUYEN, Cong Dang Khoa; GILLANI, Syed Hashim; TAHOORI, Mehdi B. Voltage-based covert channels using FPGAs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*. 2021, vol. 26, no. 6, pp. 1–25.

56. BIHAM, Eli; ANDERSON, Ross; KNUDSEN, Lars. Serpent: A new block cipher proposal. In: *International workshop on fast software encryption*. Springer, 1998, pp. 222–238.

57. TIMON, Benjamin. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*. 2019, pp. 107–131.

58. GLAMOČANIN, Ognjen; MAHMOUD, Dina G.; REGAZZONI, Francesco; STOJILOVIĆ, Mirjana. Shared FPGAs and the Holy Grail: Protections against Side-Channel and Fault Attacks. In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2021, pp. 1645–1650. Available from DOI: `10.23919/DATE51398.2021.9473947`.

Attachment contents

- `rds` ..... RDS project contents used
  - `measurements` ..... measured power traces
  - `src`
    - `impl` ..... implementation source code
    - `thesis` ..... thesis LaTeX source code
  - `text`
    - `thesis.pdf` ..... thesis in the PDF format