Analysis and Mitigation of Remote Side-Channel and Fault Attacks on the Electrical Level

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Jonas Krautter

Tag der mündlichen Prüfung:	17.02.2022
1. Referent:	Prof. Dr. Mehdi B. Tahoori, Karlsruher Institut für Technologie (KIT)
2. Referent:	Prof. Dr. Thomas Eisenbarth, Universität zu Lübeck

Acknowledgements

Many people have supported me during my time as a PhD student, and I would like to thank a few of them specifically here.

First and foremost, I would like to thank my first advisor, Prof. Mehdi Tahoori. Without the countless helpful talks and discussions we had during my time at the Chair of Dependable Nano Computing, the research presented here would not have been possible. He has contributed a great deal to my academic development, and even casual, informal talk we had at conferences or lunch could lead to new ideas anytime. I want to thank my second advisor, Prof. Thomas Eisenbarth, from the university of Lübeck for accompanying me on this journey as well. We had many meetings online and during conferences, discussing new threat models and attack scenarios, which has been very helpful for achieving many of the results presented here. From my colleagues, I want to specifically thank Dennis Gnad, who had already been my advisor during my master's thesis and has afterwards convinced me to start my PhD. He has coauthored almost all of my publications, and we had many stressful last-minute submissions, but also great conference visits and student labs together. I should also thank Falk Schellenberg and Amir Moradi from the Ruhr-University Bochum. With both we authored many publications and received helpful advice to equip our very own hardware security lab. Overall, I had a very pleasant time at the chair during my research as a PhD student and I want to thank all my colleagues and also our secretary, Iris Schröder-Piepka, who was extremely helpful from organizing student matters to travel reservations.

Aside from colleagues, coauthors and acquaintances on the academic side, I want to thank my parents and all of my family, who believed in me and supported me. They always had a place for me to relax from the latest deadlines. Moreover, without enlisting all their names explicitly I thank all my friends, who have contributed a great deal by keeping my sanity high in these times of isolation during the pandemic.

Jonas Krautter
Jägerhausstr. 16
76139 Karlsruhe

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Karlsruhe, 11. April 2022
Jonas Krautter

Abstract

The ongoing miniaturization of integrated circuits is approaching physical limitations, where atom-scale transistors are setting a lower limit on feature sizes. Moreover, the complexity of the manufacturing process for bleeding-edge chips has raised the cost for a full chip design to a level, where only the largest multinational competitors can afford to bear the financial risk. Due to this development, miniaturization is no longer the primary driver of further increasing the performance of electronic components. Instead, classical computer architectures with general purpose processing units and standard bus structures are extended towards heterogeneous computing systems with highly parallel generic and specific accelerators.

However, the new heterogeneous systems introduce new challenges for protecting the integrity and confidentiality of private data against attackers. These security challenges arise from a number of factors, such as new hardware components, novel kinds of applications and the overall increase in system complexity. Cryptographic algorithms have been secured against cryptanalysis attacks very effectively, but even provably secure algorithms are only secure under certain assumptions about the attacker. A common concept, for instance, is the black-box model, where it is assumed that the attacker has only access to the encrypted outputs and cannot access intermediate or internal signals. In real implementations, however, side-channel and fault attacks invalidate such assumptions and continue to threaten implementations that are otherwise secure against cryptanalysis. Side-channel attacks are based on measuring data-dependent observables, such as power consumption or electromagnetic emanation, whereas fault attacks actively disturb implementations to derive secret data from faulty outputs.

These implementation attacks were traditionally only considered in the context of a local, physical attacker with access to the target device. However, attacks based on memory access time side-channels escalated the threat to that of a remote attacker, who is simply able to measure execution time with sufficient accuracy. In this thesis, we consider the threat of remote side-channel and fault attacks on the electrical level, which is very closely related to the recent development towards heterogeneous systems. Field Programmable Gate Arrays (FPGAs), which are reprogrammable logic chips that can realize almost arbitrary circuits, are an example of emerging hardware in heterogeneous computing and have been introduced in the edge and the cloud as generic accelerators. Their flexibility, however, allows attackers to implement sensors to estimate voltage fluctuations or synchronously toggle massive amounts of logic elements to inject faults into other computations. We analyze this threat by extending existing attacks and develop approaches to mitigate them.

Zusammenfassung

In der fortlaufenden Miniaturisierung von integrierten Schaltungen werden physikalische Grenzen erreicht, wobei beispielsweise Einzelatomtransistoren eine mögliche untere Grenze für Strukturgrößen darstellen. Zudem ist die Herstellung der neuesten Generationen von Mikrochips heutzutage finanziell nur noch von großen, multinationalen Unternehmen zu stemmen. Aufgrund dieser Entwicklung ist Miniaturisierung nicht länger die treibende Kraft um die Leistung von elektronischen Komponenten weiter zu erhöhen. Stattdessen werden klassische Computerarchitekturen mit generischen Prozessoren weiterentwickelt zu heterogenen Systemen mit hoher Parallelität und speziellen Beschleunigern.

Allerdings wird in diesen heterogenen Systemen auch der Schutz von privaten Daten gegen Angreifer zunehmend schwieriger. Neue Arten von Hardware-Komponenten, neue Arten von Anwendungen und eine allgemein erhöhte Komplexität sind einige der Faktoren, die die Sicherheit in solchen Systemen zur Herausforderung machen. Kryptografische Algorithmen sind oftmals nur unter bestimmten Annahmen über den Angreifer wirklich sicher. Es wird zum Beispiel oft angenommen, dass der Angreifer nur auf Eingaben und Ausgaben eines Moduls zugreifen kann, während interne Signale und Zwischenwerte verborgen sind. In echten Implementierungen zeigen jedoch Angriffe über Seitenkanäle und Faults die Grenzen dieses sogenannten Black-Box-Modells auf. Während bei Seitenkanalangriffen der Angreifer datenabhängige Messgrößen wie Stromverbrauch oder elektromagnetische Strahlung ausnutzt, wird bei Fault Angriffen aktiv in die Berechnungen eingegriffen, und die falschen Ausgabewerte zum Finden der geheimen Daten verwendet.

Diese Art von Angriffen auf Implementierungen wurde ursprünglich nur im Kontext eines lokalen Angreifers mit Zugriff auf das Zielgerät behandelt. Jedoch haben bereits Angriffe, die auf der Messung der Zeit für bestimmte Speicherzugriffe basieren, gezeigt, dass die Bedrohung auch durch Angreifer mit Fernzugriff besteht. In dieser Arbeit wird die Bedrohung durch Seitenkanal- und Fault-Angriffe über Fernzugriff behandelt, welche eng mit der Entwicklung zu mehr heterogenen Systemen verknüpft sind. Ein Beispiel für neuartige Hardware im heterogenen Rechnen sind Field-Programmable Gate Arrays (FPGAs), mit welchen sich fast beliebige Schaltungen in programmierbarer Logik realisieren lassen. Diese Logik-Chips werden bereits jetzt als Beschleuniger sowohl in der Cloud als auch in Endgeräten eingesetzt. Allerdings wurde gezeigt, wie die Flexibilität dieser Beschleuniger zur Implementierung von Sensoren zur Abschätzung der Versorgungsspannung ausgenutzt werden kann. Zudem können durch eine spezielle Art der Aktivierung von großen Mengen an Logik Berechnungen in anderen Schaltungen für Fault Angriffe gestört werden. Diese Bedrohung wird hier beispielsweise durch die Erweiterung bestehender Angriffe weiter analysiert und es werden Strategien zur Absicherung dagegen entwickelt.

Contents

List of own publications included in this thesis

Transactions & Articles

[22] J. Krautter, D. R. E. Gnad, M. B. Tahoori, "Mitigating Electrical-Level Attacks towards Secure Multi-Tenant FPGAs in the Cloud", in ACM Transactions on Reconfigurable Technology and Systems (TRETS), 2019.

[21] J. Krautter, D. R. E. Gnad, M. B. Tahoori, "CPAmap: On the Complexity of Secure FPGA Virtualization, Multi-Tenancy, and Physical Design", in IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), 2020.

Conferences

[25] J. Krautter, D. R. E. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Active Fences against Voltage-based Side Channels in Multi-Tenant FPGAs", in Proceedings of the International Conference on Computer-Aided Design (ICCAD), 2019, USA.

[20] J. Krautter, D. R. E. Gnad, M. B. Tahoori, "Remote and Stealthy Fault Attacks on Virtualized FPGAs", in Proceedings of Design, Automation & Test in Europe (DATE), 2021.

[26] J. Krautter, M. Tahoori, "Neural Networks as a Side-Channel Countermeasure: Challenges and Opportunities", in Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2021.

List of own publications not included in this thesis

[13] J. Krautter, D. R. E. Gnad, M. B. Tahoori, "FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES", in IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), 2018.

[23] D. R. E. Gnad, S. Rapp, J. Krautter, M. B. Tahoori, "Checking for Electrical Level Security Threats in Bitstreams for Multi-Tenant FPGAs", International Conference on Field-Programmable Technology (FPT), 2018, Japan.

[17] D. R. E. Gnad, J. Krautter, M. B. Tahoori, "Leaky Noise: New Side-Channel Attack Vectors in Mixed-Signal IoT Devices", in IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES), 2019.

[27] D. R. E. Gnad, F. Schellenberg, J. Krautter, A. Moradi and M. B. Tahoori, "Remote Electrical-level Security Threats to Multi-Tenant FPGAs", in IEEE Design & Test, 2020.

[28] S. Meschkov, D. R. E. Gnad, J. Krautter, M. B. Tahoori, "Is your secure test infrastructure secure enough?", in IEEE International Test Conference (ITC), 2021.

[29] J. Krautter, M. Mayahinia, D. R. E. Gnad, M. B. Tahoori , "Data Leakage through Self-Terminated Write Schemes in Memristive Caches", in Asia and South Pacific Design Automation Conference (ASP-DAC), 2022.

Part I.

Preliminaries

1. Introduction

In the age of information, the thirst for more computational power is ever-increasing and drives the effort for performance increase and higher energy efficiency in both hardware and software. A main contributing factor for further improvements to performance and efficiency of Integrated Circuits (ICs) has been the higher transistor density by shrinking structures down to just a few nanometres in size. However, as the first functional atom-scale transistors [1] have emerged and herald an end of further miniaturization, chip designers and system integrators instead veer away from traditional computer architectures and general purpose Central Processing Units (CPUs). Instead, the hardware is adapted to the application, where specific accelerators enable even further performance and energy benefits. Graphics Processing Units (GPUs) have been used for general purpose computing since many years now to achieve massive parallelization of low-level mathematics, for instance, in Artificial Intelligence (AI) algorithms.

Nowadays, especially programmable logic such as Field Programmable Gate Arrays (FPGAs) is added to edge [2, 3] as well as cloud systems [4–6]. These generic accelerators provide an array of Look-Up Tables (LUTs), registers and interconnect resources as well as various other interfaces or Digital Signal Processing (DSP) cores, which can be reprogrammed at runtime to realize almost arbitrary circuits. Compared to GPUs, FPGAs offer a higher flexibility and – especially with regard to AI applications – lower latency, for instance, in speech processing or image recognition. With the increasing amount of resources available per FPGA chip, virtualization and multi-tenancy of FPGAs is a major concern for improving utilization, especially in the cloud [7].

In those emerging heterogeneous systems, hardware resources are shared among multiple untrusted users and users may utilize software or hardware Intellectual Property (IP) from untrusted third-party providers. The increased complexity and novel hardware components make security very challenging and proper isolation between different users is crucial for ensuring confidentiality and integrity of user data. Although cryptographic algorithms, which are meant to assure secure computation and communication, have been well researched and are nowadays resilient against theoretical cryptanalysis attacks, their implementations are often vulnerable. This discrepancy is due to the assumptions about potential attackers that are made when designing cryptographic algorithms. Often a black-box model is applied, where the attacker is able to access only the inputs and outputs of a cryptographic algorithm, but no intermediate values. Actual unprotected implementations, however, are vulnerable to *side-channel* [8] and *fault attacks* [9], which invalidate the assumptions about the attacker.

To perform a side-channel or fault attack, an attacker needs to measure observables that depend on secret data or manipulate system properties to cause faulty computations. An example of such an observable is the power consumption, which in Complementary Metal Oxide Semiconductor (CMOS) technology usually depends on the

amount of bits that are flipped during an operation. Likewise, the supply voltage can be lowered until the circuit malfunctions, allowing an attacker to infer secret data from the incorrect outputs. In cache timing attacks, an attacker leverages the timing differences between cached and uncached memory access to learn about data-dependent reads and writes performed by the victim. These cache timing attacks can be deployed even by a remote attacker, for instance, in a cloud server to attack other Virtual Machines (VMs) or break out of a virtualized environment. On the other hand, implementation attacks based on power consumption or voltage manipulation have been mostly carried out by a local, physical attacker, who can access the device with measurement equipment.

Along with other researchers, we recently escalated the threat of side-channel and fault attacks on the electrical level to that of a remote attacker [10–16]. These remote power analysis and fault attacks are enabled due to the flexibility of novel hardware – specifically FPGAs – in the described heterogeneous systems, which allows attackers to exploit the given resources to implement sensors or power viruses. Compared to local attackers measuring the power consumption with access to the device, such remote side-channel and fault attacks are a much bigger threat, as they scale very well and can potentially be deployed to millions of devices, requiring no expensive equipment at all. In this thesis, we extend and analyze the discovered attack vectors and develop countermeasures against a remote attacker on the electrical level. Although we consider FPGAs in most of our works, the threat is not exclusive to these devices and can, for instance, affect microcontrollers in the Internet of Things (IoT) as well [17, 18].

1.1. Contributions

The main focus of this thesis is the exploration and mitigation of novel remote power side-channel and fault attacks that have recently been enabled through emerging heterogeneous multi-user platforms, specifically on FPGAs in the cloud. On one hand, we explore the existing threats in terms of finding new attack vectors and methodologies, primarily with the goal to understand the mechanisms behind the attacks to minimize the complexity and the overhead of countermeasures. We extend our initial work *FPGAhammer* [13], which demonstrated the possibility of remote fault attacks on the Advanced Encryption Standard (AES) through a large amount of Ring Oscillators (ROs) on an Intel Cyclone V FPGA. In Section 3.1, we prove that sufficient voltage drops can be caused with a large amount of seemingly benign logic, such as benchmark circuits, instead of ROs, which are easily prohibited by design checking approaches. Moreover, we demonstrate the attack on actual cloud hardware such as the Intel Stratix 10 FPGA. We also demonstrate in Section 3.2, how side-channel leakage can occur even through the power supply of a standard PC system, which we demonstrate by detecting specific code patterns through measurements from an FPGA-based PCIe accelerator card.

On the other hand, the new hardware requires the development of countermeasures, which are either specifically tailored to the new architectures or even leverage their properties just like attackers in this scenario. We develop an elaborate workflow to check FPGA-bitstreams for malicious signatures in Chapter 5, which is based on reversing the bitstream into a flattened netlist and applying various detection algorithms.

In Chapter 4, we thoroughly analyze the impact of physical design parameters, such as placement and routing, but also process variation on the side-channel vulnerability against remote side-channel attacks. Our findings show that the amount of measurements for a successful attack differs significantly depending on placement of victim and attacker, such that the differences extend the effects of simple hiding countermeasures. We also present a generic approach for dynamically hiding voltage fluctuations in Section 6.1, where on-chip sensors activate a fence of ROs in a way to compensate for the fluctuations caused by an AES encryption module. In Section 6.2, we show how emerging hardware can also be leveraged for developing unique new countermeasures by evaluating the usage of a neural network mapping of the AES substitution function against side-channel leakage.

In the following subsections, we detail the contributions and refer to the associated publications that we have published.

1.1.1. Exploring the Extent of Remote Electrical Fault and Side-Channel Attacks

To facilitate the development of countermeasures against remote fault and side-channel attacks, it is important to understand the extent of the issue, thus exploring the attacks from an attacker perspective first. In the initial works [12, 13], the attacker makes use of long delay lines – so called Time-to-Digital Converters (TDCs) – for measuring voltage variations through transistor delay and employs a large amount of ROs to induce critical voltage drops for fault injection. Both works demonstrate attacks between designs on a single multi-tenant FPGA where the designs are logically separated, i.e. no logical connections exist between the designs. Many researchers have since then explored how far up in the power supply hierarchy the shared power supply between untrusted entities can still raise security concerns. For instance, side-channel attacks have been demonstrated across different chips on the same Printed Circuit Board (PCB) in [12]. In [19], electric covert-channels between different components connected to the same ATX power supply in a standard PC system have been demonstrated successfully.

We evaluate leakage between CPU and other components in similar constellations, through sensors on an FPGA either on a System-on-Chip (SoC), where both components share the chip-level power supply or inside a standard PC system, where the FPGA is connected via the PCI Express bus. The results are presented in Section 3.2, showing it is possible to successfully distinguish different code patterns running on the CPU through the FPGA-based sensors.

In another line of research, we explore how the ROs which have been used in [13] can be replaced by other seemingly benign logic that is much harder to detect by potential design checks performed by a device hypervisor. Moreover, we extend our experiments to significantly larger FPGA devices, such as the Intel Stratix 10 FPGA, which would actually be employed in cloud computing. In Section 3.1, we demonstrate fault injections and key recovery attacks on an AES encryption module using benchmark circuits or even AES modules themselves to cause the required voltage drop. To employ countermeasures such as the bitstream checking approach which is also part of this thesis, elaborate signatures must be developed to detect malicious intent in a design. This

contribution has also been published in [20] together with Dennis Gnad and Mehdi Tahoori.

1.1.2. Evaluating the Impact of Design Parameters on Side-Channel Vulnerability

When evaluating remote, internal side-channel attacks on various FPGAs from different vendors, we often found large differences in the amount of measurements that have to be collected by an attacker, for instance, to recover a secret encryption key. This peculiarity is further explored in Chapter 4, where we investigate the general impact of physical design parameters such as placement and routing of both attacker and victim design on the overall vulnerability. We investigate an AES encryption module as the victim design and evaluate four design dimensions, namely four global module placement locations, the local placement of primitives through four different heuristic place and route strategies and four different FPGA chips. In total, we collect up to $100k$ measurement traces for 256 possible parameter combinations and also repeat experiments on large scale cloud FPGAs as well as designs protected by simple hiding countermeasures.

Our results show that the parameters can have a larger impact on the design vulnerability than simple hiding through noise generators. On one hand, our findings can be utilized for additional side-channel mitigation at zero overhead, only by specific placement and routing of attacker and/or victim design. On the other hand, an important conclusion is the importance of those parameters when implementing side-channel countermeasures, which can turn out completely ineffective simply because of highly vulnerable placement and routing on a specific FPGA. This contribution has been published in [21] together with Dennis Gnad and Mehdi Tahoori.

1.1.3. Bitstream-Checking as a Countermeasure

We develop a potential approach to prevent the described side-channel and fault attacks at least on FPGA-based hardware, where we propose a workflow for a cloud hypervisor to check FPGA designs for malicious signatures. The details of this mitigation strategy are presented in Chapter 5.

To detect designs that could potentially be used for fault injection or side-channel measurements remotely, the hypervisor first reverts the bitstream back into a flattened netlist of low level primitives. Then, the netlist can be evaluated through various perspectives, for instance, by treating it as a netlist graph, scanning for specific connections that are unusual in benign designs or performing a simple timing analysis. Whereas timing violations can indicate a sensor in the design, fault injections usually require a large amount of logic being synchronously toggled, which can be identified by searching for nodes with high fanout in the netlist graph. To prove that the proposed approach is not too restrictive, we verify the detection mechanisms on more than 40 benign benchmark designs from different collections, where none are flagged as potential attacker designs. An obvious limitation is the constant need for updated signatures, to counteract on new methods employed by attackers to hide sensors or fault injection logic.

The work has been published in [22] together with Dennis Gnad and Mehdi Tahoori, with some preliminary results presented in [23]. Since then, it has inspired similar approaches suggested by other research groups as well [24].

1.1.4. Novel Hiding Approaches against Side-Channel Attacks

Although emerging technologies open up new vulnerabilities for attackers, they also offer new opportunities for defending against attacks. In this line of work, we present novel approaches for side-channel attack mitigation on FPGAs, enabled by the flexibility given by emerging technologies.

Hiding techniques aim to worsen the Signal to Noise Ratio (SNR) for an attacker, greatly increasing the amount of measurements required for a successful attack by either increasing the noise level or preventing data-dependent voltage fluctuations that are caused by the victim implementation. On-chip sensors made from FPGA primitives can not only be used to attack a design but also for dynamically adapting power consumption and compensate the fluctuations caused by a vulnerable module. We show the effectiveness of this approach in Section 6.1, where we surround an AES encryption module with an *Active Fence*, which is composed of a sensor and a certain amount of ROs. The sensor measures the voltage fluctuations caused by the AES module and can enable or disable ROs to compensate for them, increasing the amount of required measurements by two to three orders of magnitude. This work has also been published in [25] together with Dennis Gnad, Falk Schellenberg, Amir Moradi and Mehdi Tahoori.

Another novel approach to side-channel mitigation is presented in Section 6.2, where we employ neural networks on an FPGA to replicate certain primitives within the AES encryption algorithm that are usually targeted by an attacker. Varying different parameters of the neural network, such as the activation function, the quantization, or the input and output data representation, we find that the choice of the activation function has a big impact on the design's side-channel vulnerability. In general, the methodology of using a neural network to map well-defined functions offers an interesting starting point for future works, where further parameters during training or inference can be optimized for implementation security. Neural networks are heavily intertwined with the topic of heterogeneous computing and emerging technologies, as neural accelerators are becoming a de-facto standard in computing systems nowadays. This work has been published in [26] together with Mehdi Tahoori.

1.2. Outline

The thesis is organized into the following chapters:

- In Chapter 2, we explain the theoretical background of the investigated attacks and mitigations. Besides the general theoretical concepts required for side-channel and fault attacks, we outline the mechanisms on the electrical level that are exploited by an attacker.

- Chapter 3 presents how we extended and explored existing fault and side-channel attacks, finding new methods to cause critical voltage fluctuations and analyzing leakage occurring through the power supply hierarchy.
- In Chapter 4, we analyze the dependency of side-channel vulnerability on various physical design parameters, such as attacker and victim placement on the chip.
- A mitigation methodology for both fault and side-channel attacks on multi-tenant FPGAs based on offline bitstream analysis is detailed in Chapter 5.
- We propose novel hiding methods against side-channel attacks, which leverage the properties of emerging technologies in Chapter 6.
- In Chapter 7, a comprehensive overview on related works as well as future research perspectives is provided.
- Finally, in Chapter 8 we draw some conclusions from the research that has been done in this thesis.

2. Background

In this chapter, we provide the necessary background to understand the problem of remote fault and side-channel attack on the electrical level. First, we explain the basic principles of side-channel and fault attacks on a theoretical level – mostly on the AES as the common proof-of-concept target in all of our works – before introducing the concept of remote attacks specifically. Therefore, we also briefly introduce the basic mechanisms of Power Distribution Networks (PDNs) and how power and voltage are connected to side-channel and fault attacks.

2.1. Fault and Side-Channel Attacks

Encryption algorithms have been mostly developed with mathematical security in mind, which usually refers to an attacker being unable to recover the plaintext input from the encrypted output without knowing the cryptographic key. This notion of security has also been formalized, for instance, with various definitions about ciphertext indistinguishability [30], where a computationally limited attacker is unable to distinguish two given ciphertexts correctly with a probability of more than 0.5. However, even provably secure cryptographic algorithms are often developed under the assumption of a black-box model, where the attacker can only access the output ciphertext and no intermediate values during computation. For actual hardware and software implementations of the algorithm, this assumption usually does not hold true: Through side-channel and fault attacks, attackers are able to recover secret encryption keys in seconds, even if there is no known mathematical weakness of the algorithm.

Side-channel attacks have been first introduced in 1999 [8] and are based on the measurement of variables and parameters which depend on the data that is processed during the computation of the algorithm. Such variables include, for instance, the device power consumption, supply voltage, electromagnetic emanation, temperature, noise, photon emission and many more. By measuring those variables, an attacker can derive information about internal signals and intermediate values of the computation, invalidating the black-box model and thus the algorithm's security. The data-dependency which leads to side-channel leakage into the described domains can be a consequence of data-dependent control flow, where different computations are performed depending on secret values. However, even when the basic control flow of the circuit or the software implementation does not depend on secret data directly, the data-dependency of side-channel measurements is almost impossible to mitigate completely.

Traditionally, side-channel attacks are performed by an attacker with physical access to the target device, where measurements can be performed with dedicated equipment, such as oscilloscopes and probes. A specific class of side-channel attacks is based on measuring execution time of the target to derive secret values. These timing attacks

can be employed on algorithms with a data-dependent control flow but also through cache timing attacks [31], where for instance the cache is reset to a specific state and the attacker can determine whether a specific data-dependent cache line was loaded after executing the victim code. Unlike most other side-channel attacks, timing attacks have been successfully deployed without additional external equipment, through cycle accurate counters and other internal measurements. This makes them especially dangerous, as they can be performed remotely to millions of devices at the same time.

However, with the rise of heterogeneous computing and increasing system complexity, researchers have been able to perform remote side-channel attacks on the electrical level, where the flexibility of the system is leveraged to estimate the supply voltage indirectly [10–12, 14]. Whereas most works consider FPGA-based sensors, the problem persists in any constellation where an untrusted user can access analog measurement data which is correlated to the system's power consumption [17, 32]. Since the extension of power-based side-channel attacks to a remote attacker without dedicated measurement equipment, their mitigation is an urgent issue due to the same reasons that we explained above for the timing-based side-channel attacks.

On the other hand, fault attacks [9] aim to actively disturb the computations of an algorithm in order to leverage the incorrect output values to derive secret data values. Faults can be injected through various measures, many of which are based on causing a timing fault to occur, where at least one register in the circuit latches an incorrect value at the end of a combinational logic path due to shortening the clock cycle or increasing the transistor delay by lowering the supply voltage. However, through more advanced methods – and with increasingly expensive equipment – faults can also be provoked through electromagnetic pulses, lasers or focussed ion beams. The goals of fault attackers can be different as well: In some situations, they might want to skip a security validation, for instance, by skipping a specific instruction, whereas otherwise the target may be the secret key of an encryption algorithm, which can be recovered by disturbing computations at a specific point in the algorithm. Last but not least, fault attacks can be also a safety concern, when faults, for instance, in edge devices in transportation or critical infrastructure can have catastrophic consequences.

Similar to side-channels attacks, faults have previously been injected by local attackers with access to the victim device, but for timing faults through the electrical level the threat has recently been escalated to that of a remote attacker, initially in FPGA-based setups [10, 13, 15, 16]. However, even in modern ARM and x86-based CPU-only systems, Dynamic Voltage and Frequency Scaling (DVFS) can be misused to break security mechanisms [33–36], by intentionally manipulating voltage or clock frequency to cause timing faults. These novel threats enable fault attacks on a large scale, making them just as important of an issue as remote side-channel attacks on the electrical level.

The collection of measurements or injection of faults is only one part of successfully deploying a side-channel or fault attack. After probing the device, the measurements or faulty outputs need to be analyzed for actually recovering the secret data, usually the secret key of a cryptographic algorithm. For instance, in some cases, a visual analysis of the measurement traces can be sufficient to recognize secret data directly, in a so-called Simple Power Analysis (SPA) attack. In the following sections, we explain specific side-channel and fault attacks on the AES algorithm, which is the currently

most widespread symmetric block cipher, in more detail. These attacks will often serve as a proof-of-concept for specific vulnerabilities as well as a way to analyze the extent of leakage.

2.2. The Advanced Encryption Standard (AES)

The AES [37] is a round based block cipher, that operates on blocks of 16 bytes (128 bits) of data and allows key lengths of 128, 192 and 256 bits. It is the most widespread symmetric encryption algorithm nowadays and is implemented with dedicated accelerators in almost every modern device that has any security requirements. We only perform experiments on implementations with a 128-bit key length, where for each of the 10 encryption rounds except the last one, four operations are performed on the data block, which is referred to as the *state*. The four operations are *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey*.

In the first operation *SubBytes*, the 16 bytes of the *state* each undergo a byte-wise transformation, which mathematically is a non-linear transformation in the Galois field $GF(2^8)$. It is the only non-linear operation in the algorithm and often simply implemented as a lookup-table with 256 entries. The byte substitution, which is also called S-Box is often a target for side-channel attacks.

For the *ShiftRows* operation, the 16 byte state is interpreted as a 4×4 matrix and each row is circularly and byte-wise shifted by a different offset each. This 4×4 matrix interpretation is also used in the *MixColumn* operation, where the entire matrix is multiplied column by column with a fixed predefined matrix over $GF(2^8)$.

Finally, the initial 16 byte key is extended to 176 bytes in total, from which in each round 16 bytes are added (XOR) to the state in the *AddRoundKey* operation. The key expansion is reversible, which is important for attacks where only a single round key can be recovered, from which the original key can then be derived.

2.3. Correlation Power Analysis on AES

Correlation Power Analysis (CPA) [38] is a powerful method for the analysis of side-channel measurements in order to find an unknown fixed secret value which has been used in the computations during the measurements. It is based on computing the Pearson correlation coefficient between a power model that depends on a guess on the targeted secret value and the actual measurements. Here, we describe the method in detail for attacks on the AES block cipher.

First, we introduce the Pearson correlation coefficient ρ_{xy} of two time series x and y of length n:

$$\rho_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.1}$$

This coefficient ρ_{xy} can assume values from -1 to 1 and serves as a statistical distinguisher to identify the correct key among the key guesses. If the chosen power

model is adequate, the time series based on the key guess with the correct key will show a much higher correlation with the actual collected power measurements.

When performing side-channel attacks on the AES, the attacker's goal is to unveil the secret encryption key. It is generally assumed, that the attacker has access to the plaintext or the ciphertext – after all there would be no need for encryption if the communication channel is fully secured. In our attacks, we assume the attacker can tap the ciphertexts of an encryption stream of random inputs, where the secret key is unknown. The power side-channel measurements are n traces of length m of the victim device's supply voltage or current during the encryption process. By attacking the secret key in a byte-wise manner the complexity of a brute-force approach is greatly reduced, as instead of 2^{128} keys the attacker needs to guess only 16×2^8 keys in addition to the complexity of the analysis part. Assuming the first key byte should be recovered, an attacker performs the analysis as follows:

1. Guess a key byte hypothesis k_{hyp}.
2. Compute a model $P_{\text{mod}}(k_{\text{hyp}})$ which approximates the power consumption of a key-dependent operation in the encryption.
3. For each point in the measurement traces t of length m, compute the Pearson correlation coefficient ρ of $P_{\text{mod}}(k_{\text{hyp}})$ and t_i over all n measurement traces.
4. Repeat the computations for all $k_{\text{hyp}} \in \{0, 1, ..., 255\}$.
5. Identify, which key k_{hyp} has a significantly higher or lower correlation than all other key guesses. This key is the secret encryption key.

Finding an adequate model is critical for the success of the attack. Very often, the model is based on the Hamming weight or Hamming distance of a value after the first encryption round or before the last encryption round. In our case, where usually a ciphertext-based attack is performed, the model is based on either the hamming weight or a single bit value before the last byte substitution and round key addition. More specifically we define our model as follows, when using a Hamming weight approach:

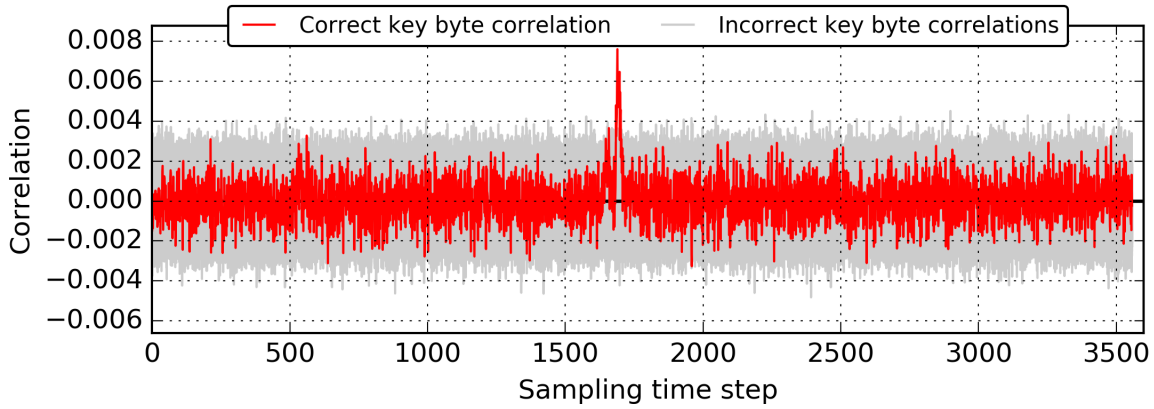
$$P^{(1)}_{\text{mod}}(k_{\text{hyp}}) = \text{HW}(\text{sbox}^{-1}(c_i \oplus k_{\text{hyp}})) \tag{2.2}$$

In the above equation, c_i corresponds to the respective byte of the ciphertext and HW to the Hamming weight of the respective value. The use of the Hamming weight as a model for power consumption, is based on the assumption, that the power depends on the amount of bitflips during a specific operation. A more generic model assumes only a general dependency on the value of a specific bit:

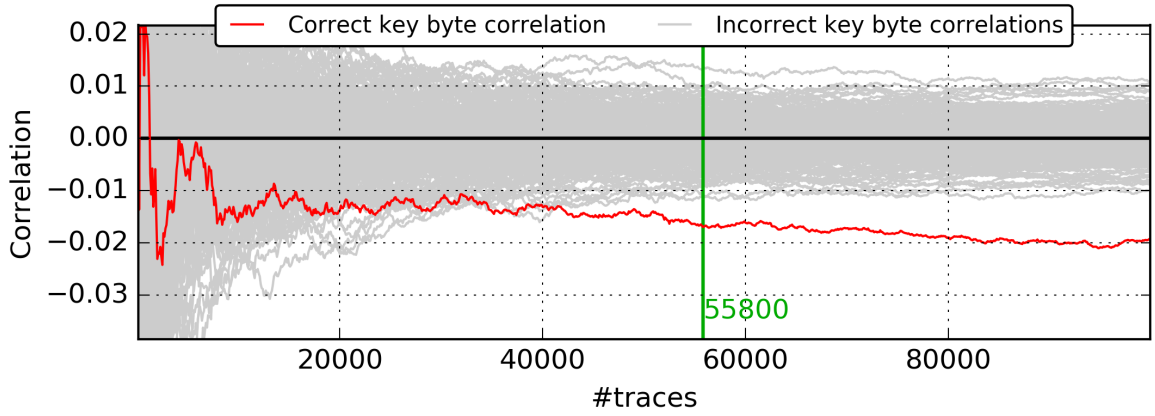
$$P^{(2)}_{\text{mod}}(k_{\text{hyp}}) = \text{sbox}^{-1}(c_i \oplus k_{\text{hyp}}) \wedge 2^b \tag{2.3}$$

This bitwise model $P^{(2)}_{\text{mod}}$ can result in an attack that is successful with fewer traces, but requires performing computations for all $b \in \{0, 1, ..., 7\}$.

The last step of the attack, which is the identification of a key byte with significantly better correlation, is typically done visually. However, for the evaluation of both attacks



(a) Total correlation values over the entire measurement trace length m.



(b) Correlation values over the amount of evaluated traces in one specific sampling time point.

Figure 2.1.: Example of total correlation values over the entire sampling time period points as well as correlation over the amount of collected traces in a specific time point when performing a Correlation Power Analysis (CPA) attack.

and countermeasures, we often want to compare the amount of traces that the attacker has to collect until the correct key byte can be identified. Therefore, we formalize a successful attack by defining a condition for identifying the correct key byte: The attack is successful iff at any point in the sampling period, the correlation for the correct key byte value is 1.5 times larger than the second-highest or 1.5 times lower than the second-lowest correlation values for incorrect key bytes.

In Figure 2.1, we present example plots of correlation values for attacking a secret AES key byte using CPA, which are used throughout this thesis. For both plots, the correlation coefficients for incorrect key guesses are colored grey, whereas the correlation with the correct secret key byte is marked red.

The first plot in Figure 2.1a shows correlations over the entire sampling period after a fixed amount n of measurement traces has been collected. Both the correct key and the point of interest during the sampling period, which is around 1 700 can be clearly identified.

Selecting a fixed sampling time point and computing the correlation coefficients over the amount of collected traces results in a progress plot as shown in Figure 2.1b.

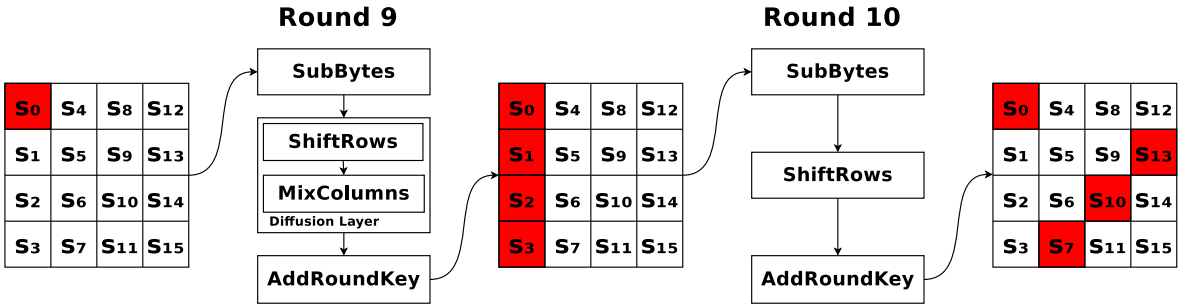


Figure 2.2.: Propagation of a single byte fault in the AES state before the 9th encryption round as shown in [13].

From this perspective, we can determine the amount of measurements that have to be performed by an attacker for a successful key recovery as previously explained. This amount is marked as a green line at 55 800 traces in the example.

2.4. Differential Fault Analysis on AES

To demonstrate the threat of fault attacks on the electrical level in heterogeneous computing systems, we employ the method described in [39], which we describe here in detail. Differential Fault Analysis (DFA) is based on the attacker being able to encrypt identical plaintexts multiple times, one time without disturbing the system and a second time when injecting a fault. If the fault is injected at a specific time during the encryption, recovering the secret key is possible.

The attack introduced in [39] is based on a theoretical fault model, i.e. an assumption about the location and the type of injected fault. Practical attacks are often imprecise, and some fault injections may be highly beneficial for key recovery, but are impossible to achieve in a real system. First, we explain the basic principle of the approach in [39] here, whereas later we will explain the methods used to make attacks practical in actual FPGA-based systems.

The fault model for the full attack in [39] is that of a random byte fault before the 8th encryption round (when encrypting with a 128bit key), which means that a single byte of the AES state is incorrect. If the attacker is able to inject faults according to the presumed model, the secret AES encryption key can be recovered in 98% of cases with only two pairs (C, C^*), where C is the correct and C^* the incorrect output for the same plaintext input.

To understand the full attack scheme, it is easier to first consider a single byte fault before the penultimate encryption round. In Figure 2.2, we show how a single byte fault in the first byte of the AES state is propagated and affects the output ciphertext, where four bytes are incorrect.

Assuming a fault in the first byte before the 9th round, the attacker can now attack the corresponding four bytes of the last round key as follows:

1. Compute all 255 possible differences $\mathcal{D}$ between the correct and the incorrect AES state after round 9. Note that the value of the 9th round key is irrelevant as it remains identical for all encryptions, and we only compute the differences.

2. Make a guess for the four bytes of the last round key $k_{\text{hyp}} \in \mathcal{K}$, where $\mathcal{K}$ is the set of key candidates, which is initially $\mathcal{K} = \{0, 1, ..., 2^{32}\}$.

3. From a pair (C, C^*) of correct and incorrect ciphertexts, compute the difference $d(C, C^*, k_{\text{hyp}})$ for k_{hyp} before the last round.

4. If $d(C, C^*, k_{\text{hyp}}) \in \mathcal{D}$, then k_{hyp} remains in $\mathcal{K}$, otherwise it is removed.

5. Repeat the process for all possible k_{hyp}.

6. Repeat the process for a new pair (C, C^*) until $|\mathcal{K}| = 1$.

The described fault analysis has a complexity of $O(2^{32})$, since the attacker has to consider the entire key space for the affected four bytes of the last round key. This complexity makes the attack feasible but it can be greatly reduced by an initial reduction of the key space using two pairs of correct and incorrect ciphertexts (C, C^*). Instead of computing differences with a guess on all four key bytes at once, the attacker generates key candidates in a byte-wise manner, checking for the existance of the corresponding differences in $\mathcal{D}$. With this approach, the initial size of $\mathcal{K}$ is significantly smaller, and the analysis can be performed within a few seconds.

To reduce the entire attack down to only two required ciphertext pairs for recovering the entire key, an attacker must be able to inject single byte faults before the 8th encryption round. In this case, the fault is propagated to affect four bytes before the 9th round and all bytes of the output ciphertext, which can then be used to attack all 16 bytes of the last round key. However, we explain later that fault injection before the penultimate round is more useful in practice, as a successful injection can be verified.

2.5. Power Delivery Networks (PDNs)

In the previous sections, we explain the general basics of fault and side-channel attacks without regard to the measurement or injection method. Here, we shed some light on the principles of PDNs and the basics behind remote attacks on the electrical level.

The PDN refers to the device-level part of the power supply hierarchy, usually starting from a board-level Voltage Regulator Module (VRM) down to individual power lines that supply single transistors on the chip. It can be modeled as a mesh of resistive, inductive and capacitive (RLC) elements as depicted in Figure 2.3. Since the VRM is always non-ideal, the supply voltage is affected by the static power drawn by the chip (IR-drop) and the designed circuit as well as its dynamic switching activity (di/dt-drop). The total voltage drop can be described by the equation $V_{\text{drop}} = I \cdot R + L \cdot di/dt$, where the dynamic drop is the dominant part in highly miniaturized circuits nowadays [40].

The fact that the supply voltage is directly affected by the activity of circuits on a board makes it a target for side-channel attacks. However, the supply voltage is not only affected by the circuits but the opposite also holds true, as a lower supply voltage increases the signal delay and transistor switching times. The high flexibility of heterogeneous computing systems may allow attackers to leverage these dependencies

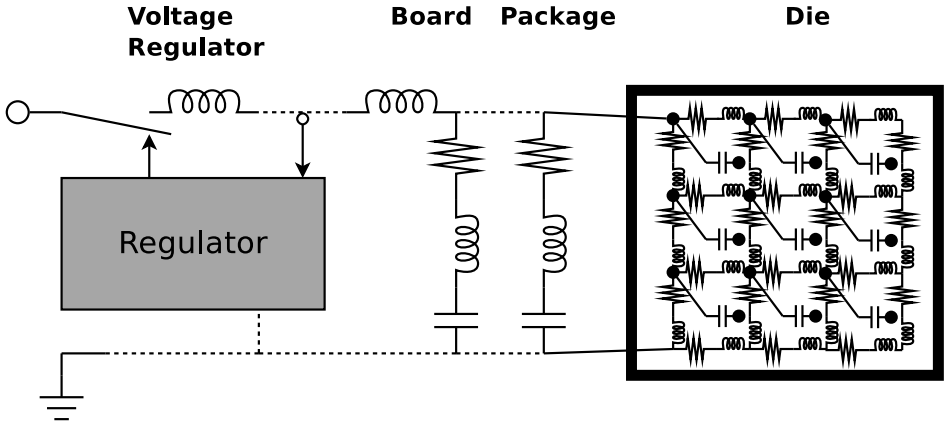


Figure 2.3.: A schematic model of the PDN as a mesh of resistive, inductive and capacitive elements from board- to chip-level.

to perform fault and side-channel attacks remotely, when data leakage between logically isolated circuits occurs through the PDN.

The threat in most side-channel and fault attacks covered in this thesis arises from the unintended misuse of circuits as sensors or power viruses that can either measure voltage fluctuations or cause voltage drops. Both attack scenarios are enabled through timing violations, where a signal is too slow to propagate a combinational logic path within a clock cycle, causing an incorrect value to be captured in a register. The timing constraints that are given by the clock period t_{clk} can be formulated as follows [41]:

$$t_{\text{clk}} > d_{\text{clk2q}} + d_{\text{pMax}} + t_{\text{setup}} - t_{\text{skew}} \tag{2.4}$$

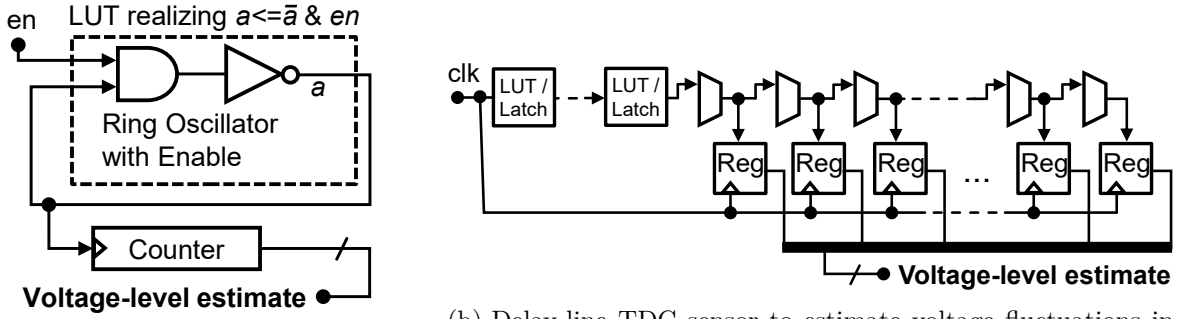
In Equation 2.4, d_{clk2q} is the delay between the rising clock edge and the register output change, d_{pMax} the maximum delay through the combinational paths, t_{setup} the setup time, which is the minimum time a signal must be stable at a register input and t_{skew} is the clock skew. To introduce a fault through a timing violation, an attacker can decrease the clock period, which is the left-hand side of the equation, or increase the delay on the right-hand side by lowering the supply voltage. However, the dependency between supply voltage and delay can also be leveraged to indirectly estimate voltage by measuring the delay, which can result in a sensor for side-channel attacks.

How exactly we leverage the above relations in our FPGA-based fault and side-channel attacks will be described in further detail in the next section.

2.6. FPGA-based Sensors and Power Viruses

In the previous Section 2.5, we describe the general relations in PDNs that are relevant for remote, electrical power side-channel and fault attacks. Here, we detail how these relations are leveraged in FPGAs specifically, which are our main platform for experiments and a major part in the heterogeneous computing trend [2–6].

FPGAs are programmable logic chips, which mainly offer a grid of LUTs, registers and interconnect, that can be programmed to realize almost arbitrary circuits. Modern cloud FPGAs offer a huge amount of resources per chip, with LUT counts in the



(a) RO-based sensor to estimate voltage fluctuations in FPGAs.

(b) Delay-line TDC sensor to estimate voltage fluctuations in FPGAs.

Figure 2.4.: Two different sensor designs from [22] to indirectly estimate voltage fluctuations inside FPGAs using FPGA-primitives to measure the transistor delay.

millions. Due to this increasing amount of resources per chip, virtualization of FPGAs is a major game changer for increasing utilization and efficiency [7]. However, multi-tenant scenarios are currently impeded by the security issues that arise from the shared chip-level PDN.

To measure voltage fluctuations in side-channel attacks on multi-tenant FPGAs, different approaches are employed, two of which we present in Figure 2.4. Both sensor variants are based on measuring fluctuations of the transistor delay, which in turn depends on the current supply voltage as explained in the previous section. The first sensor in Figure 2.4a, is based on a RO, which is fed into a counter that counts the oscillations over a specified time slot. A lower supply voltage and the consequent higher transistor delay results in a lower oscillation frequency and thus a lower value being counted. Another sensor variant is depicted in Figure 2.4b, which is based on the principle of a TDC. A clock signal is propagated into a delay line, often composed of carry-chain primitives, where the extent of the propagation is captured into registers. Here, the higher transistor delay causes the signal to propagate less into the delay line and likewise results in a lower measurement value.

Both sensor variants have advantages and disadvantages, the most obvious being the significantly lower sampling rate that can be achieved with RO-based sensors. For that reason, we employ TDC-based sensors in all our experiments. A disadvantage of the TDC-based sensor is the susceptibility to intra- and inter-chip process variation, which requires manual or automatic calibration of the length of the delay-line, more specifically the initial delay, which is not captured into registers. In the respective sections, we explain further implementation details regarding the adaption of TDC-based sensors to specific FPGA technologies.

Whereas a side-channel attacker measures the transistor delay variations through timing violations in the attacker design that are caused by the voltage fluctuations during the computations of the victim module, a fault attacker aims to increase the transistor delay to an extent where timing violations occur in the victim design instead. For that purpose, power viruses are deployed onto the FPGA, which intentionally create a lot of switching activity to eventually decrease the supply voltage down to a critical

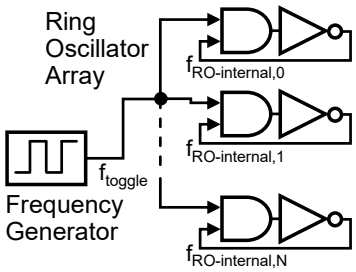


Figure 2.5.: Basic principle of a RO-design to create voltage fluctuations for fault injections in multi-tenant FPGAs as shown in [22].

level. The initial designs from [10, 13], are based on large amounts of ROs, which are implemented as a single NAND-gate in a LUT of the FPGA. The basic principle is shown in Figure 2.5, where the ROs oscillate with their internal frequency and cause high switching activity, while an additional toggle signal is used to activate and deactivate the RO-array at a specific frequency and duty-cycle. A dependency on both of these parameters to maximize the voltage drop was shown in [10], where FPGAs were successfully crashed and required a power cycle to recover.

We show in this thesis how designs for fault injection can be detected by a cloud hypervisor to mitigate such remote fault attacks and also how attackers can make the attack harder to detect, replacing the obvious ROs with seemingly benign circuits.

Part II.

Contributions

3. Exploration of Fault and Side-Channel Attacks

The work on advanced, stealthy designs for fault attacks has been published in [20] together with Dennis Gnad and Mehdi Tahoori, whereas the work on classifying code sequences through side-channel measurements has not been published before.

To enable the development of countermeasures against side-channel and fault attacks on the electrical level, a thorough analysis of the threat on the attacker side is important. In this chapter, we explore how existing attacks, such as [12] and [13], can be extended in terms of target devices, preventing detection and extent of the threat in the power supply hierarchy. For fault attacks, we evaluate the use of seemingly benign logic circuits for precise fault injection leading to a full AES key recovery, whereas for side-channel attacks, we analyze how much leakage can be observed through much higher levels of the power supply hierarchy by classifying code execution patterns running on a CPU via measurements on an FPGA accelerator in the same system. The results highlight the importance of developing countermeasures against these remote attacks.

3.1. Remote and Stealthy Fault Attacks on Virtualized FPGAs

Activity of a small fraction of FPGA logic can cause a sufficiently excessive voltage drop to crash an entire FPGA, together with an integrated CPU, if the correct pattern of switching activity is applied [10]. More recently, we demonstrated how the generation of voltage fluctuations can be carefully calibrated to inject faults exactly into the penultimate round of an AES encryption to fully recover the secret key within a few seconds or minutes [13]. In this initial work, ROs were used to maliciously manipulate power consumption in a way to impact supply voltage. Follow-up works have shown that it is possible to cause timing faults not only with ROs, but also with sequential oscillators [42], memory access collisions [15], or even by toggling a large amount of AES modules [43].

Here, we provide an extensive analysis of how attackers are able to induce timing faults and perform elaborate fault attacks by using seemingly benign logic for generating voltage fluctuations. We deploy key recovery attacks on an AES module and evaluate the attack on a broader selection of devices from different manufacturers. Moreover, we discuss how devices are affected differently and draw conclusions for improving attacks as well as countermeasures.

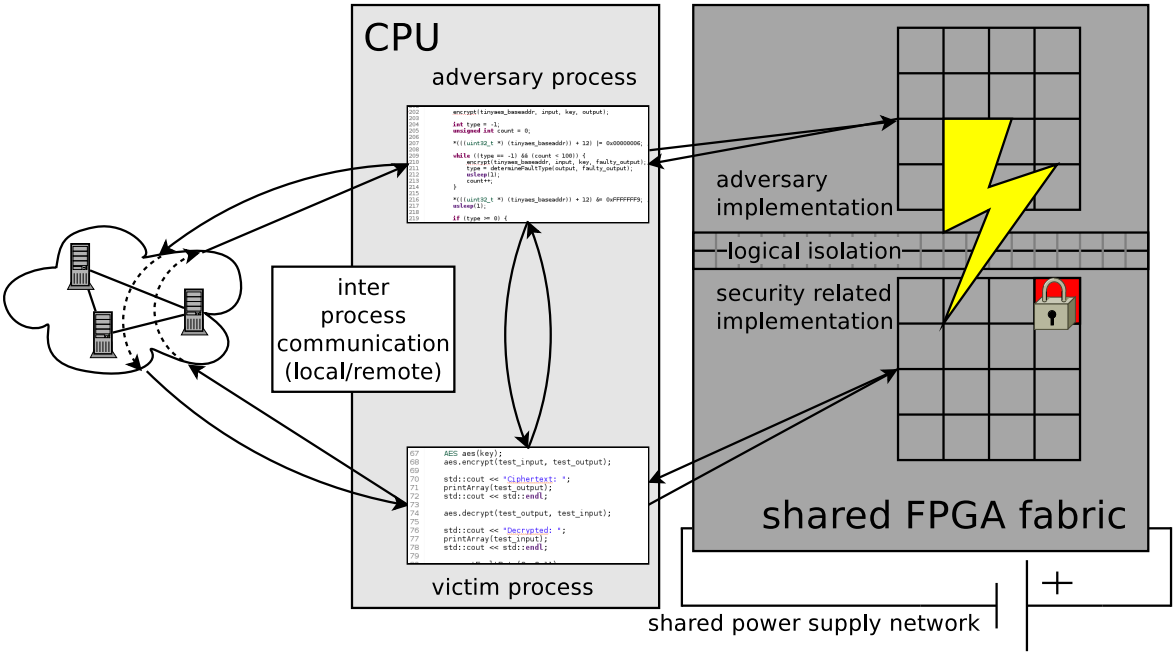


Figure 3.1.: Overview of the threat model as shown in [13]: Attacker and victim share an FPGA resource with a common power supply network, but isolated, logically disconnected partitions on the fabric

In the next Section 3.1.1 we explain the proposed threat model. Section 3.1.2 details our experiments as well as the used devices, and different methods for causing voltage fluctuations are detailed in Section 3.1.4. In Section 3.1.5, we present results on attacking an AES encryption module, showing key recovery and fault injection rates. We discuss new findings in Section 3.1.6 and conclude this work in Section 3.1.7.

3.1.1. Threat Model

A brief overview of the assumed attacker-victim scenario is given in Figure 3.1. We assume the victim and the adversary to have access to a fraction of an FPGA, in which they can load their own arbitrary design, like a cryptographic accelerator. Both attacker and victim have their respective software processes in an operating system, and their designs on the FPGA are logically and spatially separated, and follow other common best practices as explained in [44]. It is also assumed that the respective FPGA fabric is powered by a single common power supply. This scenario includes both data-center applications, in which FPGAs are utilized as standalone accelerators, as well as SoC platforms, in which multiple processes on the CPU can utilize a fraction of the FPGA logic.

We assume the victim to utilize their part of the programmable logic for a security related algorithm, such as a block cipher. A secret key used in this algorithm is either hard-coded onto the FPGA or transferred at runtime. The attacker is able to interact with the victim through a public interface, for instance, to issue encryption requests.

Vendor	Board	Device
Lattice Semiconductor	iCE40HX8K-B-EVN	iCE40HX8K
Intel	Terasic DE1-SoC	Cyclone V SoC
Intel	Terasic DE0-Nano-SoC	Cyclone V SoC
Intel	Terasic DE10-Pro	Stratix 10 SX SoC

Table 3.1.: Overview of the evaluated platforms

3.1.2. General Setup and Devices

The attacks are performed on devices from two different manufacturers, namely FPGAs and FPGA-SoCs from Intel/Altera and Lattice Semiconductor. In all cases, two logically and spatially separated designs are deployed on the FPGA fabric, whereas the software part of the threat model is realized as a single process for simplicity. Where a hard processor is available in the FPGA-SoCs, we run the software part inside the Linux system, running on that integrated processor. The smaller devices from Lattice Semiconductor are attached through a serial interface, where the software part is running on the host computer.

The AES implementation used as a proof-of-concept in this work is a simple, small module for 128-bit key length encryption. It utilizes around $300 - 400$ registers and about $750 - 850$ LUTs in the tested FPGAs and takes 50 clock cycles to encrypt a given plaintext. The module is not protected against side-channel or fault attacks.

Development tools for synthesis as well as placement and routing from all vendors offer tools for Static Timing Analysis (STA), which analyzes the design in terms of timing violations under different models (corners) for a given device with a specific speed grade. We evaluate both designs where no timing violations are reported by the analysis, which we refer to as constrained designs, and unconstrained designs, where timing violations are reported, but the encryption module still works as expected. Naturally, attacks on fully constrained designs are more difficult, as the margin for a timing fault to occur is much higher. In our results, we show how on some devices only unconstrained designs can be attacked, whereas on other devices even modules, which do not violate timing constraints according to the vendor tools, can be attacked. Table 3.1 provides an overview on the platforms where we implemented our attack.

3.1.3. Automated Parameter Calibration

As described in Section 2.6, parameters such as frequency and duty-cycle for toggling the malicious logic need to be adapted to provoke faults at the proper moment of the encryption. To make use of the possibility for injection success verification when injecting before the 9th encryption round, we develop an automated calibration algorithm, to be executed before evaluating injection rates or attack success for a given design and device. The algorithm allows using the attacker design in different setups, without the need for finding appropriate parameters in time-consuming trial-and-error experiments manually.

We adapt the signal for activating the power wasters in three parameters: The toggle frequency, the duty-cycle and the delay between starting the encryption and activating the toggling. The algorithm performs as follows:

a) The attacker activates the calibration process on the FPGA. A random input plaintext is encrypted without malicious activity. The result is stored as the correct ciphertext.

b) Afterwards the fault injection on the FPGA is activated, to enable switching activity for the following encryptions.

c) Encryption of the same random plaintext is requested. The attacker design on the FPGA activates the power wasters with an initial frequency and duty-cycle and no activation delay. If no fault is detected, the frequency/duty-cycle are decreased/increased. If a fault is detected, which affects an undesired subset of bytes, the injection occurred too early or too late, and the activation delay is increased/decreased. In any case, the attacker design reports the injection success to the attacker process, which either requests another encryption or continues the process.

d) If the injection was successful or a predefined maximum of injection attempts inj_{max} were unsuccessful, the attacker software deactivates its corresponding FPGA design and either finishes the successful scan or chooses another random plaintext for fault injection.

3.1.4. Novel Power Wasting Circuits

Whereas the initial work on chip-internal fault attacks [13] made use of a large RO-grid to generate voltage fluctuations, we explore two additional options in this work. The main objective is to investigate whether benign-looking logic can be used to inject precise faults in the threat model of multi-tenant FPGAs. In [43], among other approaches multiple AES encryption rounds performing random computations at a very high clock frequency are used to inject timing faults into a ripple-carry adder design. We confirm that AES modules can be employed as power wasters in our attack setup as well and also successfully evaluate the *s1238* circuit from the ISCAS'89 benchmark collection [45] to inject faults with seemingly benign logic. The *s1238* is described as a combinational circuit with random flip-flops in [45], which we assume to be beneficial in terms of switching activity, unlike, for instance, a comparatively static state machine circuit.

For the RO-based attack, the oscillators are implemented with a single LUT using various Verilog keywords to keep the synthesis software from optimizing the grid away. On Intel/Altera FPGAs this can be achieved by declaring virtual output pins, which are placed as fanless LUTs on the chip. For the Lattice Semiconductor toolchain, using the *keep* or *syn_keep* directives is enough to prevent optimization. The ROs can be enabled or disable through a global signal, which – on Intel/Altera FPGAs – is routed on global clock routing resources to prevent long compile times and congestion.

To inject faults with AES modules or the *s1238* benchmark circuit, we configure a Phase Locked Loop (PLL) on the chip to generate a 750 MHz clock signal, driving the respective registers without regard for any timing violations that might occur. As with the RO grid, a global toggle signal is used to switch the instances on or off with the correct parameters that have been determined by the calibration algorithm explained in the previous Section 3.1.3. Again, any optimization is prevented through the use of Verilog keywords and virtual pin instantiations. A major difference to injecting faults with ROs is the choice of input values for the power wasters, which is not trivial as the switching of gates and registers inside the designs should be maximized. For the AES modules, computing an encryption stream where the output ciphertexts are fed back as the new input to the circuits is sufficient. This is most likely due to their well distributed switching behaviour, as a good encryption algorithm should be generating ciphertexts that cannot be distinguished from random output. The *s1238* from the ISCAS'89 benchmark is initially driven with a zero input, then we calculate the new input value in_n at clock cycle n as $\text{in}_n = \overline{\text{in}_{n-1}} \oplus (\text{out}_{n-1} << 1))$. Intuitively, this approach initially flips the maximum amount of bits in the input value, but also takes the circuit output into account to prevent the circuit from getting stuck in a specific state.

3.1.5. Results

Our main experimental platform is the Terasic DE1-SoC board, which incorporates an Intel/Altera Cyclone V FPGA-SoC. On this board we present fault injection rates and key recovery statistics as shown in [13] as well as the additional results using AES modules and benchmark circuits for causing voltage fluctuations. Lastly, we present results on the iCE40-HX8K FPGA from Lattice Semiconductor as well as the Intel/Altera Stratix 10 SoC on the Terasic DE10-Pro board.

3.1.5.1. Fault Injection Rate

In order to evaluate the general fault injection efficiency, we first generate bitstreams for different percentages of logic utilization of the attacker logic in the range of 30% to 50% for the DE1-SoC board. The victim module runs at a frequency of 111 MHz, which does not violate any timing constraints as explained in the previous subsection, even in the worst-case corner of the STA (slow 1.1V/85°C model). Then we measure the number of faults occurring for one million encryption requests from a previously generated set of random plaintexts, which are reused for all experiments. The encryption key remains the same for one test series, which is repeated for a second random key.

Figure 3.2 shows the total number of faults out of 1M trials F_{tot} as well as the number of faults usable for DFA with our described fault model F_{DFA} depending on how much of the available logic resources is occupied by the attacker design. We see that both F_{tot} and F_{DFA} initially increase at the same rate. This proves the effectiveness of the calibration algorithm before each evaluation. However, if the amount of activated ROs exceeds a certain level, the effect is too strong to allow precise injections. F_{tot} still increases with more ROs, but can affect more than one round or byte per round.

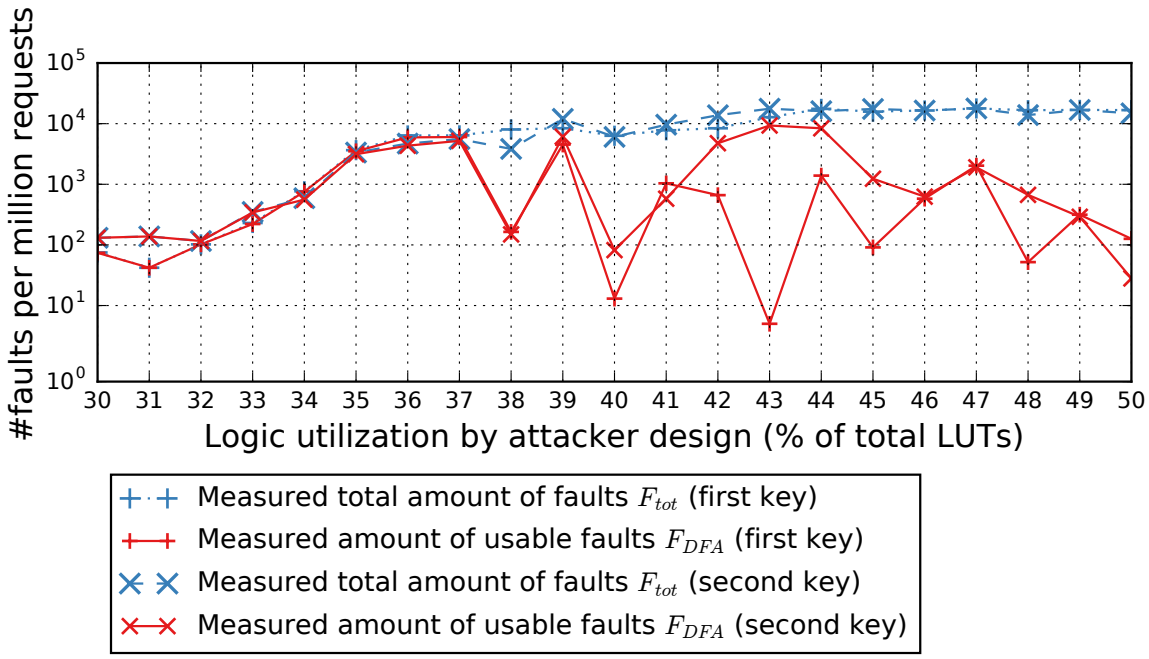


Figure 3.2.: Total measured fault injection rates F_{tot} and measured injection rate of faults usable in DFA F_{DFA} with respect to the amount of logic utilization (percentage of total LUTs) by the attacker design for two different random encryption keys

Hence, the resulting ciphertext will have more than four byte faults, which cannot be used anymore to recover the secret AES key in the used fault model.

We repeated the experiment on three Terasic DE1-SoC boards of different age and usage history with similar results. The minimum and maximum amount of logic that allows ideal injection rates is slightly different for each board, but a common suitable region that covers all devices can be determined.

3.1.5.2. Key Recovery Success Rate

Subsequently, we evaluate the success of the full DFA attack including recovery of the secret AES key. This evaluation reflects on the success of our entire algorithmic flow of injecting faults before the 9th AES encryption round, the calibration algorithm and filtering of undesired faulty ciphertexts. For each of the three boards, which we already used to investigate fault injection rates, we use the amount of ROs that lead to the highest injection rate F_{DFA} of faults usable for DFA. We generate a set of 5 000 random AES keys and collect a minimum of two ciphertext pairs, which exhibit faults at the desired positions, for each four bytes of the last AES round key. The ciphertexts along with each key are stored on the SD card of the board and later transferred to a host computer.

Figure 3.3 summarizes the results of the key recovery attempts on our three DE1-SoC boards. On all three boards, we are able to deploy the attack successfully recovering all 16 key bytes correctly for at least 87.9% of the 5 000 random keys. All recovered keys are correctly recovered, so no false positives are encountered. On all boards, we have a

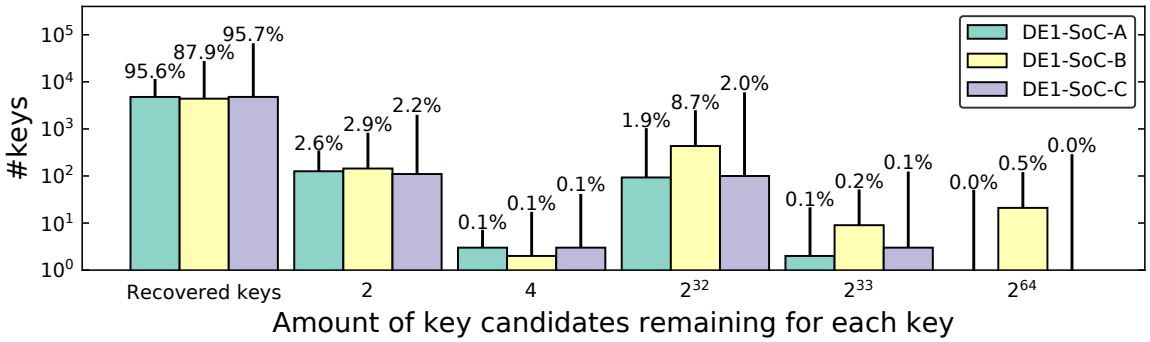


Figure 3.3.: Amount of key candidates remaining for each key recovery attempt on 5 000 random AES keys on three different DE1-SoC boards, using ROs to cause voltage fluctuations.

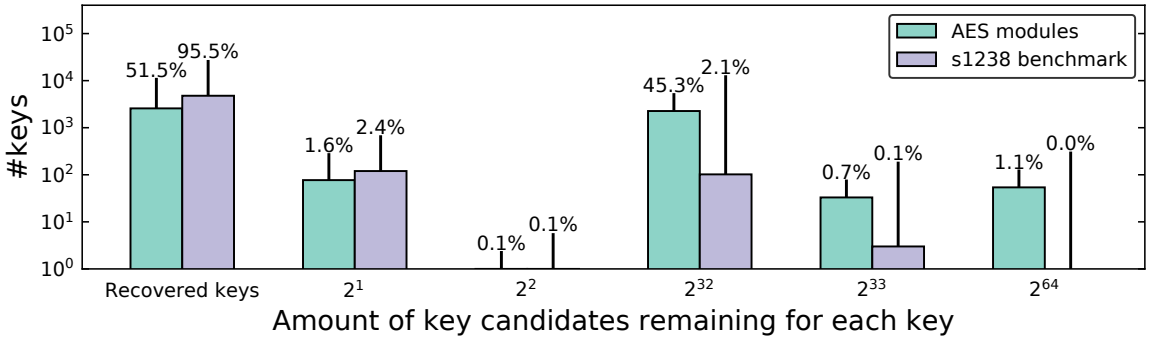


Figure 3.4.: Amount of key candidates remaining for each key recovery attempt on 5 000 random AES keys, using AES modules or *s1238* benchmark circuits to cause voltage fluctuations.

small amount of around $2-3\%$ of all keys which cannot be recovered, but less than four candidates for the last round key remain. This ratio confirms the results in [39], showing that in about 2% of the cases more than two ciphertext pairs are necessary to recover the AES key. If a sufficiently small amount of key candidates remains, the correct key can be easily recovered with an exhaustive search. We encounter, however, some keys, where more than 2^{32} or even 2^{64} candidates remain. Across all our experiments, an average of 22 usable faults were required to gather the required two ciphertext pairs per four bytes of the round key. To collect these pairs, the attacker design needs to issue 17 979 encryption requests on average to the AES module, which took on average 2 344 ms. The average time for the evaluation of one attack until key recovery on the described host machine is about 107 ms.

3.1.5.3. Using Benign Logic for Fault Injection

To extend on the original results from [13], we perform fault attacks using seemingly benign circuits described in Section 3.1.4. We perform the attacks on the Terasic DE1-SoC, replacing the previously used ROs with either AES modules or the ISCAS'89 *s1238* benchmark circuit. For successful key recoveries, we need 60 AES modules

or 280 benchmark circuits, corresponding to 50.1% and 64.7% of the available logic resources each. The larger amount of logic required when using the *s1238* is most likely due to the higher amount of switching occurring when operating AES modules. In both cases, the instances are operated at a frequency of 750 MHz.

In Figure 3.4, we again present results of attempting to recover 5 000 randomly drawn secret AES keys using seemingly benign logic for fault injection. Interestingly, fewer keys can be recovered when attempting the attack using AES modules as power wasters for generating voltage fluctuations, whereas the recovery rate when using the *s1238* circuits is very close to an RO-based attack. We observe a high amount of keys, where 2^{32} candidates remain, when using AES modules. Since faults that are injected too early or too late are filtered, the only valid conclusion is that more multibyte faults are being injected in this case. A possible explanation is the size of the AES modules, compared to the *s1238* modules or ROs, which may result in voltage spikes of slightly longer duration thus affecting multiple bytes.

3.1.5.4. Experiments on Different Platforms

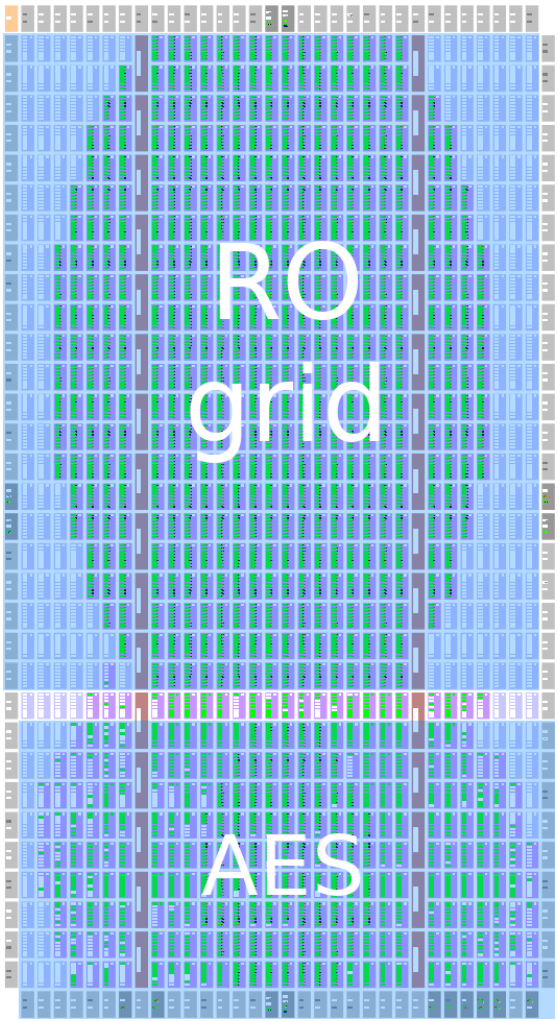
We also confirm that the attack is successful on different platforms from different manufacturers. In addition to Cyclone V devices, we present results on an FPGA from Lattice Semiconductor, as well as a server-grade Stratix 10 SX SoC on the Terasic DE10-Pro board.

In Figure 3.5, we present the evaluation designs on the two additional devices. On the Stratix 10, about 12% of the available logic resources are used for the RO grid (115 000 ROs), whereas on the small low-power FPGA from Lattice Semiconductor the attacker uses exactly 50% of the 7 680 available LUTs. Empirically we find this to be the maximum amount of ROs that can be deployed with a single enable signal on the Stratix 10 device, otherwise the board crashes during programming. Whereas a more elaborate toggle signal design may enable to deploy a larger amount of oscillators, the amount is already sufficient to inject faults into an AES module running at 255 MHz. In this setup, the AES module is reported as constrained in the fastest timing corner and works without errors when the attacker design is disabled.

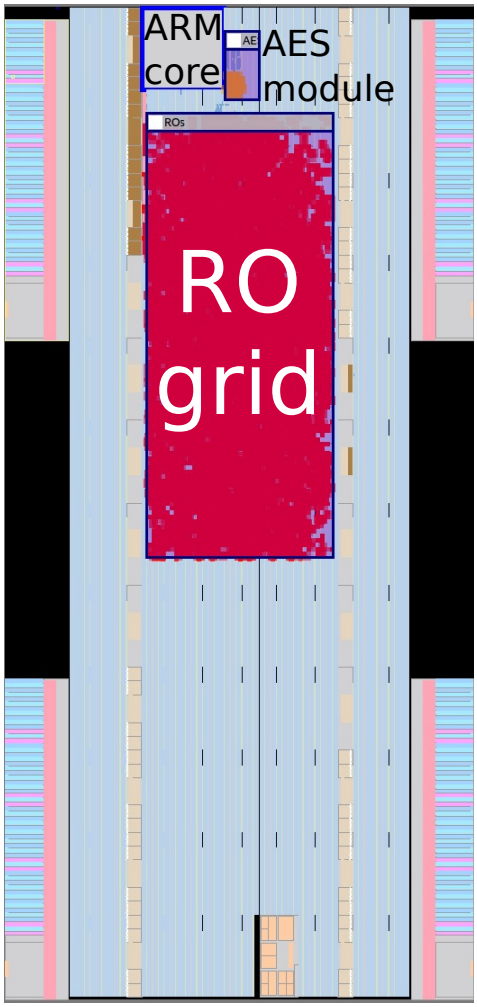
On the Lattice iCE40-HX8K FPGA, the AES module operates fully constrained at 24 MHz and we are able to recover the secret AES key within a few seconds. Increasing the amount of ROs on this board, we are also able to cause a complete crash, where only a power-cycle can re-enable the device. An important observation here is that the crash seems to affect the PLL of the devices, whereas purely combinational logic or externally clocked registers remain functional. Further investigation on which exact components are affected by voltage fluctuations may be a promising direction for future research and could result in effective countermeasures.

3.1.6. Discussion

The results presented here make the development of countermeasures more critical but also more difficult. Recently proposed solutions [24, 25] are based on checking bitstreams for potentially malicious signatures, which we also present in this thesis in Chapter 5. These offline bitstream scanners may need to be adapted to be able



(a) Floorplan of the design on the Lattice iCE40-HX8K FPGA



(b) Floorplan of the design on the Stratix 10 SX SoC

Figure 3.5.: Evaluation designs on devices from different manufacturers as seen in the respective floorplanning tools.

to detect injection logic that makes use of AES modules or other benign circuits for injection. More generic approaches, such as the estimation of a design's maximum switching activity could overcome the need for continuously introducing new virus signatures, but will be computationally expensive.

However, experiments on different platforms show, that the switching activity may affect different parts of the devices. When crashing the Lattice iCE40-HX8K FPGA the PLL is affected, whereas purely combinational logic or flip-flops driven by external oscillators can glitch but will still be responsive. Potential future countermeasures may be based around creating exclusive voltage lanes for critical components of the FPGA, such as the clock generators or hypervisor logic.

On the attacker side, we prove how the selection of input values to benign logic that is used for creating voltage fluctuations is critical to the attack success. Whereas we empirically select inputs based on successful fault injection, a more elaborate method would be to find an input sequence leading to the maximum amount of switching gates in a specific module. Although finding such an input sequence may need exponential computational effort depending on the size of the circuit, it can discover critical inputs even in non-reconfigurable hardware.

3.1.7. Conclusion

In this section, we demonstrate how elaborate remote fault attacks in multi-tenant FPGAs do not require an attacker to use highly specialized logic and can be deployed on devices from different manufacturers. With seemingly benign logic circuits the attacker is still able to cause voltage fluctuations to inject timing faults into an AES hardware encryption module on FPGAs and FPGA-SoCs from Intel/Altera and Lattice Semiconductor, proving both embedded and cloud devices vulnerable. Using a calibration mechanism to inject faults exactly before the 9th AES encryption round, it is possible to recover up to 90% of secret AES keys on the Cyclone V FPGA-SoC, without any logical connection between attacker and victim design. The results emphasize the importance of developing countermeasures to enable secure FPGA virtualization.

3.2. Classification and Detection of Code Patterns through the Power Supply

Recent publications show how covert-channels can be established through the Power Suppy Unit (PSU) of a desktop computer from the processor to an FPGA-based PCIe accelerator card [19]. Information leakage from hard processor systems to FPGAs, which we explore in this section, is especially critical in SoCs, where CPU and FPGA are integrated onto a single chip with a tightly shared PDN. Whereas covert-channels are useful to establish secret communication between different components in a complex attack scenario [19], we analyze whether FPGA-based sensors can be used passively by an attacker to deduce code patterns running on the CPU. This capability is useful to an attacker in multiple scenarios:

- When performing fault or side-channel attacks, an attacker needs to identify the timeframe during which the victim is running the target code, for instance, the beginning of an encryption. The more precise this trigger mechanism is, the faster an attack can succeed.

- Covert-channels are means of communication to leak information from a secure context into an insecure one, which can be accessed by an attacker. Previous works already demonstrated covert channels between different components through a PC PSU. [19] However, the less CPU activity is needed to convey a secret message, the harder it is to detect for the victim or a hypervisor. Moreover, victim software can be enforced to leak information unintentionally, for example, in the context of Microarchitectural Data Sampling (MDS).

- If enough accuracy can be achieved, FPGA-internal sensors can even enable side-channel-based disassembly and reverse engineering on instruction level [46].

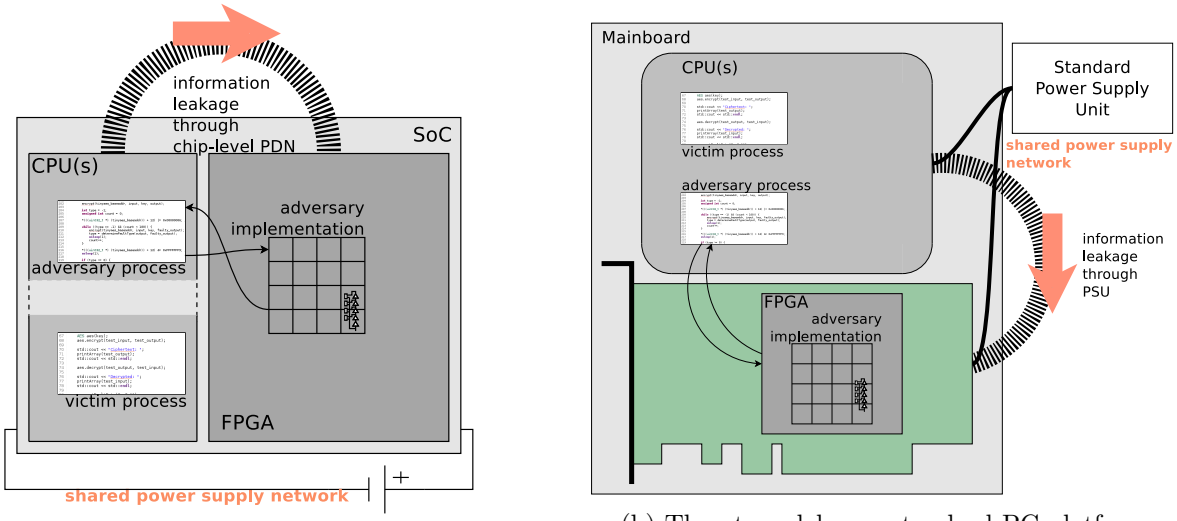
The above attack vectors motivate to analyze the detection accuracy that can be achieved with FPGA-based sensors, when targeting CPU execution patterns. In this work we implement sensors in an FPGA-SoC, namely the Xilinx Zynq 7000, as well as on PCIe accelerator cards incorporating the Xilinx Kintex-7 FPGA. The two selected platforms comprise different levels of a shared PDN between FPGA and CPU. Whereas the two components share the chip-level PDN on the Zynq 7000 FPGA-SoC, the Kintex-7 accelerator card is only sharing the ATX PSU of the desktop/server system with the CPU. This choice of test platforms covers FPGA applications from the edge (IoT) to the cloud (PCIe accelerator).

On both test systems, we classify sensor traces, which are collected while running specific code patterns on the respective hard processor cores. With the help of feature extraction algorithms and random forest classifiers, we are able to distinguish code patterns on both platforms with high accuracy. On the FPGA-SoC we can classify patterns with an accuracy of 99%, whereas in the desktop/server system up to 52% can be achieved, which is significantly better than the 20% guessing accuracy achieved by a random uniform classifier. Moreover, on the Zynq 7000, we can even detect recurring code patterns in a single long trace with a sliding window approach, proving the method suitable for the previously mentioned attacks.

We summarize our main contributions as follows:

- We show that FPGA-internal sensors can be used to classify code running on a CPU, when the devices share a power supply at a level up to a PC PSU. Contrary to covert-channels, which require an attacker to execute specific code on the CPU to cause excessively high voltage fluctuations, the classification is entirely passive, avoiding detection from hypervisor or victim. To the best of our knowledge, this is also the first demonstration of entirely passive side-channel leakage from CPU to FPGA through a standard workstation power supply.

- We demonstrate how the sensors can be also used to detect patterns within a measurement stream on an FPGA-SoC, which enables trigger mechanisms through side-channels only.



(a) Threat model on an FPGA-SoC

(b) Threat model on a standard PC platform

Figure 3.6.: Overview of the threat model, where information about executed code is leaked through the power supply on the respective platform

- Last but not least, we show that even single data accesses on an FPGA-SoC can be distinguished w.r.t. whether the data is cached or needs to be fetched from main memory. This effectively enables remote cache attacks that are based on measuring voltage fluctuations instead of timing, bypassing simple countermeasures that are based on restricting access to accurate timing measurements.

In Section 3.2.3 we detail the hardware and software setup that is used in our experiments, and we present results in Section 3.2.4. The results are discussed in Section 3.2.5, and we draw some conclusions in Section 3.2.6.

3.2.1. Threat Model

In Figure 3.6, we provide an overview of the threat model we consider in this work. Both attacker and victim are running code in a multi-user system, where access to FPGA fabric is given to the attacker. This may be an SoC in an IoT setting, as presented in Figure 3.6a, or a desktop/server system in the cloud like in Figure 3.6b. In a cloud computing scenario, the victim may also be the hypervisor, where the attacker attempts to break free from a virtualized environment.

In both cases, the power supply to the CPU and FPGA devices is shared at a specific level in the power delivery hierarchy. On the SoC, the chip-level PDN is shared, whereas in a desktop or server both CPU and the PCIe FPGA accelerator are attached to the PSU, sharing the power supply at a much higher level.

The mechanism leaking information from the CPU to the FPGA is identical in both cases. Switching activity from the CPU, which is caused by the victim code execution, impacts the supply voltage, as no power supply can be designed to maintain the supply voltage on a perfectly constant level. This local voltage drop propagates through the entire chip- and board-level PDN, up to the system-level power supply. The supply voltage fluctuations can then be measured on the FPGA by an attacker using sensors,

which are explained in Section 2.6. Due to the specific characteristics of the voltage traces, the attacker is able to classify measurements according to which code is run by the victim. Moreover, the attacker can use continuous measurements to detect a specific code pattern. This *live detection* of trigger code can then enable further malicious activity, such as fault injection.

3.2.2. Feature Extraction and Classification

In previous works on CPU execution pattern or instruction detection via side-channels, many methods have been employed to post-process sampling traces and eventually classify the traces [46–48]. Usually, a first step is to reduce the dimensionality of the problem. In [46], both Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) are evaluated for reducing the dimension of the classification subspace. Although the traces we sample internally are shorter than traces collected externally with an oscilloscope, we apply a feature extraction and filtering algorithm from [49].

After extracting and filtering relevant features from the raw measurement traces, we classify the resulting feature vectors using random forest classifiers. Random forests are decision tree machine learning algorithms, where the training set is split up to train multiple instances of decision trees and classification is performed via majority vote. In addition to the classification, tree-based classifiers provide a feature importance ranking, which we use to further reduce the amount of features used to 50, which has no significant impact on the accuracy. Moreover, we can extract not only a single class for each trace, but the probability $p_A(x)$ of trace x belonging to class A. This probability is helpful for detecting recurring code patterns in a stream of measurements with previously trained classifiers.

To classify a fixed-length sensor trace, we extract relevant features as described and apply a trained random forest classifier. When extending our approach to perform live detection of specific patterns within a measurement stream, we apply the previously trained classifiers on a sliding-window interval of the long measurement stream. Based on the probability given by the classifier, we can decide whether a pattern is detected or not. The specific parameters for all the mentioned algorithms and methods are detailed in the next section.

3.2.3. Experimental Setup

In this section, we first provide details on the hardware platforms that we use for detecting CPU code patterns via FPGA-internal sensors on an x86-64 system and an ARM-based FPGA-SoC. Moreover, we describe the experiments as well as the specific example code patterns that we classify.

3.2.3.1. Hardware Platforms

As explained previously, we use two setups for our experiments, one is a desktop/server PC system, where the information is leaked through the PSU from the CPU to a PCIe accelerator card. The other system is an FPGA-SoC, where an ARM CPU and

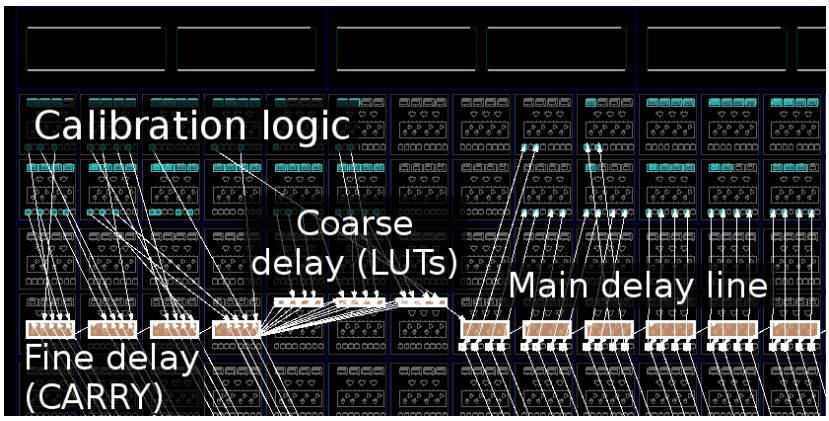


Figure 3.7.: Floorplan of the TDC sensor on the Xilinx Zynq 7000 FPGA as seen in Xilinx Vivado

programmable logic are coexisting on a single chip, communicating via AXI bus. Here we provide more details about each system and the respective setup parameters:

- Our experimental platform for detecting execution patterns through a standard ATX PSU is based on a Supermicro X8DT3, which is a dual-CPU mainboard, with two Intel Xeon E5630 and a HEC-700TE-2WX 700W power supply. The FPGA accelerator is the Xilinx Kintex-7 KC705 evaluation board. We attached two FPGA boards simultaneously in order to also experiment with artificially increasing the system load and stress on the PSU, which has led to higher leakage in previous work [19]. Additionally, an NVIDIA GeForce GT 240 is present in the system. Initial experiments with modifying the CPU clock multiplier in the BIOS did not show any improvement in the classification, which is why all BIOS settings are left at their default values. An unmodified Ubuntu 19.10 Linux operating system is installed and runs both attacker and victim software.

- On the other hand, we evaluate classification of code patterns as well as detection of code in a measurement stream on the Digilent Pynq-Z1, which is based on the Xilinx Zynq XC7Z020 FPGA-SoC, an ARM Cortex A9 dual-core CPU coupled with Artix-7 based programmable logic. Here, we experimented with decreasing the CPU clock frequency manually as well, and found no increase in classification rate during some initial experiments, which is why we leave the CPU clock at 650 MHz. The board is unmodified and so is its default PetaLinux operating system.

On both FPGAs, we are able to use identical sensor implementations according to the basic principle presented in Section 2.6 as both are based on the Xilinx 7-series FPGA architecture, with the same low level primitives available as building blocks for the sensor. The sensor values are sampled into the FPGA-internal block memory (BRAM). Those values can be read from the respective Linux system through either a memory-mapped PCIe or AXI bus.

In Figure 3.7, we exemplarily present the sensor implementation on the Xilinx Zynq 7000 as seen in the Xilinx Vivado design software. The separate parts of the initial delay are clearly visible, with a rather coarse calibration through multiple LUTs. Fine

adjustments can be made by modifying the entry point into a line of CARRY elements, which is eventually routed into the main delay line. The main delay line is sampled into registers and decoded into a 6-bit value. On both systems, we run the sensor at 100 MHz, which shows the best accuracy during classification and detection. Generally, one might assume that a higher sampling frequency is beneficial. However, a lower sampling frequency also causes the calibration logic to include more initial delay elements before the sensor delay line, which may increase the sensibility of the sensor to voltage fluctuations, as more elements are affected. This effect was also briefly mentioned in [12].

3.2.3.2. Code Patterns

To evaluate detection rates, we need a set of test patterns, which are run on the CPU, while sampling measurements on the FPGA. For that purpose, we create a set of small assembly code snippets, which are repeated over a short time period. The snippets are selected from different categories of instructions, such as memory access, floating-point computations or integer arithmetics.

Listing 3.1: ARMv7 inline assembly code snippet, which performs some integer arithmetics and is used as a test pattern to be detected on the Xilinx Zynq 7000.

```
/* The assembly code is aligned at the
   cache-line size, to avoid instruction
   cache effects */
.balign 32

/* [gp] is a pointer to the memory-
   mapped FPGA, where storing a 1
   triggers the sampling process: */
mov r4, #1
str r4, [%[gp]]

mov r10, #3
mov r11, #5

/* Repeat the following integer
   arithmetics 2000 times */
.rept 2000
add r5, r5, r10
sub r6, r5, r11
eor r6, r6, r5
and r6, r6, r11
.endr
```

In Listing 3.1, we show an example of our test set for the ARM CPU, which performs integer arithmetics while the FPGA is sampling sensor traces into the BRAM. The FPGA design is programmed to sample traces of either 2 048 (short) or 573 440 (long) measurements, depending on whether we want to evaluate simple classification or live

Table 3.2.: An overview of all example code patterns we employ on our two testing platforms to evaluate classification and detection rates.

Name	Instruction category	Used ARMv7 instructions	Used x86 instructions
NOP	Only NOP	*mov*	*nop*
MEM(C/NC)	Memory access (cached/uncached)	*ldr*	*xorb*
CPU1	Multiplication	*smlal*	*mull*
CPU2	Simple integer arithmetics	*add,sub,eor,and*	*addl,orl,subl,andl*
CPU3	Floating point arithmetics	*vmul,vadd,vsqrt, vsub*	*faddp,fsqrt*

detection. Long measurements are limited by the FPGA-internal memory capacity, in this case the capacity on the Zynq 7000. In both cases, setting a specific bit in the memory-mapped FPGA design triggers the sampling process for a single trace. This trigger mechanism can be seen in the example code, where the value 1 is written to the FPGA address, just before the actual code pattern starts. We include the trigger in the assembly code and align the snippet at the cache-line size (32 bytes on the ARM Cortex A9), to capture the voltage fluctuations during the execution as accurately as possible. This general framework is identical for all the test patterns, and we present an overview of the available patterns in Table 3.2. Moreover, we provide all the five code patterns excluding the trigger code for both platforms in the appendix in Appendix A.

Note that since x86 does not have explicit load/store instructions, we simply perform a byte XOR which accesses the respective memory address. To evaluate whether the side-channel has potential to replace timing-based cache side-channel attacks, we also investigate, whether cached and uncached memory access can be distinguished. When sampling traces for uncached memory access on the x86 platform, we make use of the *clflush* instruction before performing the memory accesses, whereas on ARMv7 we need to employ a cache eviction algorithm, since no cache flush is available in unprivileged mode. For the cache eviction we use the *libflush*-library, which was used in [50], and calibrate it accordingly to our specific ARM core. Cached access is simply achieved by writing to the respective memory locations before accessing them. In the results, we present how indeed cached and uncached access can be distinguished on the FPGA-SoC, enabling power-based cache attacks.

3.2.3.3. Experiments

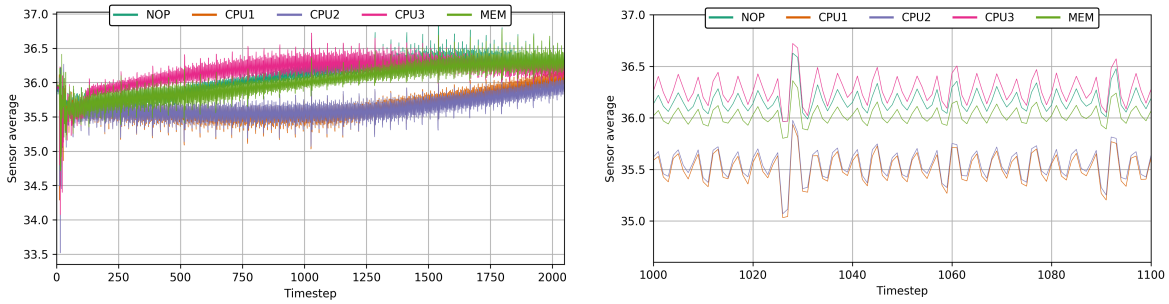
As described in the previous subsection, the FPGA implementation is designed to capture either short (2 048) or long (573 440) measurement traces after a software-controlled trigger. In this subsection, we detail how we designed our experiments to evaluate whether the FPGA-based sensor measurements are suitable for the attacker goals we enlisted in our introduction: Trigger mechanism, covert channel or disassembly.

Initially, we sample short traces of length 2 048 for the NOP, MEM, CPU1, CPU2 and CPU3 code patterns, where MEM is memory access without ensuring the access is explicitly cached or uncached. In total, we collect 100 000 traces where for each trace the executed pattern is chosen randomly from the mentioned patterns. For a first assessment we visually analyze the average over all traces for each code pattern for potential features that can be distinguished.

Then, on a subset of 50 000 traces, we extract relevant features for every trace using the methods and algorithms described in Section 3.2.2. We employ a standard five-fold cross-validation to evaluate the classification accuracy. A random forest classifier with 100 trees is trained five times on 80% (40 000) of the features vectors and we compute precision, recall and F1-score for each class as well as the overall accuracy on the remaining 20%. Afterwards, the results are averaged over the five runs. As explained in Section 3.2.2, we further reduce the amount of features to 50, using the feature importance ranking given by the tree classifier to determine the 50 most important features.

To evaluate the detection of specific code patterns within a measurement stream, we remove the trigger code from the assembly code and instead trigger the FPGA to sample a long trace with 573 440 measurements. During the sampling time, we execute a selected code pattern 5 times with a delay of $500ms$ in between. After collecting 100 long traces, we randomly select a single trace. Simulating a live detection of code patterns, which would serve as a trigger to an attacker, for example, to inject faults at the proper moment, we attempt now to find the selected code pattern within the selected long trace. For that purpose, we consider a sliding window of size s, and apply previously trained classifiers on intervals $[i \cdot s, (i+1) \cdot s)$ of the long trace. Instead of classifying each interval into one of the code patterns, we compute the probability $p_A(x)$ for each interval x, where A is the class of the selected code pattern. This probability $p_A(x)$ for interval x can now be interpreted as a measure, how likely it is for the victim to be running the code pattern A during that interval. The obvious choice for s would be the previously used 2 048 samples, but we evaluate different window sizes with different results, which are presented in the next section. We choose not to increment i one by one, as the evaluation of a single trace takes up to one day on a standard PC in that case. Instead, we increment i by $\frac{s}{4}$, where the overlap between the sliding windows is $\frac{3}{4} \cdot s$. Whereas this method is not a live detection by itself, we show that the extraction of 50 features and classification using the pre-trained random forest is sufficiently fast for an actual live detection to be implemented in the attacker software or directly on the FPGA.

Except for slight variations in the code patterns due to the different architecture, the experiments are mostly identical for both the standard PC platform and the FPGA-SoC. However, we want to detail one peculiarity regarding the experiments on the PC platform, which we mentioned in Section 3.2.3.1 already. In [19], researchers reported a higher accuracy when stressing the board-level PDN of the FPGA board. Whereas some initial experiments do not confirm a beneficial effect of stressors on the measuring FPGA board, we do observe higher accuracy, when increasing the load on the entire system-level PSU. This load is created by two components: First, we employ 15 out of the 16 CPU cores in the system to run benchmark code and second, we fill a second Xilinx Kintex-7 KC705 board with 28 000 ROs. The ROs are activated during sampling



(a) Averages over the entire traces

(b) Zoom into the range of 1 000 to 1 100

Figure 3.8.: Averages for each code pattern over 100 000 randomly sampled traces on the Zynq 7000 FPGA-SoC.

increasing the system's power consumption additionally. For classification, we are able to increase accuracy by up to 15% when inducing this additional load.

3.2.4. Results

In this section, we present the results of the previously explained experiments. We explain the results in a step-by-step approach, gradually building up the complexity of the experiments: Initially a visual analysis provides a first glance at the data, then we continue with more elaborate methods.

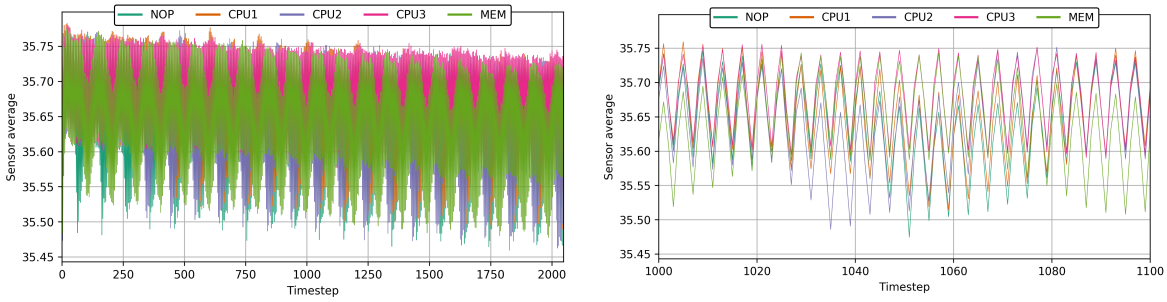
3.2.4.1. Visual Analysis

In Figure 3.8, we show the averages for each of the NOP, MEM, CPU1, CPU2 and CPU3 code patterns over 100 000 traces on the Zynq 7000 FPGA-SoC. When looking at the averages over the entire 2 048 samples in Figure 3.8a, we immediately observe significant visual differences between the averages. However, the traces for each single pattern seem to be not uniform over the sampling time either. Although we have assured to set the trigger signal as close to the evaluated code as possible – which we explain in Section 3.2.3.2 – we may get some fluctuations at the beginning of the trace, that are caused by the operating system routines after the trigger or even the FPGA trigger logic itself.

In Figure 3.8b, we show the average between sampling point 1 000 and 1 100. Even in this short timeframe of $1\mu s$, we are able to distinguish the traces of the respective patterns on average. Note that this first result does not imply that single traces can be distinguished just as easily.

On our x86 test platform, we observe significantly less difference between the averages, as shown in Figure 3.9. Note how the peak-to-peak fluctuations of the sensor values on average are only about 0.25 on the Kintex-7 FPGA accelerator, whereas they go up to around 2 on the Zynq 7000. Although we do observe some visual differences between the average traces in Figure 3.9a, they are almost invisible when looking at a shorter timeframe between 1 000 and 1 100 sampling points in Figure 3.9b.

From this first visual analysis, we can also make some conclusions when connecting the average traces back to the actual code patterns. On both platforms, we observe



(a) Averages over the entire traces

(b) Zoom into the range of 1 000 to 1 100

Figure 3.9.: Averages for each code pattern over 100 000 randomly sampled traces on the Kintex-7 FPGA, where the code is running on a Xeon CPU.

the highest sensor values (on average) for the CPU3 pattern, which corresponds to the highest supply voltage on average. The CPU3 pattern is composed of floating-point computations, as described in Section 3.2.3.2. Our assumption is that instructions with a high latency (many cycles per instruction) cause less voltage drop than high throughput integer arithmetics. Likewise, arithmetics on data from memory in the MEM code seem to cause less of a voltage drop than integer arithmetics on registers, which are performed in the CPU1 and CPU2 patterns, due to the additional access delay. At least on the ARM platform in the Zynq 7000, this is well visible in Figure 3.8b.

3.2.4.2. Classification of Short Traces

Next, we attempt to classify the collected short traces on both platforms into one of the five pattern classes. In all the presented experiments, a classifier that classifies the feature vectors of the respective traces randomly uniform within the available classes achieves an accuracy of 20%. We cross-validate a random forest classifier on 50 000 by training it five times on different splits of 80% training data and 20% test data as described in Section 3.2.3.3 and report average precision, recall, F1-score and the average of overall accuracy. The reason for considering only 50 000 traces is the computational limitations during the feature extraction and feature filtering phase, but our results show that the amount is already sufficient.

Feature extraction and filtering of traces collected on the Zynq 7000 FPGA-SoC using the TSFRESH library gives us feature vectors of size 689. The average overall classification accuracy when using the full feature vector size is 99% and all code patterns can be classified equally well. When reducing the feature vector size, using only the 50 most important features which are determined through the ranking given by the random forest classifier, we still achieve accuracy of 98%, as shown in Table 3.3.

As presented in the previous chapter, we observe some effects at the beginning of the sampling time frame, which may be caused by the aftermath of the triggering code on either CPU or FPGA. Thus, we reduce the traces to the part between 1 000 and 1 500 sampling points, to ensure we only capture the execution of a specific code pattern, which is also important for live detection of patterns within a long sampling trace. Then, we repeat the entire feature extraction and classification procedure on the

Table 3.3.: Average cross-validation results per class when classifying random traces of length 2 048 sampled on the Zynq 7000 FPGA, using a trained random forest classifier on feature vectors extracted for each trace.

(a) Results when using the full feature vectors of size 689

Name	Precision	Recall	F1
NOP	1.00	0.99	0.99
MEM	1.00	0.99	1.00
CPU1	0.97	0.97	0.97
CPU2	0.97	0.98	0.97
CPU3	1.00	1.00	1.00

(b) Results when using the reduced feature vectors of size 50

Name	Precision	Recall	F1
NOP	0.99	0.99	0.99
MEM	1.00	0.99	0.99
CPU1	0.97	0.97	0.97
CPU2	0.97	0.97	0.97
CPU3	0.99	0.99	0.99

Table 3.5.: Average cross-validation results per class when classifying random traces, reduced to 500 samples on the Zynq 7000 FPGA using a trained random forest classifier on feature vectors extracted from each trace.

(a) Results when using the full feature vectors of size 630

Name	Precision	Recall	F1
NOP	0.74	0.74	0.73
MEM	0.74	0.73	0.74
CPU1	0.89	0.87	0.88
CPU2	0.87	0.90	0.88
CPU3	0.99	0.99	0.99

(b) Results when using the reduced feature vectors of size 50

Name	Precision	Recall	F1
NOP	0.73	0.73	0.73
MEM	0.74	0.72	0.73
CPU1	0.89	0.88	0.88
CPU2	0.88	0.89	0.88
CPU3	0.99	0.99	0.99

reduced traces. The unreduced feature vectors have a size of 630 in this case and the results are presented in the first part of Table 3.5.

Even with the reduced trace, we achieve a total classification accuracy of 85% with the full-size feature vectors. Reducing the feature vectors to the 50 most important features again, the accuracy does not decrease, which is shown in the second section of Table 3.5. In this setup, we can also determine a code pattern that can be detected significantly better than all others, which is the CPU3 code pattern. It seems that floating point operations show the most distinct voltage fluctuations of the evaluated patterns. We show in the next Section 3.2.4.3, how the CPU3 pattern is also the one we can detect the easiest in a long stream of sensor measurements.

On our x86 test platform, we expect significantly lower success, as the leakage through the PSU is attenuated, compared to a shared PDN on the same chip. The general setup is identical, and we also perform a five-fold cross-validation on 50 000 traces of length 2 048. Here, we additionally stress the system, which has also been shown in [19], increasing the accuracy by around 15%.

The lower leakage can already be confirmed when looking at the size of the feature vectors after the initial feature extraction and filtering from the unreduced traces, which is only 362. Again we evaluate classification with a 100-tree random forest classifier,

Table 3.7.: Average cross-validation results per class when classifying random traces sampled on the Kintex-7 FPGA inside the PC platform using a trained random forest classifier on feature vectors extracted from each trace.

(a) Results when using the full feature vectors of size 362

Name	Precision	Recall	F1
NOP	0.47	0.45	0.46
MEM	0.56	0.63	0.59
CPU1	0.44	0.42	0.43
CPU2	0.52	0.59	0.55
CPU3	0.37	0.30	0.33

(b) Results when using the reduced feature vectors of size 50

Name	Precision	Recall	F1
NOP	0.52	0.49	0.50
MEM	0.61	0.67	0.63
CPU1	0.49	0.48	0.48
CPU2	0.56	0.63	0.59
CPU3	0.42	0.36	0.39

the results of which are presented in the first section of Table 3.7.

We observe a significantly decreased overall classification accuracy of only 48% on the x86 platform, as expected from the higher dampening through the system PSU. For comparison, a classifier that classifies the feature vectors of the respective traces randomly uniform within the available classes achieves an accuracy of 20%. Thus, albeit the accuracy is much less than on the FPGA-SoC, we can conclude that generally, the classification of code patterns using FPGA-based is possible, even through the PSU in a PC desktop/server system.

We note that on the x86 platform, the classification accuracy depends a lot on the code pattern, unlike on the Zynq FPGA-SoC, where – at least when using full-size feature vectors – the classification works equally well for all patterns. On x86, the MEM code pattern, which performs byte-wise memory access with a simple XOR instruction (*xorb*), can be detected the easiest, followed by the CPU2 pattern which consists of integer arithmetics. Just like on the ARM core, we evaluate classification with a reduced feature subset of the 50 most important features, according to the ranking provided by the initial tree classifier. These results are detailed in the second part of Table 3.7.

Interestingly, on the x86 platform we achieve a slightly higher accuracy of 52% when reducing the feature vector size, indicating a possible overfitting of the classifier when trained on the full feature vectors. However, without presenting the additional results here, we find that a reduction to only the 20 most important features decreases accuracy again. Moreover, we omit the detailed results for the PC platform when reducing the traces to the interval $[1\,000, 1\,500)$ and just report the accuracy, which goes down to only 27%. Here, shortening the traces may be unnecessary, as the trace averages on this platform are more homogeneous (see the visual analysis in the previous subsection).

Lastly, in this subsection, we compare the actual features, which are selected by the feature extractor and the classifiers as the most important for classification. This comparison provides meaningful insight, in terms of what exactly is affected by the voltage fluctuations caused by the CPU and how they are perceived by the sensor on the FPGA. We enlist the top five features for each platform, according to the ranking provided by the respective tree-based classifier. On one hand, the following features

were ranked most important on the Zynq 7000 FPGA-SoC:

1. Trace mean value
2. Highest coefficient of a maximum-likelihood estimated autoregressive (AR) model
3. Autocorrelation with lag 6 of the trace with itself
4. CID complexity of the trace [51]
5. Real part of the first Fourier coefficient of the discrete Fourier transform of the trace

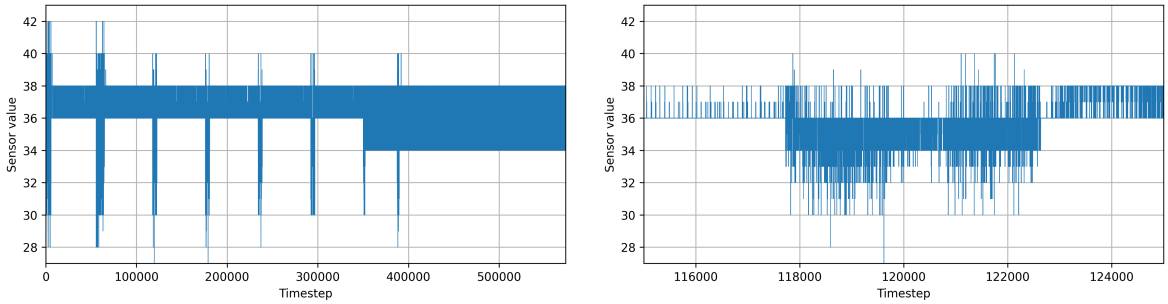
On the other hand, classifiers on the x86 PC platform derive the following top five features for classification:

1. Imaginary part of the 21st Fourier coefficient
2. Absolute of the 21st Fourier coefficient
3. Real part of the 83rd Fourier coefficient
4. Angle of the 21st Fourier coefficient
5. Imaginary part of the 42nd Fourier coefficient

As a matter of fact, the entirety of the 50 most important trace features for classification on the PC platform are Fourier coefficients. Thus, we conclude that the voltage fluctuations on the PC power supply caused by execution on the main CPU do not actually impact the absolute sensor value directly, but rather modulate the very small fluctuations of the sensor value in the frequency domain. This knowledge may help in developing more accurate sensing and classification mechanisms in the future. On the FPGA-SoC, however, the trace average is ranked the most important feature for classification. There, we have a direct impact on the absolute sensor value from the voltage fluctuations caused by the CPU, which also concurs with our visual observations in the previous subsection.

3.2.4.3. Live Detection in Long Traces

The results from the previous sections show information leakage from CPU to FPGA in two platforms, when synchronized windows of 2 048 measurements are analyzed. However, like in many other attacks based on FPGA-internal sensors, synchronized measurements can be hard to achieve in a system with strong isolation between attacker and victim. In this subsection, we show how patterns can be detected in a live stream of measurements on the FPGA-SoC. As the classification accuracy on the x86 platform in the previous subsection is rather low compared to what can be achieved on the Zynq 7000, we perform our live detection experiments on that platform only. We attempt to apply a sliding-window approach on a longer trace of 573 440 measurements, where selected patterns are executed multiple times during the sampling period. For that purpose, we execute selected patterns 5 times with a delay of $500ms$ during the



(a) Entire trace of length 573 400

(b) Zoom into the range of 115 000 to 125 000

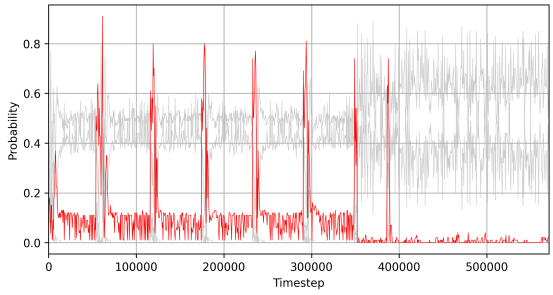
Figure 3.10.: A single long measurement trace, captured while executing the CPU1 pattern five times with a delay of 500ms on the Xilinx Zynq 7000 FPGA-SoC.

sampling time. The trigger is set at the very beginning. Before applying the sliding window and classifier, we again analyze the raw traces visually.

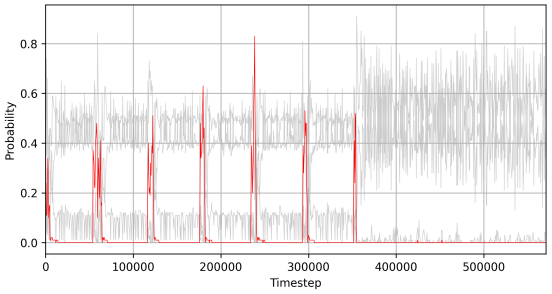
In Figure 3.10 we exemplarily show a single long measurement trace, which is recorded while executing the CPU1 code pattern on the ARM core of the Xilinx Zynq 7000 FPGA-SoC. The five fluctuation peaks caused by the code execution are clearly visible in Figure 3.10a, whereas we show the part of the trace during the second execution of the code in Figure 3.10b. Note that peaks at the beginning and the end of the code execution are visible as well, possibly caused by operating system routines or trigger aftermath. The increased noise in Figure 3.10a after around 350 000 samples stems from the software process actively polling a *done* signal from the FPGA, which indicates the completion of the measurements.

As explained in Section 3.2.3.3, we apply a sliding window approach on the long trace, calculating for each window x the probabilities $p_A(x)$ using the previously trained random forest classifiers. For all the CPU1, CPU2, CPU3 and MEM code patterns, we collect traces where the respective pattern has been executed during the sampling time five times, the same way as presented for CPU1. Then, we compute all probabilities $p_A(x)$ for each step $(\frac{s}{4})$, that the respective interval of that trace belongs to class A. The results when using a window size of 2 048, are presented in Figure 3.11. In each diagram, the classification probability with the correct class – i.e. the class of the pattern that has been executed five times while sampling the long trace – is colored red, whereas the probabilities of the other classes are colored grey.

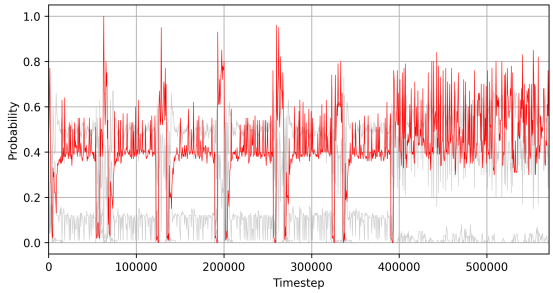
We observe obvious peaks for the time periods that correspond to the respective code pattern. However, not all the code patterns can be detected equally well: Whereas the five CPU1 and CPU3 executions are clearly visible, the classifier fails to identify CPU2 and MEM within the measurement stream correctly. After around 350 000 samples, the CPU is waiting for the FPGA to finish sampling the trace. This is done by actively polling a *done* signal from the FPGA. During that time period, the classifier is unable to classify the trace intervals into any of the classes, which can be problematic for the attacker. In Figure 3.11c, the classifier yields incorrect detections for the polling period, which would be detected as executions of the CPU3 pattern. However, an additional threshold of 0.8 on the classification probability, would lead to a successful detection of four out of the five executions of the CPU3 pattern in our case. Improving



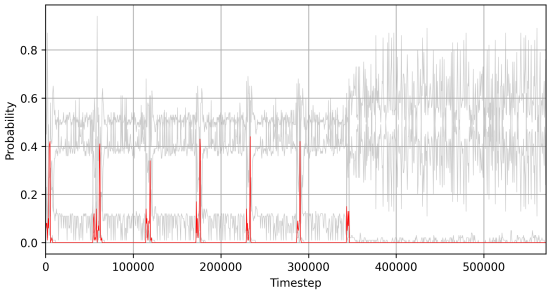
(a) Probabilities $p_{\mathrm{A}}(x)$ over a trace sampled during execution of the CPU1 pattern with a window size $s = 2\,048$



(b) Probabilities $p_{\mathrm{A}}(x)$ over a trace sampled during execution of the CPU2 pattern with a window size $s = 2\,048$

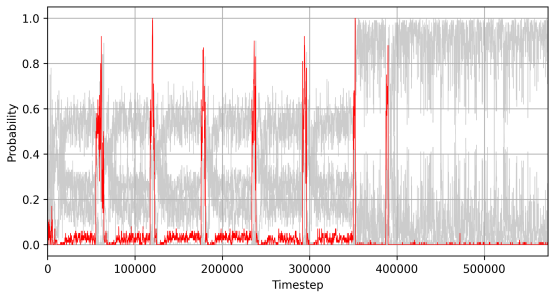


(c) Probabilities $p_{\mathrm{A}}(x)$ over a trace sampled during execution of the CPU3 pattern with a window size $s = 2\,048$

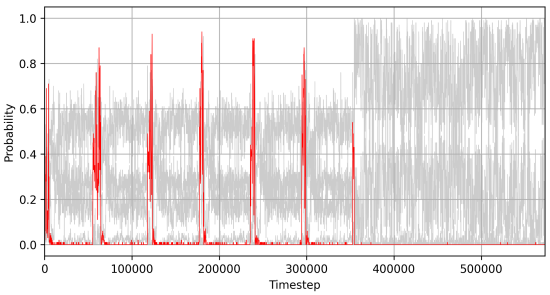


(d) Probabilities $p_{\mathrm{A}}(x)$ over a trace sampled during execution of the MEM pattern with a window size $s = 2\,048$

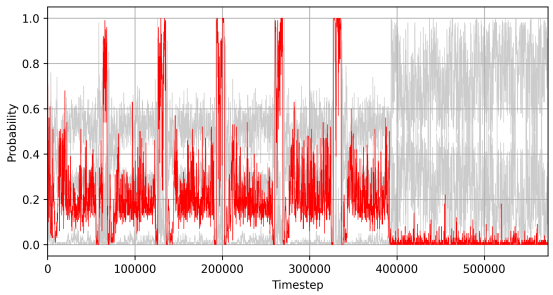
Figure 3.11.: Classification probabilities $p_A(x)$ as given by the random forest classifier for a sliding window x of size $s = 2\,048$ to belong to the respective class A. The probability for the interval to be classified as the correct class, i.e. the class of the pattern that is executed during the measurements, is marked red.



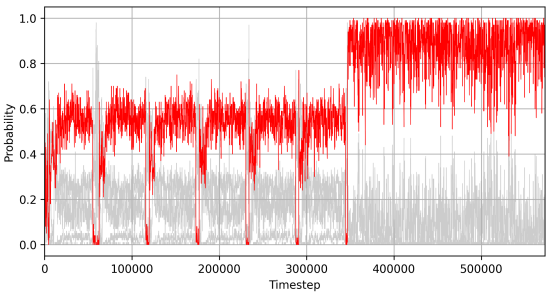
(a) Probabilities $p_{\mathrm{A}}(x)$ over a trace sampled during execution of the CPU1 pattern with a window size $s = 500$



(b) Probabilities $p_{\mathrm{A}}(x)$ over a trace sampled during execution of the CPU2 pattern with a window size $s = 500$



(c) Probabilities $p_{\mathrm{A}}(x)$ over a trace sampled during execution of the CPU3 pattern with a window size $s = 500$



(d) Probabilities $p_{\mathrm{A}}(x)$ over a trace sampled during execution of the MEM pattern with a window size $s = 500$

Figure 3.12.: Classification probabilities $p_A(x)$ as given by the random forest classifier for a sliding window x of size $s = 500$ to belong to the respective class A. The probability for the interval to be classified as the correct class, i.e. the class of the pattern that is executed during the measurements, is marked red.

the detection through further means, for example, a ramp detection or evaluation of the probabilities over a longer time period may be necessary, depending on the specific attack.

In Figure 3.12, we present the results of applying a sliding window of size 500 on the traces. Here, the classifier, which is applied on the sliding window of size 500, is trained on shortened traces, that have been reduced to the interval between 1 000 and 1 500 sampling points, as shown in the previous Section 3.2.4.2.

When using a window size of 500, the probabilities change drastically for some patterns. CPU1, CPU2 as well as the CPU3 code can now be detected easily, whereas the MEM pattern is now detected during the entire waiting time at the end. The latter, however, is well expected, as polling a *done* signal is – at least from a software perspective – no different from a memory access.

All in all, we conclude that code patterns in principle can be detected from a measurement stream via FPGA-internal sensors on the Xilinx Zynq 7000 FPGA-SoC. This live detection enables an attacker to use the FPGA-internal sensors for triggering more elaborate attacks, such as fault injection or side-channel attacks, which require very accurately aligned measurements. However, the accuracy depends on the pattern selection and many meta-parameters, of which we only investigate the sliding window

size. Regarding the window size, we find that some patterns can be detected more accurately with a larger window size, whereas others require a more narrow window. In a real attack, we would expect a much more complex pattern, such as the beginning of an encryption, which may also be distinguished more easily from fluctuations caused by operating system routines, context switching, scheduling and many other aspects that are not controlled by the attacker. We also note that in many attacks, such as side-channel attacks or fault injection, a 100% accuracy for triggering the attack is usually not required. Although our choice of classifier works very well for classifying different patterns as shown in the previous subsection, there may be better choices for the binary detection that is required in this experiment. Other researchers made use of neural networks for classification [47], which would be easy to implement entirely on the FPGA itself.

3.2.4.4. Distinguishing Memory Accesses

Finally, we investigate whether cached memory access can be distinguished from uncached access, which would be a valuable asset for attackers that are hindered from deploying timing-based cache attacks. For starters, we remark that we are unable to achieve any success on the x86 platform, which is why we omit results on that test system entirely. On the Xilinx Zynq 7000, we use a cache eviction library [50] to ensure uncached access and store a random value at the tested memory location, to make sure the data is in the cache. Whereas the MEM code pattern presented in the previous subsections performs continuous memory access, here, we want to assess whether a single memory access can be distinguished, to derive realistic cache attacks. Therefore, we use the ARMv7 memory fencing instruction *dsb* to make sure all previously scheduled memory accesses have completed before performing a single load (*ldr*) instruction from the cached or uncached memory location.

Before performing feature extraction or using classifiers, we again analyze the trace averages for cached and uncached access visually. The averages over 100 000 randomly collected traces for cached (MEMC) and uncached (MEMNC) access, are presented in Figure 3.13.

We see that cached memory access results in a lower sensor value (i.e. lower supply voltage) on average. This concurs with our results on the different code patterns, where more instructions per time unit also resulted in a lower sensor average. For the uncached memory access, the CPU has to wait a longer time for the access to complete, leading to less voltage drop from subsequent computations. In our case, subsequent computations would be the CPU polling the *done* signal from the FPGA. By executing more power-intensive instructions, such as integer arithmetics on registers, which require very few cycles per instruction, the effect may be even increased. Potentially an attacker might use the information for indirectly measuring the memory timing through the voltage trace and differentiate accesses even further. For instance, the information could be used to distinguish a memory load from L1 cache from a load where the data is fetched from L2 cache.

For classification of single traces sampled during cached and uncached memory loads, we again extract and filter features from 50 000 traces and evaluate a random forest classifier using a five-fold cross-validation. Here, we directly reduce the feature vector

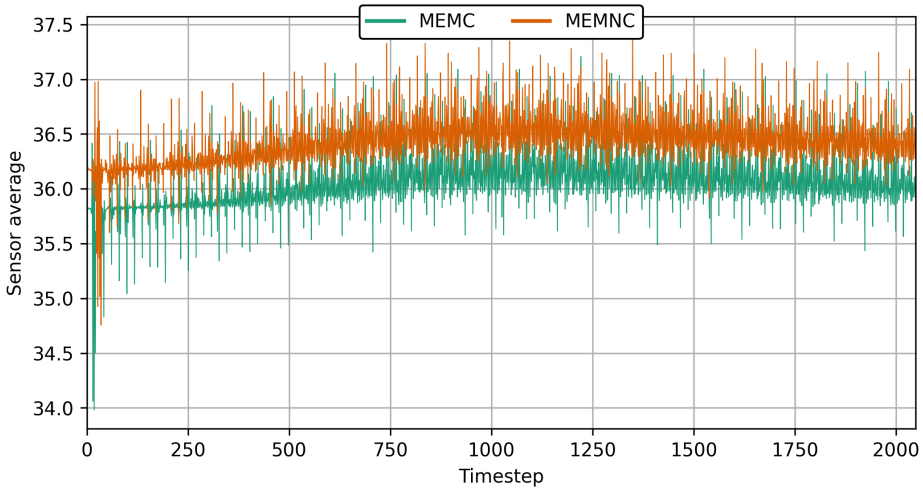


Figure 3.13.: Averages over 100 000 randomly collected traces of length 2, 048, where either a single cached (MEMC) or a single uncached (MEMNC) memory load instruction is performed during the sampling time on the Xilinx Zynq 7000 FPGA-SoC.

Table 3.9.: Average cross-validation results per class when classifying random traces sampled on the Zynq 7000 FPGA-SoC during cached and uncached memory accesses using a trained random forest classifier on reduced feature vectors of size 50 for each trace.

Name	Precision	Recall	F1
MEMC	0.98	0.99	0.99
MEMNC	0.99	0.98	0.98

size to 50 as in the previously presented experiments. In Table 3.9, we show how well the classifier is able to distinguish feature vectors of size 50 with an overall accuracy of 98%.

Being able to classify memory accesses according to whether the data is cached or not, enables many attacks that make use of the data cache as a side-channel. Simple countermeasures against cache attacks are often based around denying access to accurate timing measurement [52]. With access to programmable logic, however, an attacker can bypass such countermeasures by indirectly measuring timing through the voltage fluctuations caused by subsequent computations after the memory read. Even artificial randomization of the execution time may not be an effective defense mechanism, if the attacker can detect the memory access within a longer measurement trace.

3.2.5. Discussion

Our results show, how FPGA-internal sensors can be used to derive the code that is running on a CPU, when both of them are sharing the power supply on a certain level. In this section, we discuss the consequences and conclusions that result from our

experiments. Moreover, we provide perspectives for future work and possible countermeasures.

In the introduction, we presented possible attack vectors, where the classification of code through a side-channel can be useful to an attacker. As mentioned, this concept is by no means a novelty and many researchers looked into reverse engineering approaches based on power or electromagnetic measurements. However, so far the attacker has always been required to be physically present and have physical access to the device with measurement equipment. Our work proves, that within shared power domains and through unintended usage of emerging accelerators, such as FPGAs, this threat model can be escalated to a remote attacker. Especially in cloud computing, where FPGAs have arrived in the last years, this threat should be taken serious, as attacks can be scaled to a very large amount of devices with catastrophic consequences.

We mainly showed successful results on an FPGA-SoC, where the CPU and the FPGA are present in a single chip. Naturally, the information leakage in this scenario is much higher due to the power supply being shared at a very low level. Nevertheless, our classification results on an unmodified x86 desktop system running Linux prove that the threat is present even when the leakage source is much further from the attacker sensor in the power supply hierarchy. We evaluated a single system with only one classification method, where future works will be able to confirm the results and potentially improve accuracy on different systems or more advanced methods. Furthermore, a tight integration of FPGA and main CPU is by no means an exotic concept, as new developments for server CPUs show [53]. Last but not least, a custom programmable logic accelerator is only one of many devices that can be accessed by an attacker in a modern server system. It is not far-fetched, that a sophisticated attacker may yet discover other means of measuring supply voltage through unintended usage of other devices than FPGAs. In [17, 18], for instance, the Analog-to-Digital Converter (ADC) of a mixed-signal IoT device is used to recover secret AES keys.

The random forest classifier, which we trained on prerecorded short traces and later applied in a sliding window approach for live detection, achieved high accuracy for classifying but may be replaced by other more suitable methods. For a practical live detection, not only the accuracy, but also how fast a stream of measurements can be evaluated, is critical to the success. The extraction of 50 features from a trace of length 2 048 and a subsequent classification takes about $250ms$ on our evaluation system, which is based on an AMD Ryzen 5 3600 hexa-core CPU. With a sampling rate of 100 MHz, this time period would correspond to about 12 207 traces which are lost during classification. Even with a reduced window size of 500 samples, the entire classification process takes about $14ms$, which corresponds to around 2 800 lost traces. Although this may still be useful as a trigger mechanism for victim code that runs for a similar amount of time, an obvious improvement would be for the attacker to implement the detection directly on the FPGA itself, to achieve a much higher throughput. In [47], a neural network is used for classification, which would be well suited for an implementation on FPGAs, which are often directly advertised as accelerators for AI applications.

Being able to distinguish cached and uncached memory access enables many attacks that formerly relied on measuring the access timing. Recent attacks that are based on microarchitectural data sampling, such as the Spectre and Meltdown attacks [54, 55], rely on cache timing side-channels to leak information about data at restricted memory

locations. On one hand, this requires the attacker to be able to flush the cache or evict data from the cache, which can already be problematic on ARM platforms [56]. On the other hand, the attacker needs to be able to perform accurate timing measurements. This measurement can be impeded for multiple reasons, for instance, when cycle counters are not available from userspace. Moreover, accurate timing measurements can be explicitly restricted as a very simple countermeasure against cache attacks [52]. All of these obstacles can be avoided when employing FPGA-internal sensors to distinguish the memory accesses, which we proved to be feasible in this work.

Regarding countermeasures, the main issue, that enables attacks through FPGA-internal sensors, is the shared power supply. In our work, we see how the leakage occurs even on much higher levels than the PDN of a single or board, namely through the PSU of a standard workstation. This makes addressing the issue on a hardware or architectural level difficult. Nevertheless, the power supply and chip-level PDN have not been designed with security in mind, which needs to be a concern especially for future cloud hardware. A possible mitigation in the meantime restricts the attacker's freedom on the device, which is what we suggest in Chapter 5. In the proposed approach, FPGA bitstreams are checked for malicious designs, resulting in an FPGA antivirus, applied by the hypervisor before the bitstream is downloaded. More extreme solutions would restrict the attacker design to only certified accelerators, provided by the hypervisor. Both approaches require the user to trust the hypervisor, as bitstream encryption would not allow checking for potential misuse.

Another possibility is for the victim to preemptively apply countermeasures within their own code. Classical approaches such as masking and hiding can obviously mitigate an FPGA-based attacker as well. However, as a first step, designers must be made aware of this attack vector.

Our results point out important and novel aspects of the security issues that come with the integration of reconfigurable accelerators. Future works will continue to improve the practicality of such attacks and the development of countermeasures should be addressed urgently.

3.2.6. Conclusion

As FPGAs are getting more widely used, their potential security implications need to be analyzed in more details. In this section, we show how FPGA-based voltage sensors can be used to classify and detect code patterns running on a CPU, when the devices are sharing a power supply network at some level. Not only on an FPGA-SoC, where the power supply is shared between CPU and FPGA on the chip level, but even inside a desktop/server system with a shared ATX PSU, classification of code via voltage traces is possible. On the Xilinx Zynq 7000 platform, we achieve a classification accuracy up to 99% using feature extraction and a random forest classifier, whereas on an x86 computer with a PCIe-attached Kintex-7 KC705 accelerator an accuracy of 52% can be achieved, which is significantly higher than the 20% accuracy achieved by a random uniform classifier. Moreover, we show how on the FPGA-SoC we can even extend the attack to perform a live detection of code patterns inside a measurement stream using a sliding window approach. Last but not least, even a single cached access to data in memory can be distinguished from a load instruction that needs to fetch the data from

main memory on the Zynq 7000 with 98% accuracy. The results prove that trigger detection, covert-channels, cache attacks and even reverse engineering on a larger scale is possible for a remote attacker through FPGA internal sensors. Especially for cloud computing and other virtualized environments, this issue should be addressed urgently in future works and designers of security critical implementations must be aware of the threat.

4. CPAmap: On the Complexity of Secure FPGA Virtualization, Multi-Tenancy, and Physical Design

This work on the impact of physical design parameters on side-channel vulnerability has been published in [21] together with Dennis Gnad and Mehdi Tahoori.

Whereas remote fault attacks on multi-tenant FPGAs can at least be detected on both hypervisor and victim side, side-channel attacks impose a more challenging problem. Classical countermeasures against external attackers with traditional measurement equipment can be categorized as hiding [57, 58] or masking [59]. These well explored mitigation strategies can be applied to a multi-tenant FPGA scenario as well as FPGA-specific countermeasures based on reconfiguration [60].

The impact of physical design space on the SCA vulnerability has been investigated in previous works [59, 61–65]. Especially hiding countermeasures for power equalization struggle with Process Variation (PV) and hardware asymmetry, reducing the effectiveness of such countermeasures [66]. In the new threat model of multi-tenant FPGAs, both the victim design, such as a cryptomodule, and attacker design, such as the sensors we explain in Section 2.6, are subject to PV. Moreover, it is yet to be explored, how the sensitivity to voltage fluctuations and the ability to generate them depends on the asymmetric PDN design across the FPGA.

To close this gap, we provide a systematic analysis of the dependencies between physical design parameters and side-channel vulnerability in the context of internal attacks on FPGAs. This is critical for understanding the security vulnerabilities and challenges of multi-tenant FPGA virtualization.

In the following, we briefly summarize reasons why the attack scenario for internal on-chip attacks differs from that of a classical side-channel attacker. On one hand, physical access and external measurement equipment improves the attacker's capabilities:

- The sampling rate and precision of the measurements depend on the quality of the measurement equipment, which is usually only limited by the attacker's financial power.
- The attacker's influence on the victim design is minimal.
- Any side-channels such as power, EM or photon emission can be exploited externally.
- The measurement data corresponds to the actual observables, such as the actual supply voltage, whereas internal measurements can only give estimates.

On the other hand, an internal attacker can exploit the following circumstances:

- Sensors are directly connected to the chip-level PDN and not hindered by board-level decoupling capacitors or noise from other components.
- The internal attacker can acquire localized information, by placing multiple sensors in different regions of the FPGA.

Considering those differences between an attacker with physical access and an FPGA-internal attacker, we believe that a thorough evaluation of the internal attack capabilities is a necessity for the further development of effective countermeasures for this specific scenario, which is the aim of this contribution. More specifically, we investigate the interaction between logically isolated partitions and analyze the success rate of power analysis attacks in terms of the required amount of measurements for key recovery w.r.t. the following physical design and mapping parameters of attacker and victim modules:

- Global module placement, which includes intra-chip PV and the PDN asymmetry across the chip
- Local primitive placement within partitions
- Inter-chip PV, analyzing different boards of the same type
- Heuristic Place-and-Route algorithms, through recompilation of bitstreams

Identical switching on the logical level causes different voltage noise, depending on the physical mapping, which has been explored for internal measurements on FPGAs as well [11, 14]. This dependency is due to the PV [67–69] and the runtime variations in the PDN [11, 14] leading to differences in both how switching activity influences the supply voltage and how the impact is observed in specific locations.

Our results on Xilinx Zynq XC7Z020 FPGAs show that the success of the attack is very much dependent on the above parameters, with the amount of traces required varying between a few hundred measurements and entirely unsuccessful attacks with up to $100k$ traces. We confirm similar findings on a larger FPGA PCIe accelerator card based on the Xilinx Virtex-7 XC7VX690T-2 FPGA with up to $10M$ traces, by performing experiments on a subset of parameters. The analysis implies that these physical design parameters are just as critical to the design's side-channel attack resistance as actual countermeasures. The increase in the amount of traces required is within the range of what some actual simple side-channel countermeasures are able to achieve [25, 58, 70].

To verify the importance of physical design parameters for side-channel countermeasures, we compare the side-channel vulnerability of a module protected by a hiding scheme based on a power noise generator. In this setup, we demonstrate that not only the vulnerability of an unprotected design, but also the effectiveness of countermeasures highly depends on the physical design and mapping parameters.

In summary, our results expose the very complex dependencies between physical design and side-channel vulnerability on a multi-tenant FPGA through a systematic

analysis. This research can also lay a foundation for effective countermeasures, possibly with zero overhead, based on specific restrictions for local and global placement of trusted and untrusted modules as well as routing constraints.

In the next section we explain how traditional, external side-channel attacks are mitigated as well as some novel existing countermeasures specific to the multi-tenant FPGA scenario. Then, in Section 4.2 we explain the theoretical background behind our evaluation methods. In Section 4.3 we detail our experiments and the hardware that has been used. Section 4.4 presents the results of evaluating power analysis attacks w.r.t the mentioned parameters, which are later discussed in Section 4.5. Finally, we draw some conclusions in Section 4.6.

4.1. Existing Countermeasures against Power Analysis Attacks

Researchers have proposed defenses for the specific scenario of a virtualized FPGA, but we also want to briefly mention the major categories of classical SCA countermeasures. Considering the secret data-dependent information within the measurement trace as the signal which the attacker wants to acquire, most countermeasures aim to decrease the SNR. This can be done by generating noise [71] or equalizing the data-dependent power consumption of computations [57], thus *hiding* the sensitive information. Alternatively, secret data can be masked with randomized data on the algorithmic level, which classifies as *masking* [59]. Hiding schemes often make use of Dual-Rail Precharge (DRP) logic to achieve power equalization [66] through duplication of the original circuit. In the context of DRP countermeasures on FPGAs, the high impact of placement and routing variations is already well established albeit only with external attacks and the use of measurement equipment in mind. The vulnerability of DRP schemes against remote internal attacks is yet to be explored.

Specifically on FPGAs, researchers proposed to make use of Partial Reconfiguration (PR) to mitigate side-channel attacks through temporal jitter [60] or, for example, interleaving implementations of an AES S-Box [72]. In the context of multi-tenant FPGAs in the cloud, offline scanning of user bitstreams has been proposed as a method to detect and block fault and side-channel attacks already on a hypervisor level, before the bitstream is being loaded to the FPGA [11, 24, 25]. This bitstream checking approach, is also a contribution of this thesis and presented in Chapter 5. The success of offline detection, however, depends highly on the provided signatures and may require the user to give up on bitstream confidentiality towards a possibly untrusted hypervisor.

A hiding countermeasure based on a so-called *Active Fence* against on-chip SCA is also one of the contributions of this thesis and explained in detail in Section 6.1. The basic principle of that countermeasure is to equalize power consumption using a fence for compensating voltage fluctuations between attacker and victim modules. This fence is implemented as a row-by-row RO array, with the row activation depending on either a sensor value for power equalization or a Pseudo-Random Number Generator (PRNG) for noise increase. This countermeasure is in line with previous works on randomly activated ROs for noise generation [70] and will serve as an example for

a generic hiding countermeasure to evaluate the effect of physical design parameters on protected designs in our work. We deploy RO arrays around the AES modules which are randomly activated. The noise increase makes the attack more difficult and increases the minimum amount of traces required. A more detailed description of this experiment is provided in Section 4.2.4.

Although we do not propose a new countermeasure in this chapter, we highlight the sensitivity of SCA attacks to various placement parameters for both trusted (victim) and untrusted (attacker) modules, which will be important for further development against side-channel leakage in the multi-tenant FPGA threat model. Our results show that noise generation as a hiding countermeasure can have as much impact as the exploration of the physical design space.

4.2. Theoretical Background and Methodology

For readers to understand our assessment methodology, we provide some necessary theoretical background information in this section. Whereas the basic principle of FPGA-internal voltage sensors has been explained in Section 2.6, we use an automatically calibrated sensor variant in this work, which we detail in the next subsection. Afterwards, we briefly explain, how we utilize the CPA attack on the AES to evaluate attack success in terms of traces required for key recovery. In general, we perform two kinds of experiments: First we evaluate the general impact of a noise generation module based on ROs or toggling Flip Flops (FFs) on TDC sensors, then we assess the actual CPA attack success on AES modules. This approach allows us to analyze the general dependencies of module placement in the FPGA and SCA success before investigating the complexity of the actual attack w.r.t. all physical design parameters. In Section 4.4, we see how a simple correlation between the global placement and SCA success can not be established, but we still can infer a relation between the results from both types of experiments. Moreover, we show how a simple noise-generation countermeasure performs under varying conditions to verify the importance of a thorough analysis for mitigating SCA attacks. This exemplary countermeasure is described in detail in the last subsection of this section.

4.2.1. Self-calibrating FPGA-internal Voltage Sensors

To perform estimated measurements of chip-internal supply voltage, we use sensors based on TDCs, as explained in Section 2.6. These sensors are sensitive to process, voltage, and temperature (PVT) variations [11, 73], with voltage having the most influence during runtime.

It is usually not possible to know how long the path is supposed to be at design time, since both intra-die and inter-die manufacturing process variation can be significant. In previous works, the length of the total sensor needed to be adjusted in order to account for process variations, or operating frequency [12]. We will show a design here which allows calibration at runtime, such that the same bitstream works on multiple boards, and can also be recalibrated to account for temperature changes. Here we specifically show how they are implemented in Xilinx slices, which is similar from at least their

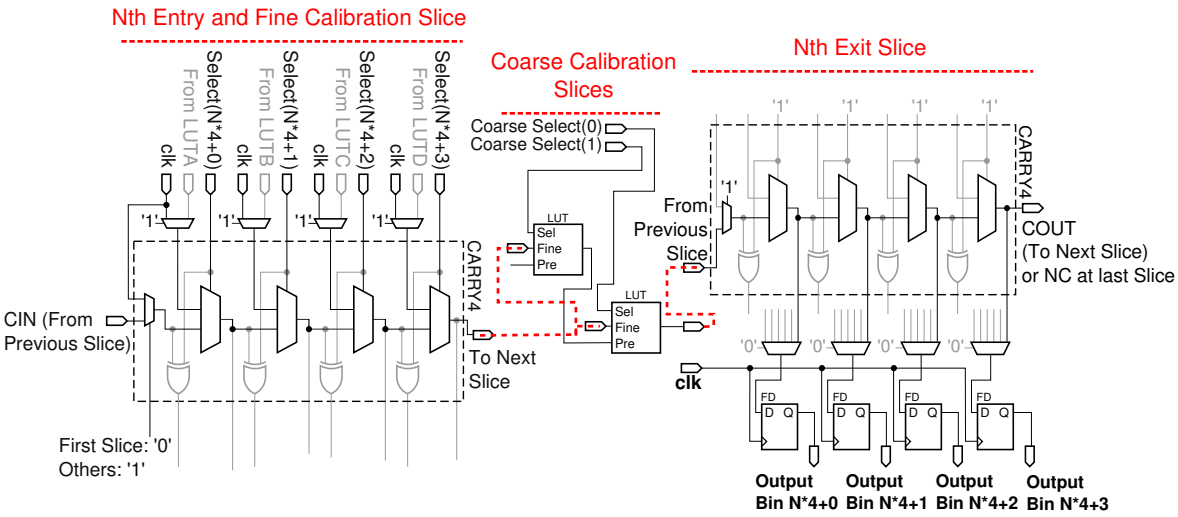


Figure 4.1.: Principle of a delay line based on Xilinx LUT, FD, and CARRY4 elements [74]. In *Fine Calibration*, selectable entry points of the clock *clk* are shown to allow for small adjustments of the total path length. The center slices use the output from the Fine Calibration as the input to *Coarse Calibration* slices based on LUTs. In *Exit* slices, multiple bins of the delay line are routed to FF (FD) primitives as the output values.

5th Generation to Ultrascale+ line. The design is made out of three types of FPGA slices, that we show together in Figure 4.1.

In *Fine Calibration* slices, we allow the clock 'clk' to enter a Xilinx CARRY4 primitive (c.f. [74]) at various depths of the delay line, by connecting it to all possible inputs that can be selected with multiplexers. Since the clock tree is balanced before the clock is connected to the CARRY4, the selection scheme allows for varying the path length in small steps. CARRY4 elements are used since they are known to allow for the smallest delays, which are about $12ps$ for one bit in a Xilinx 6-series FPGA [11]. Similar primitives are available in FPGAs of other vendors.

Coarse Calibration slices are fed with the output of *Fine Calibration* slices. In these slices, LUTs are configured in a similar cascaded way as the CARRY4 elements in Fine Calibration, but instead of the clock, they all get the output from fine calibration. Please note that this can lead to some non-linearities since the signal does not anymore benefit from the balanced clock tree, but needs to use standard FPGA interconnect resources.

Exit slices follow the design of related work [73], in which CARRY4 primitives are used again for their small delays. However, here we add FFs between the multiplexers, which can then signal how far the clock could propagate through the path.

In the end, sufficient fine and coarse calibration slices need to be implemented, such that the first FF in the exit slices does fail timing requirements significantly, since typically high timing safety margins are applied in the timing analysis. The output of the sensor can sometimes show a zero in between a row of 1's, since the process variations between the FFs can exceed the delay of one bit. Thus, so-called bubble detection has to be applied, which essentially switches the position of the highest '1'

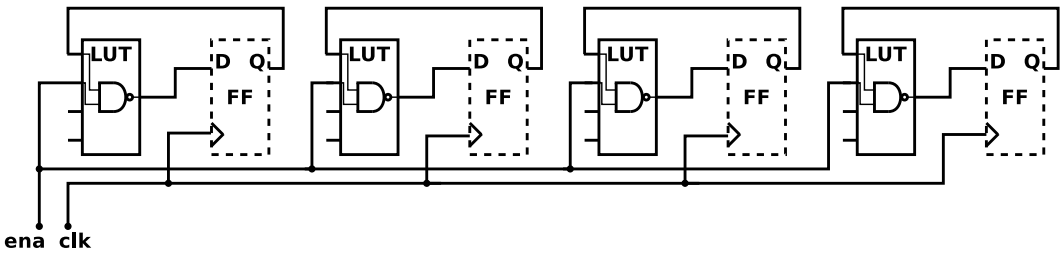


Figure 4.2.: Generic schematic of an artificial noise generation grid based on ROs or toggling FFs

with the '0' that follows after it [75]. During runtime, the sensor can be automatically calibrated by a state machine that checks the sensor output and adjusts coarse and fine calibration stages accordingly. The target is usually that the sensor is in the middle of the possible output range (e.g. 0–63), to be able to show both negative and positive voltage differences.

Please note that in [73] it was suggested to use a phase-shifted clock at either entry or exit, to adjust the effective path length instead of the presented approach with calibration slices. We have also experimented with that, but got significant noise at idle. We think that noise comes from having two separated clock trees for the two clocks, and these clock trees getting affected with a different jitter.

4.2.2. Investigating the Impact of Global Module Placement

Initially, we want to evaluate the impact of switching voltage noise caused by modules placed in different regions of the FPGA, which is modulated on the PDN and observed by sensors in different locations. Thus, we employ variants of generic voltage noise generators using ROs or toggling FFs. ROs are implemented using a single inverter LUT with a feedback loop. To evaluate the effect of toggling FFs, we simply insert a register into the feedback loop, which is clocked by an arbitrary PLL on the FPGA. The ROs or FFs are deployed as synchronously enabled grids on the FPGA roughly the size a regular AES module would occupy. A similar analysis has been done in [11], which already showed some spatial dependencies of sensor placement and switching activity, but without side-channel attacks in mind.

We outline the generic principle of our noise generation modules in Figure 4.2. These noise generators are intended to model the switching activity caused by the AES module, while eliminating the influence of local primitive placement. The amount of the switching activity as well as the local placement of all FPGA primitives within the noise generating modules is exactly the same for all modules. In addition, the local primitive placement of the self-calibrating TDC sensors is always identical. Therefore, any differences observed in the generated or sensed voltage noise can only result from inter- and intra-chip PV as well as variations in the PDN structure.

The impact of these noise generation modules and the sensitivity of sensors can now be analyzed by comparing sensor values with and without activated noise module. We compare both average and variance of sensor values during the respective period, which corresponds to 512 sensor samples each. In Figure 4.3 we show the impact of the FF

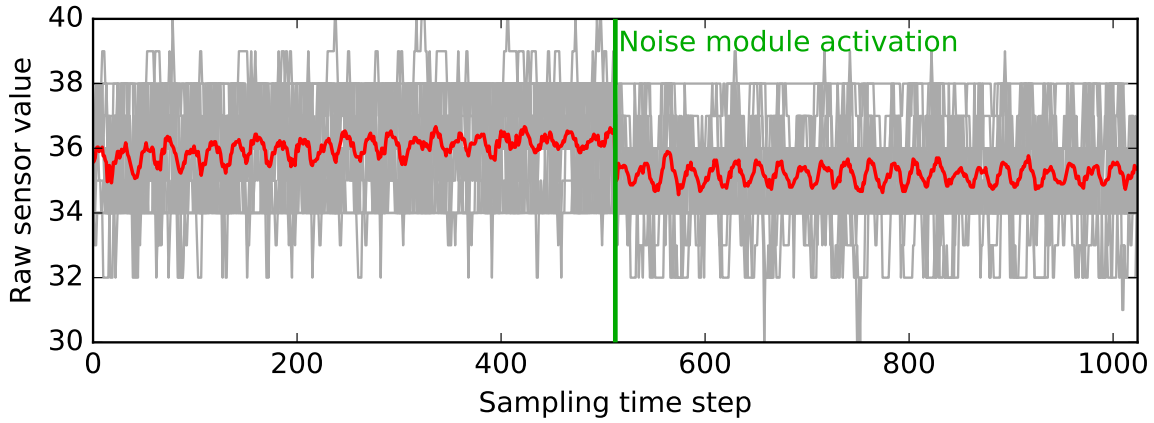


Figure 4.3.: Raw traces (grey) and average (red) over 30 different sensors for 1 024 sample points with noise module activation after 512 samples

based noise generation grid on 30 different sensors (grey) and the average of all 30 sensors (red) on our experimental platform. A significant drop in the average sensor value from $\mu_{[0,512)}$ to $\mu_{[512,1\,024)}$ is clearly visible after the FF toggling is activated at 512 samples. Albeit not clearly visible in the figure, a variance increase after toggle activation can be measured as well. We define a measure of impact or sensitivity (depending on the perspective of either noise generators or sensors) as the absolute of the subtraction $\delta^{\mu} = |\mu_{[0,512)} - \mu_{[512,1\,024)}|$ for the average or $\delta^{\sigma} = |\sigma_{[0,512)} - \sigma_{[512,1\,024)}|$ for variance respectively.

4.2.3. Investigating Side-Channel Vulnerability through CPA Attacks on AES

To evaluate the impact of physical design parameters on the side-channel vulnerability, we employ a classical CPA attack [38] as explained in Section 2.3. We assess the attack success, by determining the minimum amount of traces required for recovering the first key byte of the last AES round key. If an attacker is able to recover only a single key byte or even 50% of the secret AES round key, the remaining key space is of course still too big for an exhaustive search. However, our goal is to analyze the general impact of design space parameters on side-channel vulnerability without considering a practical attack in a real-world scenario. For comparing the amounts, we define an exact measure for a successful key recovery: First, we determine the sampling point $t_{\max}$ with the highest overall correlation, which most likely corresponds to the point in time when the last encryption round is computed. As defined in Section 2.3, we consider an attack successful, if the correlation for any of the 8 bits in $t_{\max}$ for the correct key guess is larger than 1.5 times the second-highest correlation value or smaller than 1.5 times the second-smallest correlation value.

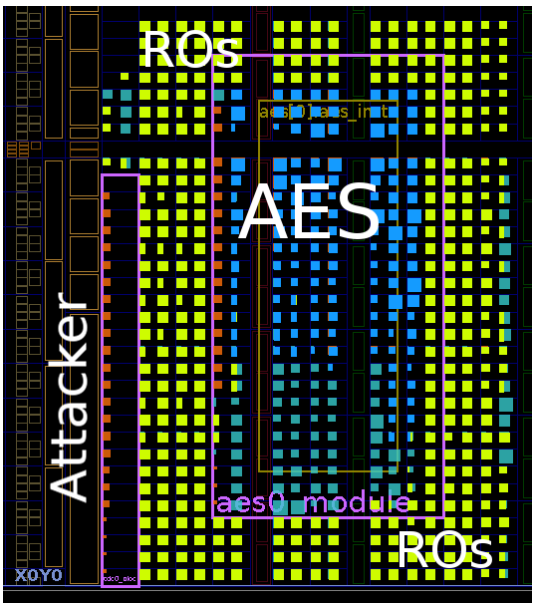


Figure 4.4.: Floorplan of our rudimentary example countermeasure based on a randomly activated array of ROs (yellow) around the AES module (blue).

4.2.4. A Simple Noise-Generation Countermeasure

To evaluate the impact of physical mapping on SCA mitigation, we employ a hiding countermeasure based on previous works [25, 70], which artificially increases the circuit noise using ROs. An array of ROs is mapped layer-by-layer around the AES modules and the amount of activated layers is determined randomly during encryption. In Figure 4.4 we show the floorplan of the countermeasure around a single AES partition as seen in the Xilinx Vivado design software. Activating a higher amount of ROs causes the supply voltage to drop, whereas deactivation of RO layers raises the supply voltage. Thus, the RO array creates randomized voltage fluctuations over the fluctuations caused by the AES module. Consequently, the attacker needs a higher amount of traces to recover the secret key due to the worse SNR. In Section 4.4, we show how physical design mapping parameters impact this countermeasure to motivate their importance not only for future but also for existing attempts at mitigating SCA.

4.3. Experimental Setup

As mentioned before, we perform two different kinds of experiments on our evaluation platform. In the first subsection, we provide details about the platform itself, whereas the experiments are explained in the following two subsections.

4.3.1. Platform

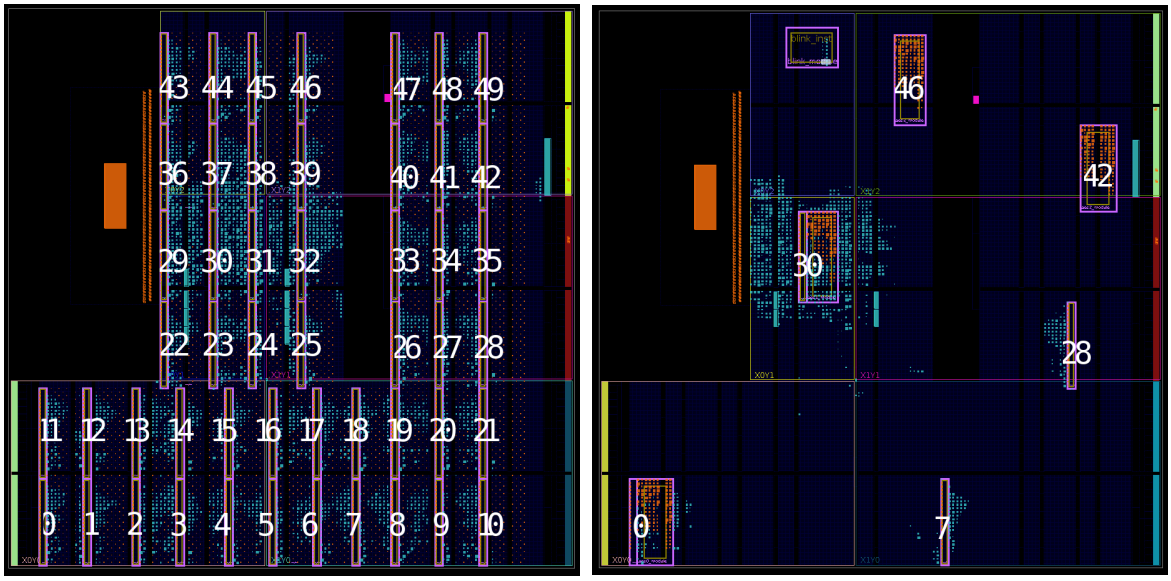
For our experiments, we use the Pynq-Z1 board from Digilent, which is based on a Xilinx Zynq XC7Z020-1CLG400C. This SoC consists of Artix-7 based Programmable Logic (PL) together with a dual-core ARM Cortex-A9 processor. The board is capable

Figure 4.5.: Picture of one of the four boards encapsulated in a metal case inside an ordinary household refrigerator. Three cables coming out of the case are USB, Ethernet, and Power.

of running a Linux system from an SD card on the ARM core, which allows easy interaction with the PL and fast sampling of sensor values through an AXI interface directly onto the SD card. We run our experiments on four different Pynq-Z1 boards to estimate the impact of inter-chip PV. During sampling, the boards are encapsulated in a metal casing inside an ordinary household fridge (c.f. Figure 4.5), to minimize environmental impact of temperature and electromagnetic radiation. Moreover, we repeat any experiment twice and the first result is discarded using only the second result for evaluation to minimize the differences in actual chip temperature. We also perform experiments on the ADM-PCIE-7V3, a Virtex-7-based PCIe accelerator card, where sensor values are sampled through the PCIe interface onto the host computer directly. As our objective is to confirm the generality of trends, we only evaluate a subset of the previously elaborated design space parameters on this larger platform.

4.3.2. Evaluating the Effect of Global Placement

Initially, we attempt to investigate the general influence of the locations of both cryptomodule (victim) and sensor (attacker), by performing measurements as explained in Section 4.2.2 for 50 different locations. A floorplan of our evaluation design for the Pynq-Z1 board as seen in the Xilinx Vivado design software can be examined in Figure 4.6a. In each of the 50 designated locations, we place a self-calibrating TDC sensor next to an FF-based noise generation grid next to each other. The size of the grid is chosen to correspond to the size of the AES module implementation we use in later experiments. Several multiplexers, which are controlled by registers from the ARM core, allow the activation of a specific sensor/FF grid combination to evaluate all 2 500 possible combinations. For a specific combination, we measure either $\delta^{\mu}_{s,n}$ or $\delta^{\sigma}_{s,n}$, where δ^{μ} and δ^{σ} are defined as in Section 4.2.2 and s and n correspond to the sensor and noise generator locations as presented in Figure 4.6a respectively. We examine which



(a) Floorplan of our placement impact evaluation design on the Pynq-Z1.

(b) Floorplan of our attack success evaluation design on the Pynq-Z1.

Figure 4.6.: Floorplans of our evaluation designs as seen in the Xilinx Vivado Design software for analyzing the impact of global placement on the entire FPGA and assessing the attack success on four by four different sub-locations.

sensors show the most sensitivity to all noise generators as well as which grid modules generate the largest impact on all sensors.

Thus, all 2 500 possible combinations are activated one by one and for each noise generator location $i = \{0, 1, ..., 49\}$ we compute the total impact caused by noise generators on either sensor average or variance, $I^{\mu}(i)$ or $I^{\sigma}(i)$ as the sum of all impact values caused by the FF grid in that location. Likewise, for each sensor location $i = \{0, 1, ..., 49\}$ we compute the total sensitivity based on sensor average or sensor variance, $S^{\mu}(i)$ or $S^{\sigma}(i)$, as the sum of the sensor's sensitivity in that location to all FF grids. More specifically, the total impact and total sensitivity values are defined as follows:

- Total sensitivity based on sensor average of a sensor at location i:

$$S^{\mu}(i) = \sum_{k=0}^{49} \delta^{\mu}_{i,k}$$

- Total impact based on sensor average caused by a noise generator at location i:

$$I^{\mu}(i) = \sum_{k=0}^{49} \delta^{\mu}_{k,i}$$

- Total sensitivity based on sensor variance of a sensor at location i:

$$S^{\sigma}(i) = \sum_{k=0}^{49} \delta^{\sigma}_{i,k}$$

- Total impact based on sensor variance caused by a noise generator at location i:

$$I^{\sigma}(i) = \sum_{k=0}^{49} \delta_{k,i}^{\sigma}$$

These four total impact and total sensitivity values $I^{\mu/\sigma}(i)$ and $S^{\mu/\sigma}(i)$ are measured 1 000 times and averaged to finally acquire a map of locations on the FPGA fabric with their respective capability to sense or generate voltage noise.

4.3.3. Evaluating Attack Success

To assess the attack success on the Pynq-Z1 we evaluate CPA with up to $100k$ measurement traces as explained in Section 4.2.3. For evaluation of the parameter impact on a protected design, we collect up to $500k$ traces with the enabled hiding countermeasure described in Section 4.2.4. On the ADM-PCIE-7V3 the attack is more difficult in general, which is why we collect up to $10M$ traces for each parameter selection on that platform.

The AES core implements a block cipher encryption with a 128-bit key length and takes around 580 LUTs ($\approx$ 1% of all LUTs on the Pynq-Z1) on our Xilinx 7-Series FPGAs. The module uses a 32-bit data-path, so 4 bytes are computed simultaneously, and one round takes 5 clock cycles to complete. The four S-Boxes are implemented in logic and not, for instance, as a lookup table in BRAM. We always operate the AES encryption at a frequency of 25 MHz, whereas the attacker sensor samples at 100 MHz.

Since the amount of data we would require to evaluate all possible 50×50 locations from the initial impact experiments would exceed multiple terrabytes for only a single local placement or board, we restrict the attack evaluation to four different sensor locations and four different AES locations. This restriction results in four by four CPA results. The choice of locations is rather arbitrary but with the results of the previous experiments on global placement in mind, choosing sensor and AES locations which are more sensitive or have a larger impact on the voltage fluctuations. Figure 4.6b shows the selected locations on the FPGA floorplan for the Pynq-Z1.

We extend the experiment to four different boards and four different local placement strategies, to explore the effects of PV and local placement of the primitives within the AES module. The different placement strategies to explore the impact of local placement of FPGA primitives within each AES module are selected from the available options in the Xilinx Vivado design software:

- Default strategy
- Optimize for performance
- Optimize for area
- Optimize for power consumption

After compiling the design with a selected placement strategy, the placement constraints for one AES module are saved and replicated to the other four AES locations.

Design parameter	**Available parameter choices**			
Inter-chip PV	Board A	Board B	Board C	Board D
Sensor location	0	7	30	28
AES location	0	30	42	46
Local primitive placement in AES modules	Default settings	Performance optimized	Area optimized	Power optimized

Table 4.1.: Overview of our design space exploration, leading to 256 experiments on the success of CPA on AES.

Then, we recompile the design for evaluation with default settings but applying the placement constraints of one of the four strategies on the AES modules.

In total, this gives us 256 experiments, which we can then analyze w.r.t. the influence of each parameter on the success of the attack. An overview of the entire considered design space is given in Table 4.1, with the global location identifiers corresponding to the assignments in Figure 4.6a. Additionally, we evaluate the effect of routing variations by recompiling the design with identical placement constraints on the AES modules. On the ADM-PCIE-7V3, we evaluate only a single board and the default local placement strategy, varying the global sensor and AES placement parameters, which leads to 16 experiments on that platform.

If not explicitly defined otherwise, we always attack the first round key byte of the last AES encryption round with the default key used in the examples in the appendix of the NIST AES specification [37], where the correct hexadecimal value of the key byte is 0xD0. As an example, we also show results for the second round key byte on only a single board in the appendix in Appendix B. The random plaintexts used in the attack are identical and issued in the same order for each experiment.

4.4. Results

In this section, we present the results of our experiments, which are later discussed in Section 4.5. We start with presenting the measurements from evaluating the impact of noise modules on sensors in various respective locations, followed by the results of CPA on AES in different setups.

4.4.1. Effect of Global Placement

Here we show the results of performing measurements as described in Section 4.2.2 and Section 4.3.2 to evaluate the influence of voltage noise generators on voltage sensors in different respective locations on the chip. Through these measurements we intend to identify locations that generate a high amount of voltage fluctuations (as a source

of information leakage) as well as locations that are more sensitive to them (as a sink, for a potential attacker sensor). The initial motivation is to be able to correlate the results with the CPA attack success evaluation, as presented in the next subsection. Although we are unable to actually present a direct correlation, we can still identify some connections between the results of the two experiments, which is why we present both of them in this work.

We measure all four variants of the total impact and total sensitivity $I^{\mu/\sigma}(i)$ and $S^{\mu/\sigma}(i)$ as defined in Section 4.3.2 of all 50 locations on two different boards. In Figure 4.7a, we see how all noise generation grids cause a specific distribution of sensitivity of the sensor average $S^{\mu}(i)$ for each sensor location i on the FPGA fabric. On the other hand, we show in Figure 4.7b the impact $I^{\mu}(i)$ on all sensor average values caused by each toggling FF grid location i. In all figures, the color scale, which is logarithmic, reflects the sensitivity of the sensor location or the impact of the noise generator location respectively: A higher sensitivity or higher impact is red, whereas lower sensitivity corresponds to a green color. The actual impact/sensitivity values have no specific meaning and are therefore omitted in the diagrams. The locations in the figures correspond to the respective locations as presented in Figure 4.6a in Section 4.3.2.

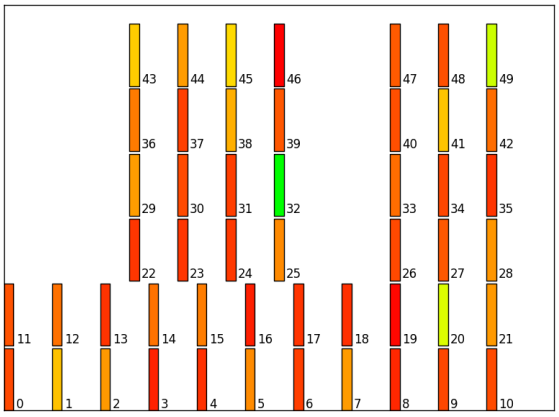
When considering the impact and sensitivity values $I^{\sigma}(i)$ and $S^{\sigma}(i)$ based on the variance of the sensor measurement presented in Figure 4.7c and Figure 4.7d, we observe almost no difference in the total impact $I^{\sigma}(i)$ caused by each noise module at location i, but quite a few differences in the total sensitivity of each sensor to all noise generators $S^{\sigma}(i)$.

We also evaluate the impact and sensor sensitivity values based on sensor average and sensor variance on a second board of the same type. The results, which can be found in the appendix in Appendix B, are consistent with the ones obtained on the first board: Again, the spatial distribution of the total impact caused by each noise module is very similar to the results on the first board, whereas the distribution of the sensor sensitivity differs significantly.

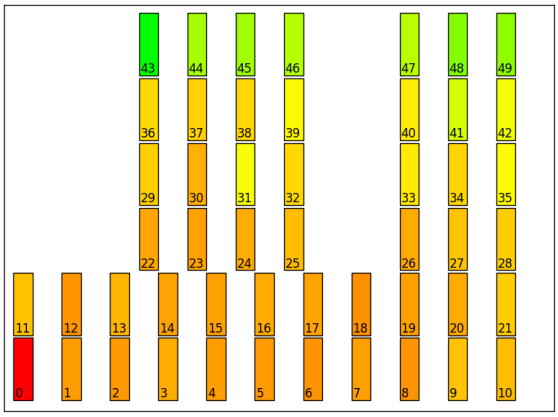
From these initial experiments, we can identify sensor locations, that seem to be more sensitive to the voltage fluctuations in general, such as location 46. However, the location sensitivity is not fixed across boards, since locations 20 and 49 are rather insensitive on board A, but much more sensitive on board B. Moreover, sensitivity depends on the measurement method, as the impact on sensor average has a different distribution than the impact on sensor variance. On the other hand, we observe very similar distributions of the impact caused by noise generators in specific locations, where location 0 is the strongest across boards and even with different measurement approaches.

In conclusion, the sensitivity of specific sensors seems to be predominantly defined by inter- and intra-chip PV and the asymmetry of the PDN across the chip, whereas the impact of switching activity on the supply voltage depends mostly on the noise generator location.

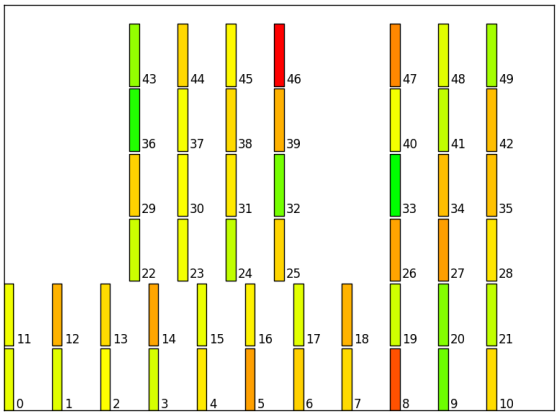
We present in the next subsection that CPA attack success seems to be correlating with neither sensor sensitivity distribution nor noise generator impact distribution. This is due to the high impact of local primitive placement when replacing the noise generators with the actual AES encryption modules. However, we are able to draw some conclusions from the attack success w.r.t the global AES module placement.



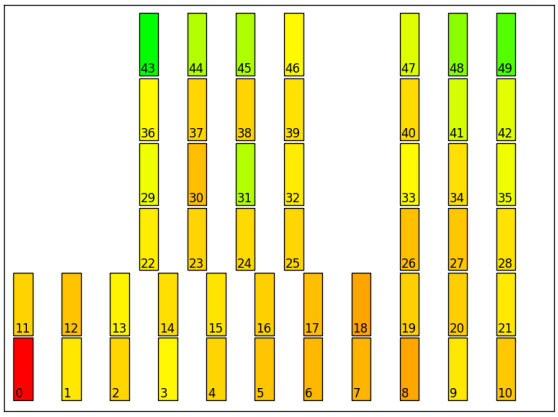
(a) Sensitivity based on sensor average $S^{\mu}(i)$ in different sensor locations i.



(b) Impact based on sensor average $I^{\mu}(i)$ caused by noise generation grids in different locations i.



(c) Sensitivity based on sensor variance $S^{\sigma}(i)$ in different sensor locations i.



(d) Impact based on sensor variance $I^{\sigma}(i)$ caused by noise generation grids in different locations i.

Figure 4.7.: Influence of global placement of FF-based noise generators and TDC sensors evaluated and averaged over 1 000 measurements on board A. Locations that are colored orange/red correspond to sensors that are more sensitive or noise generators that cause a higher impact respectively. Green colored locations are less sensitive or cause less impact.

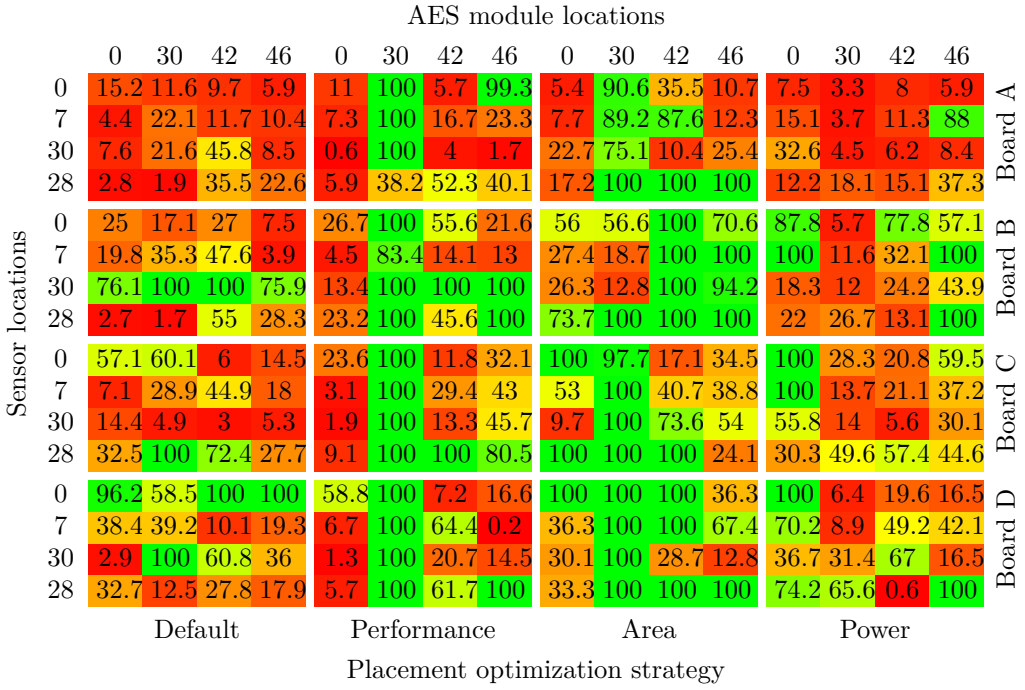


Figure 4.8.: Minimum amount of traces (×1 000) required with different placement strategies for all possible combinations of AES and sensor location on four different Pynq-Z1 boards. Red colored cells correspond to combinations where an attack is easy, green colored to parameters resulting in a difficult attack. A value of $100k$ means that we were unable to recover the key with $100k$ traces.

4.4.2. Attack Success Evaluation on an Unprotected Design

The main part of our experiments is the assessment of CPA success on AES in all 256 scenarios, as explained in Section 4.3.3. As mentioned in Section 4.3.3, we choose a subset of four locations, which are shown in Figure 4.6b for sensors and AES modules out of all 50 locations from the previous experiment on voltage noise impact. The choice of locations is partially based on the results of the previous experiments. We include locations of interest such as location 0, which caused the highest impact on all sensors in the experiments in Section 4.4.1, and select the rest arbitrarily, but further apart to reasonably evaluate the global mapping impact.

In Figure 4.8 we present all results of performing CPA on four different boards, four different local placement strategies and the four by four possible combinations of sensor and AES location. The table cells are colored according to the SCA success assessment: Less amount of traces required corresponds to a more vulnerable setup and is thus colored in a red tone, whereas a higher amount of traces is marked green, to indicate less vulnerability. We note again that only attacks with up to $100k$ traces have been evaluated, therefore all values of $100k$ indicate an unsuccessful attack with $100k$ measurements. First, we notice that despite attacking the same byte of the last AES round key, despite the identical random plaintexts and despite our extensive efforts to eliminate fluctuations in environmental parameters, the difference in the required amount of traces is remarkably up to 500×.

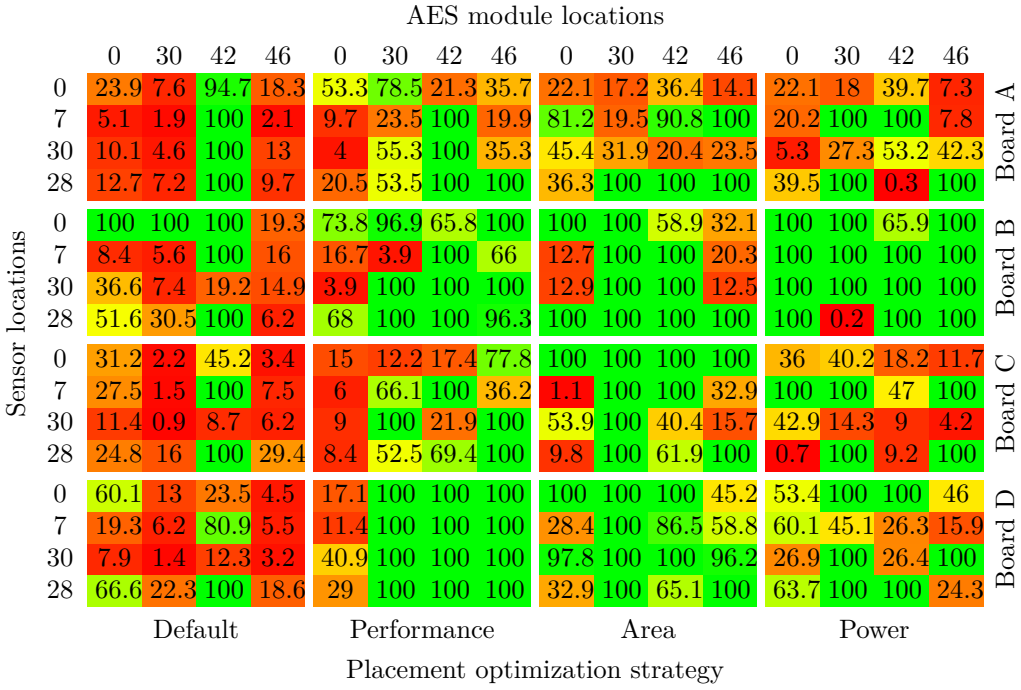


Figure 4.9.: Minimum amount of traces (×1 000) required on recompiled designs to analyze the effect of routing randomization. Red colored cells correspond to combinations where an attack is easy, green colored to parameters resulting in a difficult attack. A value of $100k$ means that we were unable to recover the key with $100k$ traces.

We note that the routing randomization impact is by no means a negligible effect from a security perspective. Since we are only able to fix the primitive placement in our implementations, a recompilation causes the Vivado Design software to re-route certain nets. Figure 4.9 presents the results of evaluating CPA on recompiled bitstreams under the same parameter variations as before. Although the previously discovered dependencies are still valid, the overall results differ quite significantly. These results show the significant impact of local placement and routing on the security of the module in terms of SCA vulnerability.

Next, we try to extract dependencies between the four given parameters (board, global placement of AES, global placement of the sensor, local placement strategy) and the attack success. Therefore, we also average the results across different dimensions and also across the results from original and recompiled bitstreams, which is presented in Table 4.2.

In the following, we summarize and discuss the major observations from the experiments:

- The *local placement strategy* of the AES modules seems to have the strongest influence on the attack success, although the difference in required traces is only a little more than 2× for default and area-optimized placement.

- The *board* parameter, which corresponds essentially to the inter-chip PV, shows that board A, which is also the oldest board, is the most vulnerable of the four.

Overall average	42.6			
Local placement strategy	Default	Perf.	Area	Power
	26.5	47.1	56.4	40.2
Sensor location	0	7	30	28
	51.5	48.1	44.4	60.1
AES location	0	30	42	46
	37.4	59.9	58.7	48.1
Board	A	B	C	D
	37.1	58.8	50.1	58.1
Attacker-victim distance	adjacent		non-adjacent	
	57.8		40.4	

Table 4.2.: Averages over the minimum amounts of traces (×1 000) required for a successful attack across all other dimensions and both original and recompiled design, when considering only specific design space parameters.

- Considering the *global sensor location* on the FPGA fabric, we see that it seems to be the least important parameter. We are unable to deduct a board-independent sensor location that has the best or worst attack success, when averaging over all other parameters.
- Although the impact of the *global AES location* on the FPGA fabric is not strong either, we determine location 0 to be the most vulnerable one on average. This concurs with the results of the previous experiments in Section 4.4.1 on voltage noise impact depending on global module placement, from which location 0 is expected to be the most vulnerable. However, when applying the power-optimized local placement constraints, location 0 seems to be actually one of the less vulnerable ones.
- Regarding the *design distance*, attacks are not necessarily easier, when the attacker design is placed closer to the victim design. In the last row of Table 4.2, we compare the attack success for adjacently and non-adjacently placed AES modules and sensors. We see that in fact attacks are slightly easier on average between the non-adjacent locations.

As the differences in the independent results in Figure 4.8 and Figure 4.9 reach up to 500×, we conclude that combinations of multiple design space parameters need to be considered in order to eventually improve security in a multi-tenant FPGA. When considering the location of the AES module, we see that for a specific local primitive placement strategy, we are clearly able to infer a less vulnerable and a more vulnerable location across different boards. The most definite example would be the AES location 30, which is the least vulnerable for all boards when placed with the performance or area optimizing strategy. In the same way, we see how location 0 is most vulnerable, when using performance-optimized mapping.

In general, a dependence of CPA success on the global placement can only be identified w.r.t. a specific local placement strategy. However, we can draw some general

Sensor locations \ AES module locations	0	30	42	46
0	290	10,000	1,340	10,000
7	4,560	10,000	780	2,210
30	2,170	90	2,490	2,270
28	5,210	10,000	250	2,060

Figure 4.10.: Minimum amount of traces ($\times 1\,000$) required for all possible combinations of AES and sensor location on a Virtex-7-based ADM-PCIE-7V3 accelerator card. Here, a value of $10M$ means that we were unable to recover the key with $10M$ traces.

conclusions from the evaluation of the 50×50 locations in the previous subsection. In those experiments, the sensitivity of a specific sensor location was very different across boards, whereas the impact of a specific noise generation module was very similar. This initial result is reflected in the results in Figure 4.8, Figure 4.9 and Table 4.2 where the location of the AES module is more critical to the attack success as, for example, the board-independent low vulnerability of the AES module in location 30 shows.

We want to mention that results are also different, if we attack another AES round key byte. This is expected in our setup, as the implemented AES module uses four independently placed S-Boxes. For the amount of traces required in a successful CPA, however, the placement of S-Box and state register is critical. Attacking a different byte is thus equivalent to attacking a different local placement. In the appendix, we exemplarily show the results on attacking the second round key byte instead of the first one on only a single board.

Finally, we present results of evaluating attack success on the Virtex-7-based ADM-PCIE-7V3, to confirm our findings on design parameter impact on SCA vulnerability across different platforms.

The attack is generally harder on the ADM-PCIE-7V3 which could be due to various reasons, such as a more stable power supply or the hierarchical layout of the larger FPGA with different clock regions. Thus, we need to collect up to $10M$ traces for an adequate comparison of the required amount of traces for a successful CPA. However, our key observation – the significant impact of design space parameters on side-channel vulnerability – is clearly visible in the results on this platform as well. In Figure 4.10, we see again differences in the minimum amount of traces required for a successful attack of more than $100\times$, depending on the global placement parameters only. An AES module in location 30 can be attacked by a sensor in location 30 with only $90k$ traces, whereas other combinations are not vulnerable with up to $10M$ traces.

4.4.3. Impact of Physical Design Parameters on SCA Countermeasures

Last but not least, we implement a noise-generation-based hiding countermeasure, as explained in Section 4.2.4, and evaluate the success of attacks on a protected design on board A. To acquire comparable results, the countermeasure is implemented in a way, that allows to disable the noise-generating RO array without recompiling the bitstream. First, we need to evaluate the unprotected modules again as we cannot

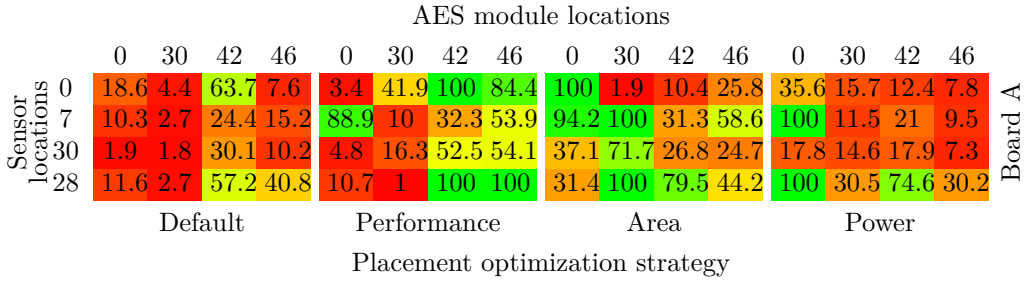


Figure 4.11.: Minimum amount of traces (×1 000) required for a successful attack on board A without the RO-based countermeasure enabled.

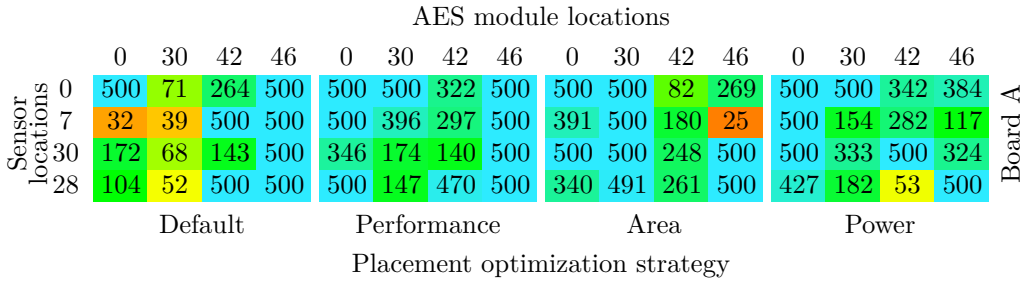


Figure 4.12.: Minimum amount of traces (×1 000) required for a successful attack with up to 500k traces on a design protected by our simple RO noise generator on board A. Blue cells reflect setups where up to 500k traces are required. Here, a value of 500k means that we were unable to recover the key with 500k traces.

compare the protected implementation with the previous results due to the changes to local placement and routing introduced by the countermeasure. The results of the baseline evaluation are presented in Figure 4.11. As expected, the results differ from the previous experiments but are well within the expected range.

Next, the countermeasure is enabled and up to 500k traces collected to account for the increased attack difficulty. Figure 4.12 shows the attack success of CPA on the protected AES modules. We observe an expected significant increase in the minimum amount of traces required for a successful key recovery, in many setups the design cannot be attacked even with 500k measurements. However, whereas in some configurations the countermeasure can raise the amount of traces required by a factor of over 260×, in other settings the increase is only minimal. For a very small subset of the physical design parameters, the amount of traces even decreases with the enabled countermeasure.

These results clearly show that the effectiveness of SCA countermeasures are greatly influenced by physical design and mapping parameters. Although hiding countermeasures come with significant area and power overhead, by carefully considering the impact of local and global mapping, one can achieve similar level of protection with virtually zero-overhead. From a different perspective, neglecting the effects of physical design can significantly hinder the success of such explicit countermeasures.

4.5. Discussion

After reporting our results in the previous section, we would like to discuss them and carve out consequences for secure virtualization and multi-tenancy of FPGAs in this section.

4.5.1. Impact of Physical Design Parameters on Side-Channel Security

The most important conclusion to draw from our experiments is the importance of physical design parameters w.r.t side-channel vulnerability. We observe significant differences in the minimum amount of required measurements to successfully recover the secret AES key. On our experimental platforms, we are sometimes able to recover the key byte with only 200 traces, whereas other parameter settings prevent key recovery even with $100k$ traces. Those differences are well within the range of some side-channel countermeasures, making physical design mapping almost as critical.

Experimenting with an exemplary noise-based hiding scheme shows how critical the dependence on mapping parameters actually is. We reach an increase in the minimum amount of traces required for key recovery by more than $260\times$ in some cases, whereas in other setups the countermeasure does not help at all to prevent side-channel leakage. Researchers already pointed out the importance of placement and routing for specific masking countermeasures [59], but our systematic analysis shows its general impact, especially in the context of FPGA-internal attacks.

The presented results demonstrate the high influence of local placement and routing on the SCA vulnerability of victim modules. This is also the reason, why a surrogate model of the module activity based on the global placement on the FPGA fabric (represented by noise generating modules in the first set of our experiments) does not correlate with the success of CPA on AES modules for actual key recovery attacks. A challenge to the research community will be to investigate physical design and mapping from a side-channel security perspective.

Our evaluation of attack success on a data-center scale Virtex-7 based device proves the impact of design space parameters on larger FPGAs as well. Moreover, we observed how the layout and organization of the larger fabric further increases the design space and thus the variation in side-channel vulnerability. Modern high-end FPGAs are often composed of multiple dies, where side-channel attacks have been shown to be still possible [76]. However, those multi-die chips add another design space parameter to be considered.

Without extensive vendor knowledge, we are unable to determine the exact parameters and characteristics of the underlying FPGA architecture and hardware, which lead to the observed effects. Nevertheless, we assume two causative factors: The PDN design is non-uniform across the chip, which leads to differences in the impact of current draw on the supply voltage, and the mapped user logic components are subject to intra-chip PV. Both factors, on one hand, impact the sensitivity of sensors in different locations and, on the other hand, lead to differences in the voltage traces caused by modules in different locations. However, a thorough knowledge of the architecture and underlying hardware is reserved to the vendors. With some FPGAs being actively

reverse-engineered by a growing community, we may be able to unveil dependencies between specific placement and routing and side-channel vulnerability, despite the lack of knowledge about the underlying PDN architecture.

4.5.2. Possible Countermeasures based on Physical Design Parameters

Our results do not necessarily only introduce a new problem to the area of defending against on-chip side-channel attacks, they may be also useful in developing new countermeasures. We may be able to leverage the high influence of placement and routing to optimize algorithms for security.

Countermeasures based on design space parameters on both hypervisor and user side will require thorough, per-device analysis. This surely requires – depending on the device size and the approach – a significant amount of effort and time.

A possible approach could be built on three steps:

- First, a hypervisor generates multiple local placement mappings for a specific cryptomodule.
- Then, for each mapping of the module, a global analysis determines regions, which are less vulnerable to side-channel attacks. Evaluating actual attacks for all possible combinations is hardly feasible. However, in future works we may be able to identify an adequate model similar to our noise generator approach in Section 4.3.2, which can assess side-channel vulnerability in less time than a full attack.
- Nevertheless, in the third and final step, using the results for improving side-channel security on a specific FPGA comes at zero overhead. On the hypervisor side, a global map of secure locations and precompiled cryptocores can be provisioned, which can be deployed by the user as a building block in security-critical applications.

This approach could improve security without requiring any additional area or resources of the FPGA and can be used in combination with other well-known SCA countermeasures [57, 59, 60, 66, 71, 72].

All in all, our contribution points out an important aspect of the development of countermeasures against side-channel attacks in virtualized multi-tenant FPGAs.

4.6. Conclusion

In this chapter, we thoroughly analyze the impact of physical design mapping parameters in multi-tenant FPGAs on the success of SCA attacks. The results of more than 256 experiments with CPA attacks on an AES FPGA implementation with up to $100k$ measurement traces reveal differences in the required amount of traces for key recovery of up to several hundred times. We show that the attack success depends on where attacker and victim modules are placed on the FPGA, how exactly the primitives within the module are locally placed, and on chip-to-chip variation.

Moreover, our experiments on modules, which are protected by a noise generator hiding scheme, prove that SCA countermeasures can be more or less effective under certain design mappings. Consequently, those parameters are of great importance when implementing protected cryptographic designs in multi-tenant FPGAs.

However, our analysis does not only introduce a new aspect to the application of SCA countermeasures on virtualized FPGAs, but may also be a starting point for future works on hiding leakage through design mapping only. Such countermeasures would come at zero overhead and would provide a valuable asset in securing multi-tenant FPGA access against internal side-channels. On the other hand, neglecting such device and physical design dependencies can compromise security of virtualized FPGAs in the cloud.

5. Mitigating Electrical-level Attacks towards Secure Multi-Tenant FPGAs in the Cloud

This work on a proposed bitstream checking scheme to find malicious signatures has been published in [22] together with Dennis Gnad and Mehdi Tahoori.

In the previous chapters, we explore the extent of remote fault and side-channel attacks on multi-tenant FPGAs. Here, we first discuss a practical countermeasure to mitigate the threat, expecially in the cloud-computing scenario.

On one hand, it might be possible to change the hardware architecture of the FPGA itself, such as separating the FPGA fabric into blocks of individually powered voltage islands. Fault attacks can be detected with the high-speed voltage sensors we described in Section 2.6. On the other hand, the hypervisor can be made responsible to only allow the programming of benign bitstreams to the FPGA fabric, by checking them before loading, similar to antivirus software checking binary executables. Existing work already mentioned that partial configuration bitstreams might require checks as a safety measure, to prevent dangerous configurations that cause short-circuits [77]. Two security related works did also briefly express the idea of checking bitstreams for malicious circuits as a countermeasure to their attacks [10, 14]. Since architectural changes are expensive, require a new chip generation, and sacrifice flexibility, it is crucial to follow the path of bitstream checking, without overly restricting benign designs.

Hardware trojans are a related security issue, where malicious logic can be hidden in an otherwise legitimate design. Typically, they get introduced into ASICs by having access to a specific part of the IC design or manufacturing phases, which can be on netlist or layout level [78]. By contrast, a malicious design on a multi-tenant FPGA is not integrated into the existing victim circuit, but implemented as a completely separate design with no signal connections. Thus, hardware trojan detection countermeasures [79] are not suited in the given scenario. Functional trojan detection fails to identify a malicious design without logical connections. On the other hand, detection methods based on power analysis require a golden reference design for comparison, which does not exist for an arbitrarily mapped multi-tenant FPGA.

In this chapter, we propose a methodology to check FPGA bitstreams for patterns of malicious structures, which we call *signatures* in analogy to computer virus signatures. By extracting a technology-mapped netlist graph from existing bitstreams, we check for combinational and sequential circuits that are of high risk to become a security threat. Our work is orthogonal to existing methods for hardware trojan detection, such as [80], as we focus on potential threats on the electrical level from logically

isolated designs. We present experimental results from implementing specific subsets of our methodology on Lattice iCE40 FPGAs and evaluating our bitstream checker on a magnitude of benchmark designs as well as on reference designs, reproducing attacks from previous works. Finally, we discuss to which extent the imposed limitations affect benign designs, and what are the possible next steps to decrease these limitations without sacrificing security.

Together with proper isolation measures on the logical level [44, 81] and hardware trojan countermeasures [79], our methodology contributes to enable sub-chip virtualization of accelerators in the cloud and other systems. To demonstrate the feasibility of our approach, we extend the existing attacks and reproduce them on Lattice iCE40 FPGAs, and discover more fundamental properties of potential attacker designs. Furthermore, evaluating our bitstream checking methodology on a subset of legitimate benchmark designs from different collections proves that the proposed security-related restrictions are practical. We analyze designs from the IWLS 2005 benchmarks [82] and the Verilog-to-Routing (VTR) benchmarks [83, 84] as well as a RISC-V processor implementation [85] and a LEON3 processor design [86]. All the evaluated benchmark designs do not exhibit properties that would identify them as potentially malicious. Our extensions to the existing tools and new implementations are available online.[1]. We can summarize all our contributions as follows:

- Description of fundamental signatures of designs that can be used in side-channel or fault attacks on the electrical level
- A generic bitstream checking flow to detect those signatures using static as well as dynamic analysis methods
- Implementation of previously known attacks on Lattice iCE40 FPGAs
- Extended investigation of attacks, which make use of FPGA primitives, that have not been explored in previous works, and can evade detection by previous bitstream checking algorithms [23]
- A thorough evaluation of our methodology on a large set of benchmark and attacker designs, showing the feasibility of our method

The remaining chapter is structured in the following way: In Section 5.1, we summarize the possible attack scenarios based on the respective related work. Section 5.2 details various signatures that we find eligible for fault injection or side-channel attacks. Subsequently, we reproduce the known attacks in Section 5.3.1. Our experimental setup is presented in Section 5.3, and we show the results of evaluating our bitstream checker on benign benchmark designs and attacker designs in Section 5.4. Finally, we draw a conclusion in Section 5.5.

5.1. Existing Attacks and Related Methods

In Section 2.6 we already explained some basic concepts of remote fault and side-channel attacks in FPGAs. Here, we provide further background information on the basics of

[1] https://cdnc.itec.kit.edu/MaliciousBitstreams.php

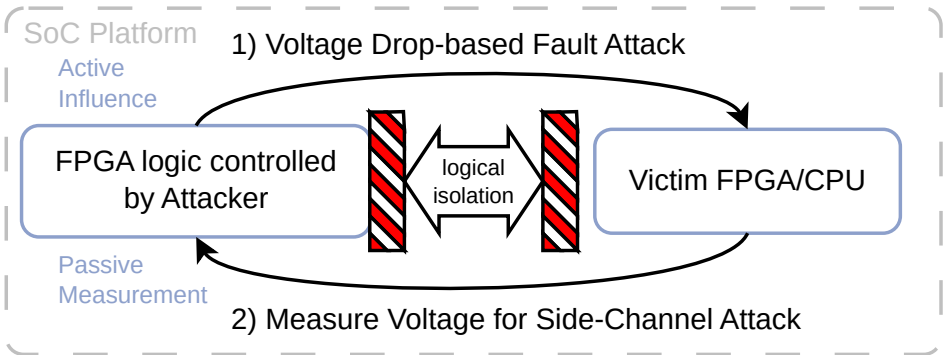


Figure 5.1.: Summary of the two voltage-based threat categories in a system with shared FPGA logic, as an overview over the previous works [10, 12–14]

existing attacks, which have been used to extract basic signatures for threat detection. An overview on the different threats ins systems with shared programmable logic is provided in Figure 5.1. Note that more recent attacks, such as the one we present in Section 3.1, can evade detection by the approach presented in this chapter, and would require to adapt the signatures. However, the general methodology to extract metadata from bitstreams and detect threats based on specific properties is still valid.

5.1.1. Fault Attacks

Since the most effective way to cause a voltage drop from within an integrated circuit is to cause a high change in current within a short time interval (dI/dt), this behavior was exploited in a Denial-of-Service attack to crash FPGA and FPGA-SoC devices [10] and also used to inject faults into an AES hardware module on a shared FPGA [13].

Ring Oscillators (ROs) can have a very high toggling frequency, and thus require a high current in a short time frame, since every change from 0 to 1, or vice versa, requires electrical current for charging and discharging the respective transistor gate capacitances. These ROs can be implemented in large quantities using programmable resources on the FPGA fabric. In the top part of Figure 5.2, we show how a Lookup-Table (LUT) can be re-used to implement an inverter with enable gate as a combinational cycle (or loop[2]). By using many LUTs, a whole array of ROs can be implemented, as shown in Figure 5.4. While the ROs are enabled, they internally oscillate at their own frequencies $f_{\text{RO-internal,0}}$ ... $f_{\text{RO-internal,N}}$, depending on physical variations.

On top of that, it is required to start and stop the ring oscillators at specific frequencies, defined through f_{toggle}. This f_{toggle} is necessary, since causing a single high voltage drop alone was shown not to be sufficient to crash the FPGAs [10]. Due to the complexity of RCL networks and voltage regulators, specific toggling frequencies for a large RO array can impact the power supplies severely, and finally lead to crashing a full FPGA, or a complete SoC that contains the FPGA fabric [10]. A more precise selection of RO grid activation parameters also allows fault injection into cryptographic implementations of another user on the same FPGA, which was shown in [13] on an AES implementation.

[2] In correct graph theory terms, a loop is only a single node connected to itself

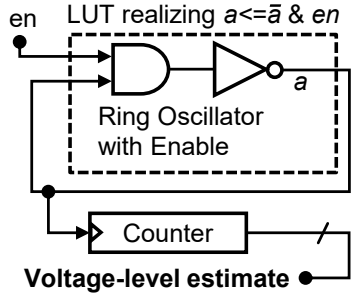


Figure 5.2.: Ring oscillator to sense voltage fluctuations (cf. [14, 87])

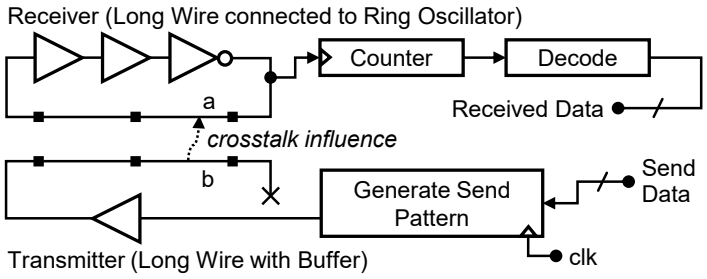


Figure 5.3.: Concept of receiving and transmitting data through adjacent long wires of FPGA interconnect, using a ring oscillator and an oscillation counter (cf. [88, 89])

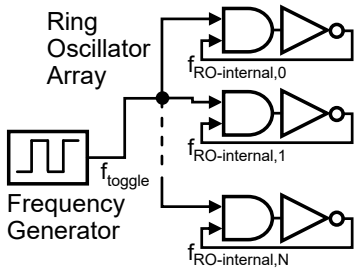


Figure 5.4.: Ring oscillator array to cause voltage drop-based faults (cf. [10])

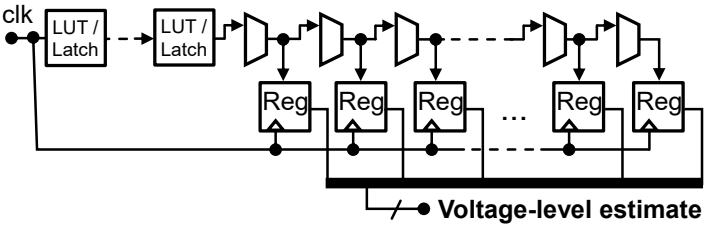


Figure 5.5.: LUT/Latch based delay line with carry-chain based Time-to-Digital Converter to measure voltage fluctuations in FPGAs (cf. [12, 73])

It was shown that partial reconfiguration of erroneous bitstreams can cause short circuits on FPGAs at runtime, which is a suitable method to cause unexpected behaviour and attack an FPGA based system [77]. An attacker can craft a specific bitstreams, where two multiplexer inputs are connected to a LUT output at the same time, and activate the short circuit at runtime to inject faults. The authors already explained a software implementation of a bitstream scanner, included with their ReCoBus-Builder tool chain, which allows detecting this kind of short circuits, which can be activated at runtime, in corrupted bitstreams.

Since the development of our approach, many further designs for injecting faults have been proposed. We discuss the most recent attacks in Section 7.2.1 and also provide a comprehensive overview on attacks and countermeasures at the end of this thesis.

5.1.2. Side Channel Attacks

In addition to causing voltage-based faults, ROs can sense voltage fluctuations. Figure 5.2 shows the principle of how FPGA lookup-tables (LUTs) are used to implement ring oscillators. Their output signal can be connected to a counter, to measure how

often the ring oscillator toggles in a specific time frame. Since the speed of the RO is voltage dependent, its output value gives an estimate of the voltage level inside the FPGA [68, 87].

If the feedback loop a of the RO is a sufficiently long wire, as shown in Figure 5.3, the RO frequency can be affected through EM crosstalk by an adjacent long wire b, leading to information leakage. Using this principle, a logic state in a long wire that carries secret information can be estimated in the receiving long wire through the number of toggles in the ring oscillator [88, 89]. The complete data transfer scheme through EM effects is depicted in Figure 5.3. Although we continue considering this as a problem, one should note that a distance of just a few interconnect wires already prevents this type of side channel [89], and thus applying common recommendations provide a sufficient countermeasure, for instance in the Xilinx Isolation Design Flow [81].

The sensor we present in Section 2.6, which is based on a *Time-to-Digital Converter* (TDC) [73], can sense voltage fluctuations much faster, up to typical circuit speeds. We show the principle of this sensor again in Figure 5.5, which is based on routing the clock signal through a chain of buffers. In an FPGA, these buffers can be made of transparent LUT and latch elements at the beginning of the path, and in the last stages parts from the multiplexers of carry chains. Between these multiplexers, registers or latches are connected, allowing to capture how far the clock propagates through the buffer chain during one clock cycle. Since the delays through the buffers and their signal propagation depend on the supply voltage, it is possible to estimate voltage fluctuations from the values captured at the output registers of the delay line. Timing analysis tools warn about timing violations for paths through the delay-line, which are purposefully too long for propagation within one clock cycle, but this is a requirement for the sensor to work.

The aforementioned sensors were sufficient for side-channel attacks within a shared FPGA fabric or SoC [12, 14]. In both cases the voltage-dependent propagation time of a signal is measured. For this purpose, TDCs allow a faster sampling rate, possible at actual speeds of cryptographic implementations on the FPGA, while ROs are simpler in implementation. Since even a slow sampling rate works in principle, both structures need to be considered as a potential security problem.

5.1.3. Detecting Malicious Software and Hardware

Before discussing our approach of checking bitstreams for malicious logic, we discuss how existing methods to scan for malicious software work, and which approaches from hardware verification or trojan detection can be re-utilized. These strategies are not an exact match to our problem, but can give some useful insights.

5.1.3.1. Antivirus Software

Current commercial antivirus software products still use the approach of pattern matching to detect known malicious code structures, based on the well-established tool *YARA* [90]. These *signatures* can be as simple as matching an exact code fragment on the binary data or code level, but might also incorporate pattern matching methods based on regular expressions, allowing to potentially catch new variants of the same malware.

An obvious drawback of this approach is the failure to detect malicious software that deploys a previously completely unknown code pattern. Only well-known malware can be mitigated, whereas new approaches remain undetected due to the entirely structural code evaluation, as opposed to a behavioural analysis of a potentially threatening program. These are known limitations of antivirus software, but they still offer basic protection against known threats and variants of them.

Regarding malicious designs on FPGAs, only a structural code evaluation will not be enough to detect fault attacks or sensors for side-channel analysis. High switching activity causing voltage drops, for example, may occur only at a specific point during runtime, and requires timing simulation and verification in order to be detected.

5.1.3.2. Hardware Verification and Hardware Trojans

The goal of hardware verification is to prove the correct behavior of a design according to its specifications, and to fulfill all functional requirements. In practice, a mixture of functional and formal methods is used.

Functional approaches typically use tests that check specific functionality of the circuit by having verification engineers apply test patterns onto the circuit and checking the results. Formal verification uses an exhaustive approach in which specified properties (i.e. using system verilog assertions) are proven based on their satisfiability according to any combination of the input parameters. Complexity can be reduced by choosing input parameters statistically in Monte-Carlo simulations. Formal approaches can usually be only applied to submodules. Despite many innovations in SAT-solving, combinatorial explosion is still an issue when entire large designs need to be verified for every possible state of the system. Thus, the semiconductor industry typically applies a combination of all these approaches.

More recently, the common goal to fulfill all *safety requirements* in hardware verification has shifted to also incorporate *security requirements*. Among the emerging concerns in hardware security, *hardware trojans*, which describe the malicious alteration of integrated circuits, have gained attention. Trojans can be introduced during design time through malicious IP blocks, or at fabrication time in the foundry [78].

Because of this risk, there is a need to detect the existence of hardware trojans. Since hardware trojans are typically considered to be inactive and require a rare event to get activated, standard chip testing can not provide this type of assessment [78]. Instead, a common aspect of hardware trojan detection in the final IC involves comparisons at run-time or test-time. These comparisons can check side-channels such as power consumption, or involve logic tests against a golden reference. If such a reference is absent, the divergence in logical and physical runtime variations can signal the recent activation of a trojan, assuming that it is usually inactive [78].

In summary, hardware trojans use a very small area on the chip, and are integrated into an existing design. In contrast, any tenant in a shared FPGA can introduce arbitrary logic on his dedicated resources, that might be able to maliciously affecting other users in the system. In this scenario, logic-level isolation can be achieved [44], but threats on the electrical level need a new approach.

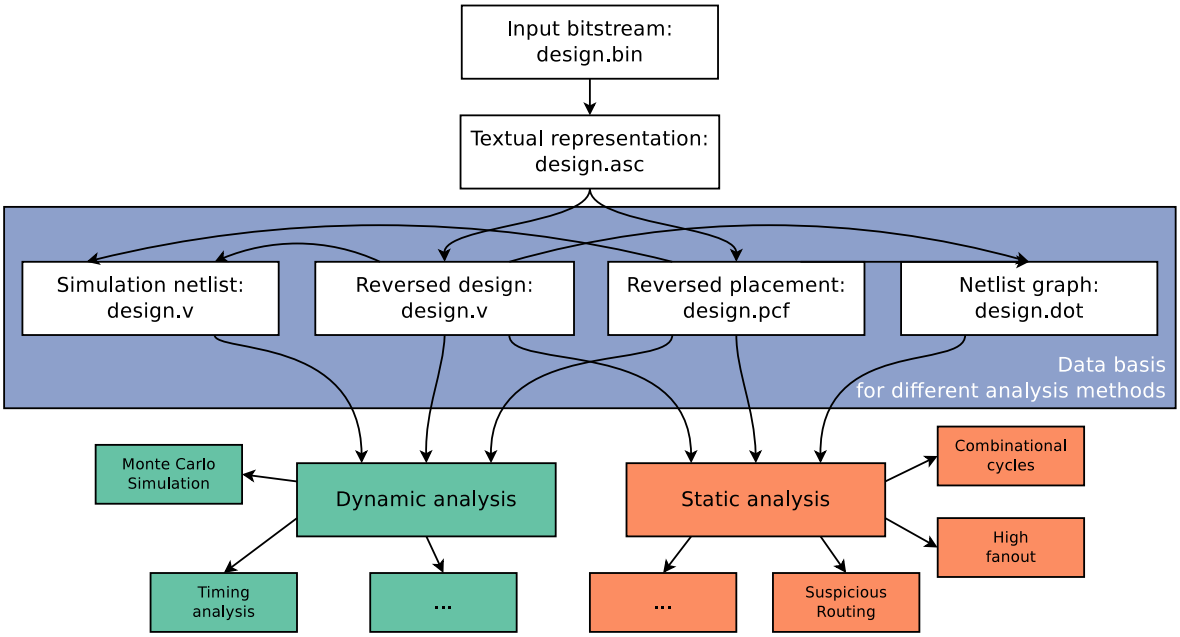


Figure 5.6.: Overview of the methodology and implemented flow to check bitstreams for threats in combinational or sequential logic.

5.2. Detecting Malicious Signatures

In this contribution, a bitstream-checking methodology is established that re-utilizes some concepts from antivirus software, as well as hardware trojan detection and verification techniques to check for various threats in systems containing FPGA logic (cf. Section 5.1). The basic idea is to check for both structural and behavioral properties that are uncommon in benign bitstreams, thus often indicating malicious intentions.

Please note that this does not mean to simply check for the known patterns of the attacks shown in the existing works, since these are just examples of possible attacks. Instead, we precisely formulate the fundamental properties that allow both branches of attacks, measuring side-channel leakage or causing faults on the electrical level. However, it is worth noting that some recent attacks, which we also present in Section 3.1 and Section 7.2.1, may evade detection, highlighting the importance of updating malicious signatures.

In Figure 5.6 we present our overall flow for checking bitstreams. To enable checking of existing FPGA bitstreams for malicious logic, they have to be made readable by common tools first. The bitstream is first unpacked into a vendor-specific text representation, which in turn is converted into regular HDL code using another tool. This HDL code is a flattened technology-mapped netlist, with wires named in numeric order, and thus very hard to understand for humans. Additionally, we extract placement and routing information, which are required for accurate timing analysis on the design. In the end, a full representation of the bitstream in terms of any possible constraints is generated, that is equivalent to the original bitstream after re-synthesis and mapping. We also generate a graph representation of the netlist graph, to facilitate structural analysis on the design using commonly-known graph algorithms.

On the acquired netlist and layout data, we can run a variety of algorithms to detect malicious signatures either through static analysis of the netlist and netlist graph, or dynamic analysis of potentially malicious runtime behaviour. In combination with the previously mentioned approaches for logical isolation and bitstream verification for short circuits, we can ensure a secure FPGA environment on the electrical level.

We extracted multiple signatures, which we matched with existing attacker designs from the literature [10, 12–14] as well as benign benchmark designs to optimize detection of malicious signatures while minimizing false positive results. Distinguishing fault and side-channel attack variants, we can generalize the respective patterns in the following way: For side-channel attacks, the attacker needs to measure fluctuations of the supply voltage by observing changes in the delay of FPGA primitives. To inject faults, the attacker needs to cause a voltage drop, which can be provoked through high switching activity.

Therefore, we check for malicious patterns that indicate measurement of voltage fluctuations with the following signatures:

- Paths, which directly violate timing constraints and can be used to estimate voltage fluctuations
- Unusual connections from data to clock pins, which can be used to manipulate timing analysis results and hide potential timing violations
- ROs, which also show a voltage dependent behaviour, since a longer delay implies slower oscillation

Whereas to mitigate fault injection, we identify the following properties:

- Runtime behaviour, which would lead to high current variations through controlled switching activity
- Structures, that can be used to synchronize and toggle a large amount of elements
- Primitives, which are suitable for causing oscillation, such as ROs

We note that the signatures for fault and side-channel attacks are by no means exclusive. For example, ROs can be utilized by an attacker for both passive measurement and active fault injection. In the following subsections, we describe in more detail, how we can possibly detect the above properties in a given design with the help of the information we acquired from the original bitstream. Although, in general, the prohibition of those signatures poses some restrictions on non-malicious designs, the low amount of false positives shows the applicability of the proposed approach.

5.2.1. Fault Attack Signatures

In previous work, it has been explored that specific switching activity is able to reduce voltage to a level that crashes FPGA chips and SoCs which contain them [10]. Moreover, timing faults can be injected into cryptographic designs to perform fault attacks [13]. The basic principle behind fault injection is a supply voltage LdI/dt drop,

as explained in Section 2.5 and Section 2.6, caused by a repeated activation of a vast amount of elements, each of which induces a high current variation. Predominantly, *Ring Oscillators (ROs)* have been used as the elementary building block to implement large RO grids. A grid of ROs as presented in Figure 5.4 is deployed with a common *enable* signal, which allows toggling the high frequent oscillation with a lower resonance frequency, which can lower the supply voltage significantly. In previous literature [13], it was yet unknown, which other basic structures may be used to provoke voltages drops in FPGAs. We show for the first time that it is also possible to inject faults using sequential logic elements. These elements will pass verification by tools checking for combinational cycles, as shown also in [91], where Denial-of-Service (DoS) attacks have been successfully deployed to FPGAs in the Amazon cloud. To enable the detection of both, the fundamental grid structure for synchronous toggling and ROs as the primarily used oscillating building blocks, we deduce two algorithmic modules to be included in our bitstream checking methodology.

The synchronous repeated activation, which is required to maximize current variation is usually controlled by a single control register node or a tree-like structure in a more elaborate attacker design. In order to prevent synchronization of a large amount of structures on the FPGA, we calculate the highest node fanout in the design, which is the highest node out-degree in the netlist graph. Netlist graph nodes which represent a valid clock in the design (clock input pin or clock generator output) are disregarded. As presented in Section 5.4, the highest fanout in an attacker design typically exceeds the average of benign designs by a significant amount. Subsequently, we can introduce a device-dependent threshold as a maximum fanout limit, to prevent designs that can induce faults from being deployed. A more elaborate method, allows identifying tree-like structures in the design netlist graph, in case an attacker utilizes a node tree to synchronize multiple malicious logic blocks. Computing *centrality* metrics [92] on the netlist graph allows identifying root nodes of tree structures within the graph. However, the definition of a threshold in this case requires a more thorough assessment of centrality metrics in netlist graphs.

ROs can be extracted from the netlist graph using common graph algorithms. In our experimental framework we use an algorithm for enumerating cycles in graphs, which is included in the graph-tool python framework [93, 94] and has a computational worst case complexity of $O((|V| + |E|) \cdot (C + 1))$, where V is the graph nodes, E is the graph edges and C is the number of cycles in the graph. The number of graph nodes V corresponds to the total amount of registers, logical elements and interconnect buffers in the netlist, whereas the graph edges E represent the directed nets between those primitives. We iterate over the list of discovered cycles and remove those that include register primitives to get a reduced list of only combinational cycles. The result must be filtered again for non-malicious cycles, which depends on the design of the underlying FPGA logic cells and synthesis software. In some cases, for example, combinational cycles exist only for logic packing convenience and can not be used to induce oscillation and fault injection. These cases must be determined by not only considering the netlist graph itself but also additional information from LUT configuration masks and known structural properties of the underlying FPGA elements.

Figure 5.7 shows an example of an inverter packed with a carry element, which results in a non-malicious combinational cycle. Our additional filter steps do not increase the

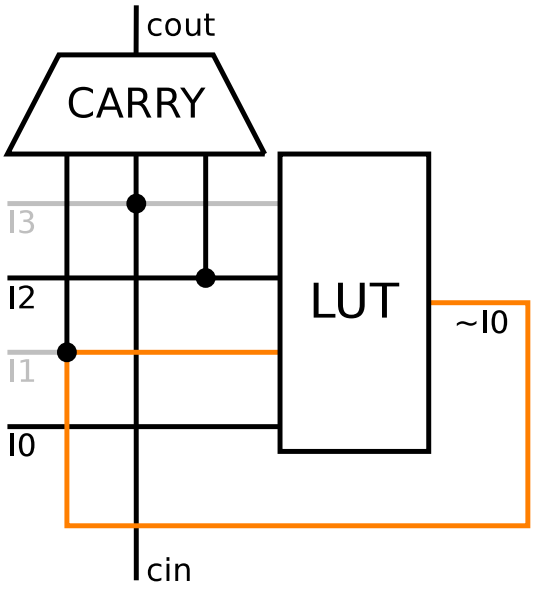


Figure 5.7.: An example of a non-malicious combinational cycle, where the LUT output is routed back to the LUT input I1, but only I0 is part of the LUT's combinational function

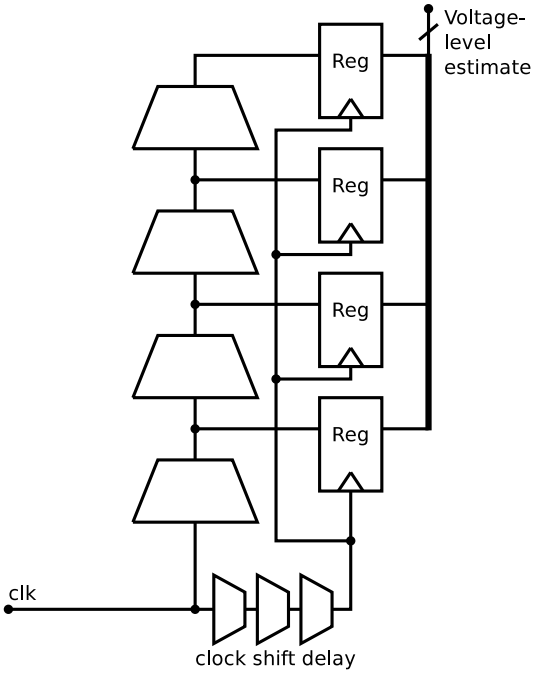


Figure 5.8.: Basic principle of a delayed clock signal used to capture the value of a short delay line, which is not detected by all timing analysis tools

overall worst-case complexity of the basic cycle enumeration tool.

An additional analysis of the dynamic runtime behaviour can further narrow down the detection. Previous publications show how the evaluation of switching activity in a design based on real or random input stimuli can be used to estimate the power consumption of an FPGA device [95]. Alternatively, test patterns can be generated specifically for the purpose of determining dI/dt voltage drops in a design with the help of SAT-solving tools [96]. The same approach is suitable for determining, whether a design is able to cause high current variation that may eventually provoke a voltage drop at runtime. However, the additional computational overhead is significant and requires a much more thorough knowledge of the internal device structure, which is available to the vendors.

5.2.2. Side-channel Attack Signatures

Side-channel attacks are based on observing the voltage dependent behaviour of FPGA primitives or constructs, and analyzing a sufficient amount of collected traces. The supply voltage is influenced by the activity of other components on the chip, such as a cryptographic module, which can leak information through causing voltage fluctuations during computations on intermediate values. Even a single bit with unstable output, which depends on the supply voltage, can be enough to recover a secret encryption key. When a sufficient amount of voltage traces has been recorded, it allows the use of simple or differential power analysis [8]. In all previous works, the observed property

to estimate the supply voltage is the switching delay of FPGA primitives, either in a combinational path for a TDC-sensor [12, 73], where a signal propagation can be sampled, or in a RO-based sensor [14, 87], where oscillation speed depends on element delay as well. We elaborated in Section 5.1, how sensors based on various FPGA primitives can be utilized in SCAs on multi-tenant FPGAs. In the previous subsection on Fault Attack Signatures, we already explained, how malicious combinational cycles can be detected with commonly known graph algorithms on the design netlist graph.

The basic principle of an RO based sensor is presented in Figure 5.2. During one sampling clock cycle the number of oscillations is counted as a measurement sample for a supply voltage estimation. Additionally, we must prevent the implementation of TDC sensors, which are based on sampling the signal propagation in a combinational path. To be able to evaluate fluctuating voltage levels, the path endpoint should not be reached by the signal within the sampling clock cycle. On the other hand, the failure of a signal to reach the endpoint of a combinational path within a clock cycle is a timing violation and should be reported by timing analysis tools. A reasonable approach to detect sensor implementations in attacker designs is therefore, to perform a thorough timing analysis on the design, after resynthesizing it from the reverted bitstream.

Please note, since the design is extracted from a bitstream, the netlist will come with no timing constraints. This will strip the design from all user constraints that could otherwise hide TDC sensors. However, it will also remove any legitimate constraints, such as false path constraints for clock-domain crossing synchronizers or similar circuits. The timing information must be derived again from the basic board configuration and external clock generator or crystal, as well as internal clock generators, and also from the placement of the logic elements from the bitstream. We show in Section 5.4 how *Phase-locked Loops* (PLLs) can be derived from the bitstream and combined with board clock source locations, which are known to a hypervisor of a multi-tenant FPGA resource.

A successful TDC sensor detection depends on the quality of the timing analysis, which will have to include all *possible* clock configurations which can be set at runtime. More complex designs might, for instance, use dynamic clock control, phase-shifted clocks or clock-domain crossings. These are general tasks for timing analysis and verification tools, which have to report dynamic clock setups in which timing violations can occur. For security, only safe dynamic clock setups must be enforced, which is out of scope for this work.

Another combinational signature, that can indicate a malicious design and is rarely used in benign implementations, is the routing of a data signal – LUT or register output – into the clock input of a register. In various scenarios, this structural element can be deployed in either sensors, or fault attacks, using a delayed clock to capture a TDC signal or causing oscillation for voltage drops. Figure 5.8 depicts the basic principle of a TDC sensor which does not require a long delay line to capture signal propagation. The clock signal is routed to the clock inputs of the capture registers through a short delay line, causing a small clock signal shift of time t_Δ. Therefore, the capture registers can sample, how far the clock signal has propagated into the delay line in time t_Δ. Clock signal manipulations with FPGA primitives are not necessarily detected by timing analysis tools, which is why we include them in our bitstream verification workflow. Since we reverse the original bitstream into a technology mapped netlist, which uses

device-specific primitives, we can easily check for non-clock signals connected to register clock inputs. We backtrack clock input signals through wires and buffers, to ensure they are connected to valid clocks in the design, which are defined by board pins controlled by the hypervisor and PLL outputs. In the results, we show that common benign designs do not implement such connections.

5.3. Experimental Setup

To evaluate our bitstream checking approach, we choose to implement known attacks as well as a collection of popular benchmark designs on an FPGA device. Since low-level vendor-specific design details of the FPGA chip are unavailable to us, a complete and thorough software implementation of all proposed bitstream checking algorithms is not possible. We use the iCE40-HX8K-CT256 device embedded into the ICE40HX8K-B-EVN breakout board from Lattice Semiconductor, since the bitstream of this device has been reverse engineered and a toolchain is publicly available [97]. It has a simple hardware structure without, for example, designated signal processing cores. In total, it offers 7 680 logic cells, each of which contains a LUT, a register and a carry chain element. Furthermore, the device has two PLLs to generate clock signals from the base board clock frequency of 12 MHz. On this device we can implement all known side-channel and fault attack variants and are able to deploy almost all basic algorithms on the bitstreams to detect malicious signatures. In the next subsection, we present the implemented attacks and properties of the attacker designs. Afterwards, we describe the extensions we made to the existing tools and the benchmarks we used to evaluate our methodology.

5.3.1. Attacks in Lattice FGPAs

To implement and evaluate our approach of checking bitstreams, we use Lattice iCE40 FPGAs, since their bitstreams were partially reverse engineered and an open-source tool for bitstream reversal into verilog code exists [97]. The tool supports the iCE40 LP/HX 1K/4K/8K chips, and we perform all experiments on the iCE40-HX8K-CT256, which is embedded onto the ICE40HX8K-B-EVN breakout board. As the described side-channel and fault attacks were developed on Xilinx and Intel FPGAs, we first reproduce the designs on the Lattice FPGA to be able to evaluate our bitstream checking methodology. In this subsection, we describe the details of our implementations of attacker designs on Lattice FPGAs.

5.3.1.1. Implementation of Fault and Side-channel Attacks

The fault attack from [13] as well as the power analysis side-channel attack from [12] are reproduced. Furthermore, we are able to crash the FPGA as in [10] with a higher amount of ROs, making a power cycle necessary. Both attacks can be combined into a single reference design, shown in Figure 5.9.A. It utilizes about 62% of the iCE40-HX8K resources.

ROs for Fault Attack: Fault attack logic based on ROs through LUTs can be reproduced following the concept in Figure 5.2 (without the counter) and Figure 5.4.

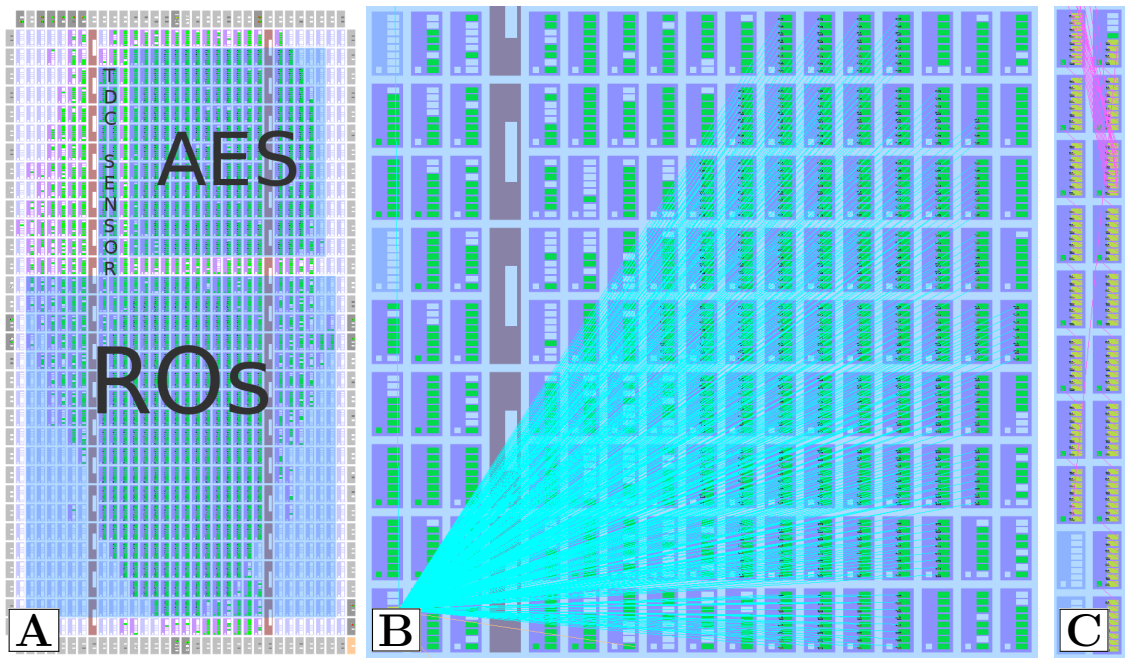


Figure 5.9.: Floorplans of our reproduced design in the Lattice iCE40-HX8K. **A:** Complete design with 1920×ROs, 1×TDC, and 1×AES module; **B:** RO grid with a single node for activation; **C:** TDC-based Voltage sensor.

Floorplanned in the Lattice iCEcube2 tool, they are visualized in Figure 5.9.B. The RO module implementation needs an output signal not to be deleted by optimization algorithms, but the final result mapped onto the FPGA is a single LUT for one RO. We were able to both inject faults into an AES cryptomodule and crash the entire FPGA with a large amount of about 6 000 ROs. From the crash, it resets and loses its configuration bitstream, but can be reprogrammed again from the host PC. As already stated in [10, 13], different oscillator designs may result in a more effective attack, which requires less resources. However, we focussed on investigating different designs for hiding oscillators from detection, as explained in the next subsection.

TDC Sensor for SCA: We select to only reproduce the TDC-based (cf. Figure 5.5) voltage sensor attack, since the detection of RO-sensors has no additional novelty over detecting ROs for fault attacks. It can be implemented in Lattice FPGAs similar to [12], and floorplanned as we present in Figure 5.9.C.

We essentially reproduce the power side-channel attack presented in [12], and put a basic AES-128 module on the Lattice FPGA which uses a 32-bit data-path, and run it at 24 MHz, requiring 50 cycles for one encryption. In the other half of the FPGA, we put the TDC-based voltage sensor, and use a synchronization helper signal to be able to easily analyze our traces. The TDC is clocked at 96 MHz. For the power analysis attack [8], we use the bitwise leakage model from [12] as presented in Equation 2.3 in Section 2.3.

$$P^{(2)}_{\text{mod}}(k_{\text{hyp}}) = \text{sbox}^{-1}(c_i \oplus k_{\text{hyp}}) \wedge 2^b \tag{5.1}$$

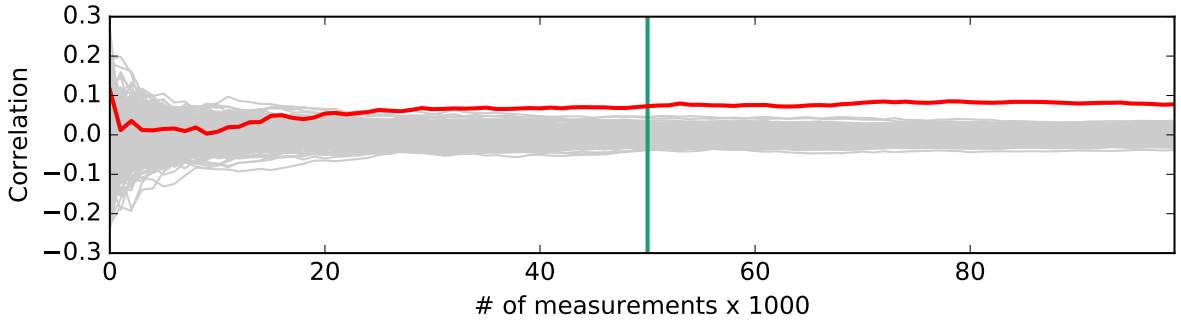


Figure 5.10.: Example progress of successfully recovering a key bit on the Lattice iCE40-HX8K Breakout Board FPGA after about 50 000 traces.

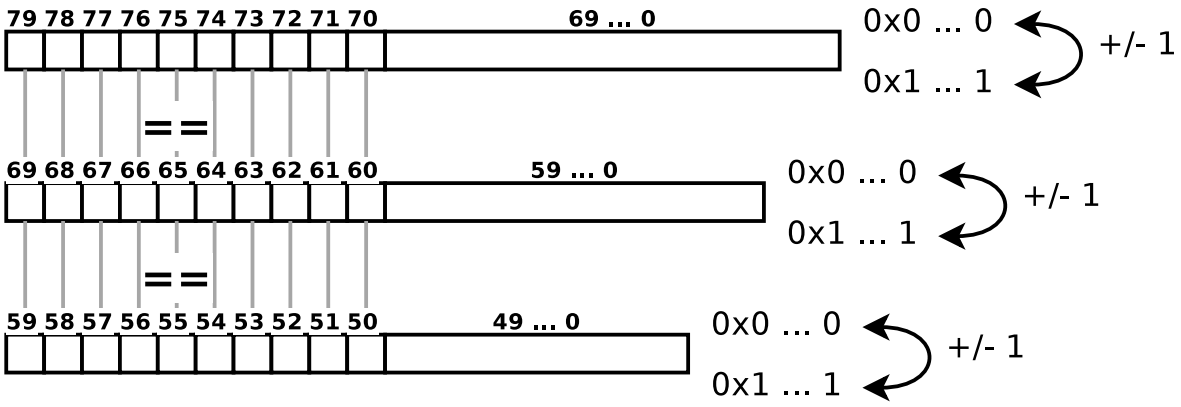


Figure 5.11.: Testing design from [13] to evaluate fault injection with three adders of different length alternating between minimum and maximum value.

Each single bit is used to evaluate a key hypothesis k_{hyp}, based on the correlation of the bit at position b and the corresponding ciphertext byte c_i. In Figure 5.10, we show an example of correlating one bit of each key hypothesis in relation to the number of used measurement traces for correlation. The correct key bit is marked red, while all other key candidates are grey. Over the progress of 0 to 100 000 traces of measurements, the correct key bit is found after about 50 000 traces.

5.3.1.2. New Attack Variants

In the original attacks [10, 13], ROs were used to inject faults or crash the FPGA. For a more thorough analysis and to cover alternative attacker designs, we considered other constructs, which could be used by an attacker to avoid detection in bitstream checking hypervisor programs. Intuitively, sensors have to be based on primitives that exhibit different behaviour dependent on voltage levels. In purely digital designs, only combinational path delay has been investigated as a feasible building block to detect fluctuations through ROs or timing violations. Therefore, we focus only on fault injection variants, that make use of different FPGA primitives.

Fault injection verification design: To quickly check fault injection success, we implement a primitive testing design composed of three adders with alternating value. The basic principle of the testing design is depicted in Figure 5.11, which was also used

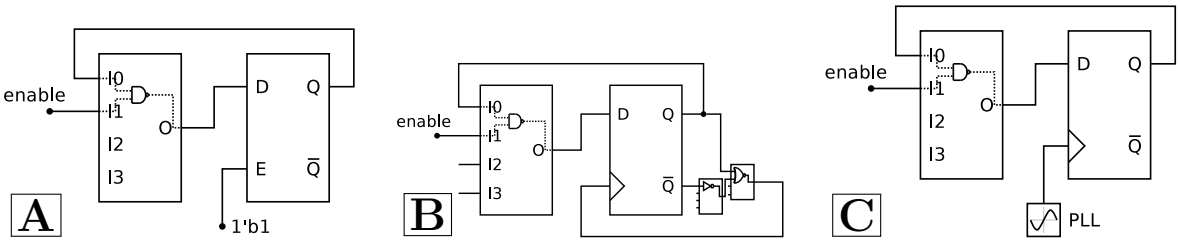


Figure 5.12.: Different variants to generate high switching activity using non-combinational cycles. **A:** Ring oscillator through a transparent latch; **B:** Self-clocked oscillator using glitches; **C:** Clocked oscillator using a Phase-Locked-Loop (PLL) at maximum frequency.

for initial experiments in [13]. Three adders of different length, which are initialized with zero, are constantly incremented and decremented every clock cycle. Meanwhile, the Most Significant Bits (MSBs) are checked for equality. If the MSBs are found to be unequal, an LED is switched on and a specific character is sent through the UART interface of the iCE40-HX8K breakout board. Due to the alternating values, with every clock cycle, a carry bit must be propagated to the MSB of each adder. Whenever a significant voltage drop increases the gate delays enough to prevent the carry bit from propagating to the MSBs of an adder, we can detect a fault on the LED and the UART interface. This testing design is running at an operating frequency where no faults are detected during normal operation.

Hiding oscillators: Several options to generate high switching activity without combinational cycles exist, however, not all of them are suited for every FPGA. In [10], they used transparent latches to create combinational loops through a LUT and a register configured as latch. This method, which is shown in Figure 5.12.A, is not suited for use in the iCE40 FPGAs, since they do not provide registers that can be configured as latches. Another way to potentially induce faults is presented in Figure 5.12.B. Here, the output of the XNOR-gate, which is fed back to the clock input of the register, glitches every time the output value of the register changes due to the signal delay from $\overline{Q}$ through the inverter, making the register capture the next inverted input. In Figure 5.12.C, a PLL is configured to output a very high frequency, clocking a toggling register which acts as a sequential oscillator. We experimentally compare normal ROs with the clocked oscillators in Figure 5.12.C regarding their effectiveness to inject faults into isolated designs. A PLL of the iCE40-HX8K is configured at 240 MHz as a multiple of the 12 MHz base board frequency. We analyze the power consumption of the entire evaluation board during the activity of the combinational ROs and the sequential oscillators from Figure 5.12.C respectively. In this analysis, two designs with 6 000 of either of the oscillators are uploaded to the FPGA. The placement of the oscillators is identical for both designs, and they are toggled with a fixed signal at a slower frequency of 750 kHz and a 12.5% duty-cycle. We measure the current through the entire board by measuring voltage over a 0.1Ω resistor in the USB cable, which powers the system. The current is averaged over 10 measurements, where each measurement is carried out for a $7s$ duration using an oscilloscope. Without activated oscillators, we observe an average current of 210.3 mA for the idle design. When toggling a combinational RO grid, the average current is 291.3 mA, as opposed to 240.9 mA when using sequential

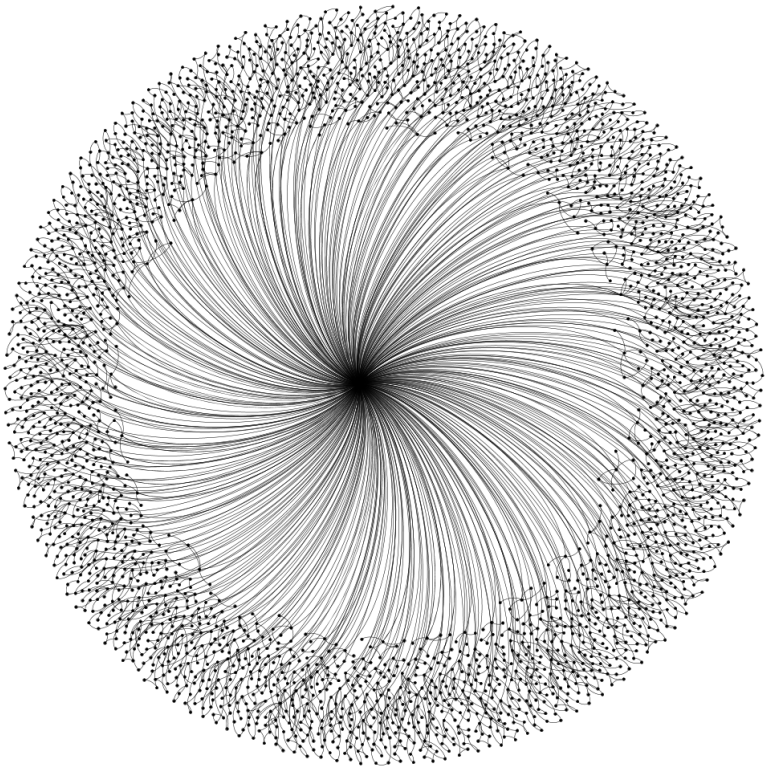


Figure 5.13.: A RO-grid extracted from the netlist graph, rendered using the graph rendering tool gephi [98]. The subgraph shows two proposed malicious signatures: A high fanout node for synchronization and a combinational loop for each RO in the grid.

oscillators. Therefore, although sequentially clocked oscillators are less effective, they can still be used to inject faults into an isolated design, as shown by our experiments. We are indeed able to inject faults in the testing adder design described above using these sequentially clocked oscillators, which can later enable fault attacks.

5.3.2. Toolchain Extensions

To implement our flow described in Section 5.2, we base the bitstream reversal on a tool from the *icestorm* tools [97]. The unmodified *icebox_vlog* tool allows reverse engineering into verilog code from Lattice iCE40 FPGA bitstreams. A reverse-engineered design is functionally equivalent to the original design. However, to analyze timing constraints and other critical structural subcomponents, we require an exact representation of the original design with iCE40 technology library primitives, such as for example LUTs (*SB_LUT4*), carry elements (*SB_CARRY*), registers (*SB_DFF[N/E/S/R]*) or PLLs (*SB_PLL40_**). Therefore, we need to recover the locations of the FPGA primitives from the bitstream and generate a placement constraint file, since placement has a significant influence on the timing of paths. We extended the *icebox_vlog* tool to fit our requirements, which has been open-sourced as well. It now creates a verilog file instantiating the iCE40 FPGA primitives from the original design with the respective

connections. Additionally, a placement constraint file is created, reflecting the original design placement. A limitation is the lack of routing constraints, which do not exist in either the commercial or the open-source software. Production-ready secure bitstream checkers would need to consider routing, which is excluded in this proof-of-concept evaluation.

For a structural analysis of the netlist, we also use the *yosys* open-source synthesis/-analysis tool [99] together with the python graph analysis library *graph-tool* [93]. Yosys allows analyzing the generated verilog source files and convert the netlist into the commonly used GraphViz/DOT format [100]. In Figure 5.13, we show a RO grid, extracted from a netlist graph, which was generated by *yosys* and rendered in the open-source graph editing tool *gephi* [98]. The extracted subgraph exhibits two of our proposed properties of malicious designs: A high fanout node for synchronization and a combinational loop for each RO in the grid. We pre-process the output DOT-file to remove some unneeded information and import the netlist graph using graph-tool. Eventually, various algorithms for graph analysis can be applied, such as finding combinational cycles or determining node degrees.

Although the described tools and methods are specific for Lattice iCE40 FPGAs, which was possible due to the publicly reverse engineered information about those devices, the general approach of reversing bitstreams into HDL code and checking the design for threat signatures can be deployed by all FPGA manufacturers in a similar and more extensive way, when detailed vendor knowledge is available.

In total, our extended toolset allows acquiring the basic dataset from a given bitstream only and check potential attacker designs for the following properties:

- Combinational cycles, which can be used for SCA and fault attacks
- Data-to-clock routings, which can hide more elaborate TDC sensors from timing analysis
- Highest non-clock fanout, to identify a toggling grid
- Timing violations, which can indicate a basic TDC sensor

In Section 5.4 we show the results of checking for the above properties in various designs.

5.3.3. Benign Benchmark Designs

In Section 5.2, we described signatures, which are used in attacker designs to record side-channel leakage or induce faults in victim designs. To show that the described signatures do not restrict legitimate non-malicious designs unnecessarily, we perform the bitstream verification on a multitude of different benchmark designs. Due to the small size and architectural restrictions of the iCE40 FPGAs, only a subset of designs from each collection can be fully synthesized into a bitstream which we can check for potentially malicious signatures. Among others, the IWLS2005 collection, contains designs from the ISCAS'85 and '89 benchmarks [45, 101], the ITC'99 benchmarks [102] as well as from the free available designs at OpenCores [103]. We are able to create

Design	Collection	Description	#LEs
s27	ISCAS'85/'89	unknown	19
s208_1	ISCAS'85/'89	based on programmable logic device	73
s298	ISCAS'85/'89	based on programmable logic device	141
s344	ISCAS'85/'89	resynthesized *s349* without redundancies	132
s349	ISCAS'85/'89	4-bit multiplier	123
s382	ISCAS'85/'89	resynthesized *s400* without redundancies	159
s386	ISCAS'85/'89	controller	167
s400	ISCAS'85/'89	traffic light controller	163
s420_1	ISCAS'85/'89	digital fractional multiplier	187
s444	ISCAS'85/'89	traffic light controller	162
s510	ISCAS'85/'89	controller	256
s526	ISCAS'85/'89	traffic light controller	237
s526n	ISCAS'85/'89	resynthesized *s526* without redundancies	228
s641	ISCAS'85/'89	based on programmable logic device	193
s713	ISCAS'85/'89	based on programmable logic device	192
s820	ISCAS'85/'89	resynthesized *s832* without redundancies	351
s832	ISCAS'85/'89	based on programmable logic device	333
s838_1	ISCAS'85/'89	digital fractional multiplier	367
s1196	ISCAS'85/'89	resynthesized *s1238* without redundancies	483
s1238	ISCAS'85/'89	combinational circuit with random flip-flops	586
s1423	ISCAS'85/'89	unknown	607
s1488	ISCAS'85/'89	resynthesized *s1494* without redundancies	636
s1494	ISCAS'85/'89	controller	643
s5378	ISCAS'85/'89	unknown	1294
s9234_1	ISCAS'85/'89	based on real chip	974
s13207	ISCAS'85/'89	based on real chip	1219
s15850	ISCAS'85/'89	based on real chip	685
b01	ITC'99	FSM comparing serial flows	86
b02	ITC'99	FSM recognizing BCD numbers	45
b03	ITC'99	resource arbiter	385
b04	ITC'99	min/max computation	881
b05	ITC'99	memory access	1021
b06	ITC'99	interrupt handler	86
b07	ITC'99	counts points on a straight line	738
b08	ITC'99	finds inclusions in a sequence of numbers	227
b09	ITC'99	serial converter	260
b10	ITC'99	voting system	311
b11	ITC'99	scrambles string with variable cipher	805
b12	ITC'99	single-player game	2017
b13	ITC'99	sensor interface	402
sha	VTR	SHA1 hash computation	1833
diffeq2	VTR	differential equations mathematics	4049
vexriscv	-	RISC-V processor system	2587
leon3	-	SPARC-V8 processor system	29304

Table 5.1.: All benign benchmark designs, used to evaluate our bitstream checking approach, with their respective amount of required logic elements (#LEs). Functional descriptions for ISCAS'85/'89 designs are taken from [45].

bitstreams of the ISCAS'85/'89 benchmarks, except for the *s38584*, *s35932* and *s38417* designs. Out of the ITC'99 collection we investigate all except the *b14-b22* design variants. From the Verilog-to-Routing (VTR) benchmarks [83, 84], we evaluate the *sha* and *diffeq2* benchmark designs, which fit on the iCE40-HX8K without extensive modifications. A RISC-V processor implementation for the iCE40 [85] is evaluated as well. To show scalability of the approach, we additionally evaluate a complete LEON3 quad-core processor system [86]. Since this design does not fit on the iCE40-HX8K FPGA, we omit full bitstream compilation and only evaluate a flattened and technology-mapped netlist directly for structural signatures. A timing-analysis on the LEON3 softcore system can not be performed either. In Table 5.1 we provide an overview of all evaluated benchmarks together with short functional descriptions, if available. We also detail the amount of logic elements (#LE) each design occupies after mapping to iCE40 technology primitives.

We will show, how our bitstream checking methodology performs on these benchmarks and use the information to establish thresholds to detect malicious designs. The results suggest that this methodology can be applied to more complex designs and larger FPGAs from different manufacturers as well.

5.4. Results

We present here the results of running our bitstream checking flow, which we detailed in Section 5.2 on legitimate designs from various benchmark collections as well as on the designs we implemented to reproduce known attacks on the iCE40-HX8K FPGA. The Python implementation of our verification runs on an Intel i7-7700HQ notebook device.

5.4.1. Evaluation on Benign Benchmark Designs

In Section 5.3.3, we enlisted a subset of benchmarks from various design collections, which we were able to map on our testing device and evaluate using our bitstream checking implementation.

Table 5.2 presents the results of our evaluation. First, we notice that all evaluated benign designs do not make use of combinational cycles or data-to-clock paths. Moreover, the iCEcube2 does not report any timing violations for any design. When investigating node fanouts, we see that the *sha* design has the largest maximum fanout of all designs, not considering the *leon3* design, which we can not fit on the iCE40 devices. The maximum fanout of the *leon3* design corresponds to 6.3% of the entire design. Thus, its fanout w.r.t. the device resources can also not reach more than 6.3%. A fanout of 969 corresponds to 12.6% of the available device resources. The evaluation of malicious attacker designs in the next subsection shows that a toggling grid for fault injection has a significantly higher maximum fanout and can thus be detected with a simple threshold. Furthermore, the analysis for maximum fanout nodes can be further improved, if the FPGA hypervisor is able to define exclusive reset inputs or generators. As described in Section 5.2.1, we disregard nodes for fanout analysis, if they can be backtracked to valid board-level clocks or clock generators. The maximum-fanout-node

Design Name	#LEs	Comb. Cycles?	Data-to-Clock?	Highest Fanout?	Timing violations?	Runtime (structural)	Runtime (timing)
s27	19	×	×	3	×	4.54*s*	17.34*s*
s208_1	73	×	×	8	×	4.60*s*	17.66*s*
s298	141	×	×	14	×	4.63*s*	18.33*s*
s344	132	×	×	18	×	6.21*s*	18.56*s*
s349	123	×	×	18	×	6.24*s*	18.51*s*
s382	159	×	×	22	×	6.20*s*	19.11*s*
s386	167	×	×	13	×	4.57*s*	19.33*s*
s400	163	×	×	22	×	6.16*s*	18.85*s*
s420_1	187	×	×	16	×	4.59*s*	19.21*s*
s444	162	×	×	22	×	6.19*s*	19.61*s*
s510	256	×	×	26	×	4.76*s*	20.52*s*
s526	237	×	×	22	×	6.22*s*	18.51*s*
s526n	228	×	×	22	×	6.15*s*	18.27*s*
s641	193	×	×	14	×	4.73*s*	19.83*s*
s713	192	×	×	14	×	4.63*s*	19.66*s*
s820	351	×	×	42	×	4.84*s*	20.32*s*
s832	333	×	×	25	×	4.80*s*	20.29*s*
s838_1	367	×	×	33	×	6.38*s*	20.18*s*
s1196	483	×	×	42	×	6.83*s*	29.33*s*
s1238	586	×	×	44	×	7.31*s*	22.70*s*
s1423	607	×	×	75	×	6.74*s*	21.91*s*
s1488	636	×	×	46	×	5.29*s*	24.04*s*
s1494	643	×	×	46	×	5.60*s*	23.44*s*
s5378	1294	×	×	156	×	7.58*s*	27.23*s*
s9234_1	974	×	×	130	×	7.10*s*	25.06*s*
s13207	1219	×	×	210	×	7.17*s*	26.43*s*
s15850	685	×	×	129	×	6.67*s*	22.17*s*
b01	86	×	×	10	×	4.74*s*	17.91*s*
b02	45	×	×	4	×	4.53*s*	17.26*s*
b03	385	×	×	30	×	6.25*s*	19.13*s*
b04	881	×	×	66	×	8.33*s*	21.59*s*
b05	1021	×	×	36	×	6.70*s*	22.17*s*
b06	86	×	×	8	×	4.82*s*	17.82*s*
b07	738	×	×	44	×	6.35*s*	19.86*s*
b08	227	×	×	21	×	6.35*s*	18.74*s*
b09	260	×	×	32	×	6.27*s*	18.35*s*
b10	311	×	×	24	×	6.42*s*	18.64*s*
b11	805	×	×	35	×	6.51*s*	20.02*s*
b12	2017	×	×	119	×	7.13*s*	24.52*s*
b13	402	×	×	50	×	6.36*s*	19.88*s*
sha	1833	×	×	969	×	15.29*s*	69.13*s*
diffeq2	4049	×	×	97	×	19.78*s*	111.06*s*
vexriscv	2587	×	×	241	×	23.86*s*	87.49*s*
leon3	29304	×	×	1838	–	19.28*s*	–

Table 5.2.: Results of checking bitstreams, which have been generated from the collection of benign designs, for malicious signatures.

in the *sha* design is used as a reset node, which could be predefined by the hypervisor on a specific pin of the FPGA. This methodology would allow users to implement proper reset mechanisms and further narrow down the fanout analysis to not overly restrict benign designs. When further analysing the *sha* design, we find the second-highest fanout node to have a fanout of only 255, which corresponds to only 3.3% of the device resources.

We split up the runtime of our entire flow and specify the duration of structural and timing analysis separately. The runtime of the timing analysis is significantly larger, than the time needed for the structural netlist graph algorithms. This is due to the fact that we use the vendor provided timing analysis tool, which requires a full recompilation of the entire design to report timing violations. A bitstream checking tool implemented by the FPGA vendor, would be able to reduce this time significantly, by evaluating the flattened netlist and placement constraints directly.

The structural analysis takes a maximum of about 24*s* for our benchmark collection, which we consider reasonable in a FPGA-accelerated execution environment. Bitstreams will be uploaded at program start and will most probably not be changed during the entire execution time. Furthermore, we believe the process can be accelerated with detailed vendor knowledge of the FPGA technology, especially regarding the bitstream reversal.

5.4.2. Evaluation on Reproduced Attack Bitstreams

To evaluate the detection of actual malicious designs, we run the bitstream checking flow on our various attacker designs, which we presented in Section 5.3.1. In total, we implemented three different design containing different kinds of malicious signatures for fault and side-channel attacks:

- *reference01* is a reference design for both fault injection with ordinary RO-grids and SCA with TDC sensors. The design contains in total 3 000 ROs, an AES encryption module and a single TDC sensor, based on a long delay line.
- *reference02* is able to crash the FPGA device with a large amount of 6 500 ROs, making a power cycle and reprogramming necessary. A simple LED blinker has been added to verify the board status.
- *reference03* uses the alternative sequentially clocked oscillators we presented in Section 5.3.1.2 as well as the fault verification adder design from [13].

Table 5.3 summarizes the results of checking the respective attacker designs using our bitstream checking methodology.

We can see that the node with the highest fanout is in direct control for synchronizing the oscillator grid, as the fanout corresponds to the amount of ROs used in the respective design. Moreover, the highest fanout is significantly higher, than the highest fanout in the benign benchmark designs in the previous subsections. A fanout of 3 000 corresponds to 39.06% of the total amount of logic cells in the iCE40-HX8K compared to only 2.7% for the benchmark designs. For instance, to prevent the implementation of malicious logic, a hypervisor for a multi-tenant iCE40 device may define a fanout

Design Name	#LEs	Comb. Cycles?	Data-to-Clock?	Highest Fanout?	Timing violations?	Runtime (structural)	Runtime (timing)
reference01	6075	✓	×	3000	✓	29.32s	152.31s
reference02	6810	✓	×	6500	×	20.32s	169.67s
reference03	4077	×	×	3000	✓	25.09s	127.05s

Table 5.3.: Results of checking bitstreams of implemented attacker designs for malicious signatures. *reference01* contains both ROs and a TDC sensor, *reference02* can crash the FPGA device with a large amount of ROs and *reference03* makes use of sequential oscillators to inject faults into the simple adder design from [13].

threshold of 15% of the device resources. In the previous subsection, we explain how this threshold could be defined more accurately by establishing designated reset signals on specific board pins, which are controlled by the hypervisor.

In the *reference01* design our bitstream checker reports timing violations, due to the presence of the TDC sensor. From the correct identification of the combinational cycles and fanout nodes, we deduce the correct function of our structural analysis algorithms. As we want to show the successful bitstream reversal of logic elements and placement recovery as well, we manually analyze the reversed and recompiled design and the results of the timing violation analysis in the first reference design. This manual analysis also allows us to evaluate the impact of lacking the possibility to recover and redefine routing constraints in the reversed design. Therefore, we compare the results of compiling the original and reversed design and their respective timing reports. After compilation of the original design files, the following most critical path is found, which has the final node of the 64bit delay line at its end:

```
Start            clkin
End              sense_inst.line_63_
                 _genblk1_linelatch_LC_9_20_7/in3
Reference        clk60mhz
Setup Constraint 8333(p)
Path Slack       -6319(p)
```

We unpack the attacker bitstream and generate a single verilog netlist file along with placement constraints for every single node. After resynthesizing the generated verilog and applying timing analysis, we acquire the following report:

```
Start            clkin
End              n19_inst_LC_9_20_7/in3
Reference        PLL_16_0/PLLOUTGLOBAL
Setup Constraint 8333(p)
Path Slack       -4622(p)
```

The path end node is denoted by a numeric label *n19*, which is why we verify the result in the floorplan. Figure 5.14 shows the critical path end node in the original floorplan, which is identical with *n19* in the reversed floorplan in Figure 5.15. Subsequently, we can say that the reversed design conforms to the original design with

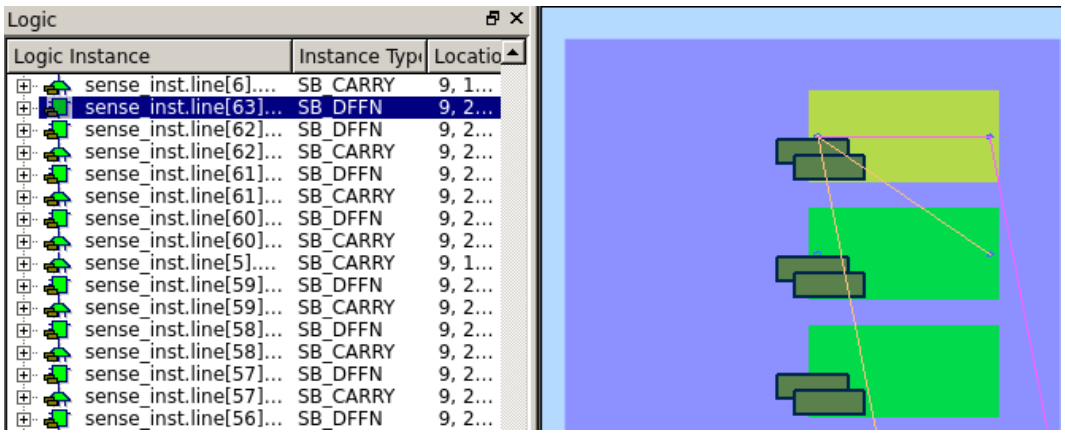


Figure 5.14.: End node of most critical path in the original design (yellow)

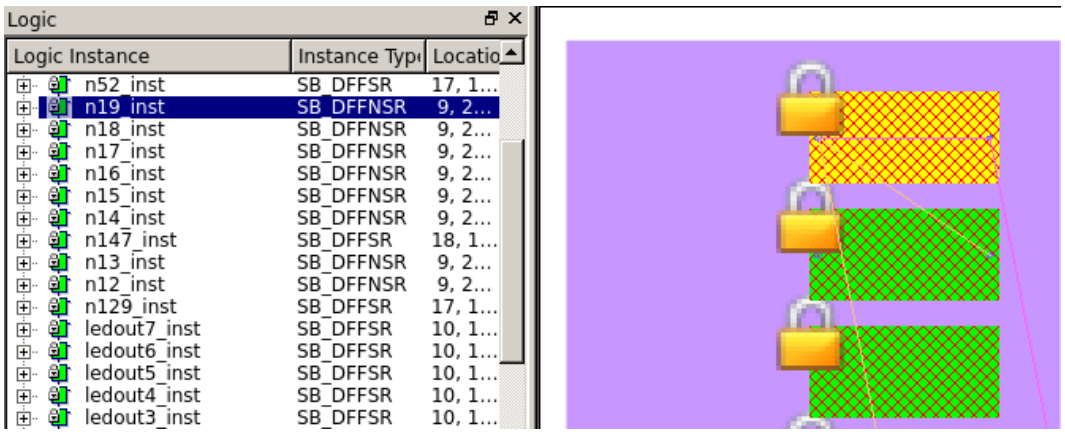


Figure 5.15.: End node of most critical path in the reversed design (yellow)

enough accuracy, to identify malicious TDC sensing circuitry. We do note, however, that the path slack values of the original and the reversed design differ, either due to the lack of reversed routing constraints or because of varying randomized heuristics applied by the iCEcube2 timing analyzer.

All in all, we can conclude that rejection of a candidate bitstream upon detecting any of the following signatures is an appropriate way to increase security on our iCE40-HX8K device:

- Combinational cycles
- Data-to-Clock paths
- Maximum fanout higher than 15% of the device resources
- Timing violations

The iCE40-HX8K is by no means suited for multi-user access and serves only as an example application of our methodology. For different devices and technologies, significant modifications to the rejection criteria and detection algorithms may be required. Furthermore, our list of signatures is not exhaustive, and other structures that can

be used to perform an attack on another logically isolated design exist, as presented in Section 7.2.1.

5.4.3. Discussion

We show a first approach that tries to detect all possibilities in which FPGA logic in a multi-tenant FPGA can be re-used maliciously on the electrical level, to either inject faults, or measure voltage fluctuations to perform a side-channel attack on a victim circuit. Since both of these aspects require quite atypical use of FPGA primitives, our approach will still allow most normal FPGA designs, which we prove by evaluating multiple benign benchmark designs. We present a generic flow, that extracts a collection of different data from the original bitstream only, on which an arbitrary variety of algorithms to detect malicious signatures can be executed.

To mitigate fault attacks we describe fundamental properties of designs that can toggle a vast amount of oscillating elements – such as ring oscillators – repeatedly and in a synchronized manner. We found that combinational cycles, which are required for implementing ring oscillators, are typically not used in standard sequential FPGA design. Thus, if we restrict them, the risk of false positives should be rather low and just affect very specialized designs. Limiting the highest fanout of a design, can restrain an adversary from deploying large grids of toggling elements, which we demonstrate can even inject faults using sequential oscillators and no combinational cycles. In larger designs, a fanout limit may be too restrictive. However, maximum fanout nodes often correspond to clock or reset signals, which could be specifically defined at the board level by the hypervisor and white-listed in the fanout analysis. A more thorough approach needs to be developed in future work, for instance by including an analysis on transient power consumption, based on simulation with test patterns or random input stimuli. This approach would also capture the most recent attacks, such as the ones we present in Section 3.1, which are based on seemingly benign logic modules. With the aid of more detailed low-level device data, that is available to FPGA vendors, a solution to the problem will be much easier to implement.

We show, how different sensor variants can be detected by evaluating timing analysis reports and investigating designs for suspicious paths, fed into clock input pins of FPGA primitives. In our collection of benchmark designs, no data-to-clock paths are found, which leads us to believe that a restriction on them will not restrict benign designs excessively. Timing analysis, on the other hand, might be too restrictive in our basic variant and the risk of false positives is higher than disallowing combinational cycles. Thus, we think that more sophisticated approaches are still required for larger designs in which, for instance, clock-domain crossings or dynamic clock control are required. In larger designs, synchronization and respective structures are usually handled by the designer, who will mark appropriate false path constraints. Since our approach cleans a design for all constraints, the respective synchronization structures will inherit all timing constraints from the basic system clock, and would again be detected with timing violations in the clock-domain crossing. Nevertheless, a simple white-listing of such synchronization structures is probably not the correct and easy way to go, since they might still allow measuring voltage through unconventional re-use. Dynamically adjustable clocks have already been shown to be a security problem in processor-based

systems, where too much power management control can lead to a compromise of the entire SoC [33]. Thus, each step to loosen any of the shown restrictions has to be handled with high care.

5.5. Conclusion

FPGAs offer a high degree of design freedom to allow implementation of arbitrary designs in hardware. This freedom can also be exploited by any user with access to a shared multi-user FPGA to launch power-based side-channel or fault attacks on an entire system through the electrical level. A limitation of this freedom is required in order to prevent these type of attacks, by restricting a user from programming potentially malicious (partial) bitstreams to FPGA fabric. In this chapter, we perform a first analysis on which fundamental restrictions are required and propose a flow how these can be enforced using bitstream checking. Several comprehensive attack signatures are formulated to be checked on technology mapped netlists, and we show how these are applied to three known attacker designs. Furthermore, we evaluate our methodology on a variety of benign benchmark designs and ensure that no false positives are detected. Finally, we discuss the possible limitations of these signatures and give research directions for future work, that will lead to both more secure designs and reduce the restrictions put on legitimate designs.

6. Novel Hiding Approaches against On-Chip Side-Channel Attacks

The work on a dynamic hiding approach through RO-based fences has been published in [25] together with Dennis Gnad, Falk Schellenberg, Amir Moradi, and Mehdi Tahoori, whereas the evaluation of deep neural networks for hiding against side-channel attacks has been published in [26] together with Mehdi Tahoori.

6.1. Active Fences against Voltage-based Side Channels in Multi-Tenant FPGAs

As already explained in Chapter 1, side-channel attacks can be performed on multi-tenant FPGAs by placing a voltage sensor within the same FPGA-fabric without any logical connection, capturing fluctuations on the PDN [12, 14]. However, due to their passive and solely observing nature, such attacks cannot be easily detected within the system. Instead, countermeasures against power analysis attacks try to reduce the information that can be gained from measurements to a minimum, categorized in two general groups of *hiding* and *masking* countermeasures.

Masking schemes are implemented on the algorithmic level and focus on randomizing internal values to detach the power consumption from the actual secret data being processed. Especially for non-linear functions, this change comes with a large overhead. Furthermore, implementing such countermeasures is challenging and closely tied to the algorithm to be protected. Instead, hiding aims at reducing the SNR at the electrical level, i.e., either by raising the noise floor using additional noise sources or by equalizing the instantaneous power consumption. For the latter, many schemes use some variant of dual-rail precharge logic for equalization [104]. Unfortunately, this barely works on ASICs due to manufacturing tolerances and cannot be applied directly to FPGAs. Although some recent work duplicated and inverted the whole circuit [66], the very coarse grained access to FPGA resources hinders a straight-forward implementation of such schemes [105]. In practice, hiding and masking can be used side-by-side to increase the security level significantly [105].

Whereas existing work on countermeasures considers an external attacker, there have been no results against an on-chip attacker. Note that passive fences to enforce logical isolation (cf. [44, 81, 106]) are entirely ineffective against attacks on the electrical level. In the previous Chapter 5, we also suggest to check for potentially malicious design signatures, such as voltage sensors, in user bitstreams before permitting the upload to a multi-tenant FPGA. However, this bitstream checking methodology must

be updated continuously to account for new ways of sensor implementation. Here, we propose an active fence on the electrical level instead that adaptively equalizes the power consumption that is visible to other tenants on the FPGA. Our solution uses ROs as an artificial source of power consumption and voltage fluctuations, whenever needed as indicated by an internal in-fabric voltage sensor. By incorporating the capabilities of an attacker to counteract the side-channel leakage on-the-fly, we achieve some form of inherent symmetry between attacker and defender. Furthermore, the countermeasure is independent of the algorithm to be protected or its implementation.

Indeed, our results show that fencing the circuit to be protected performs significantly better than placing the same amount of ROs randomly. The number of required traces can be increased even further when enabling the feedback by the internal sensor. This is the first mitigation approach that is specifically tailored against on-chip side-channel leakage in multi-tenant FPGAs.

Overall our contributions can be summarized as follows:

- First mitigation method against on-chip voltage side-channel leakage in multi-tenant FPGAs.
- The hiding strategy works without changes to the cryptographic implementation and is independent of the implementation or algorithm to be protected.
- Two to three orders of magnitude leakage reduction.
- The hiding strategy can be applied to various FPGA architectures and devices from various vendors.

In the remaining section, we first present the considered threat model and put our approach in the context of side-channel countermeasures in general. Next, our methodology is explained in Section 6.1.3, followed by our experimental setup in Section 6.1.4. Finally, we show our results in Section 6.1.5 and conclude in Section 6.1.7.

6.1.1. Adversary Model

We follow the well-accepted adversary model that was previously described in various works on electrical level threats in multi-tenant FPGAs [10, 12–14], visualized in Figure 6.1. The adversary in this model can program an arbitrary design in a partial region of an FPGA, which is shared with at least one other victim user. There are no logic or clock signal connections between the users (i.e. tenants) of the FPGA. This isolation is ensured with empty slices put as fences between the users, as suggested by previous design practices for security in FPGAs [44, 81, 107]. Thus, the remaining connections are through the power distribution network, and all possible attack vectors are side-channels. In this work, the goals of the attacker are to impact the confidentiality [12, 14], for which we present a mitigation method. Previous works also considered attacks on integrity [13] or availability [10] in the same model.

Affected systems can be SoCs or high performance computers in the cloud, utilizing FPGAs as custom accelerators. The assumed adversary uses FPGA logic to integrate a sensor to record side-channel information on the electrical level, i.e. voltage

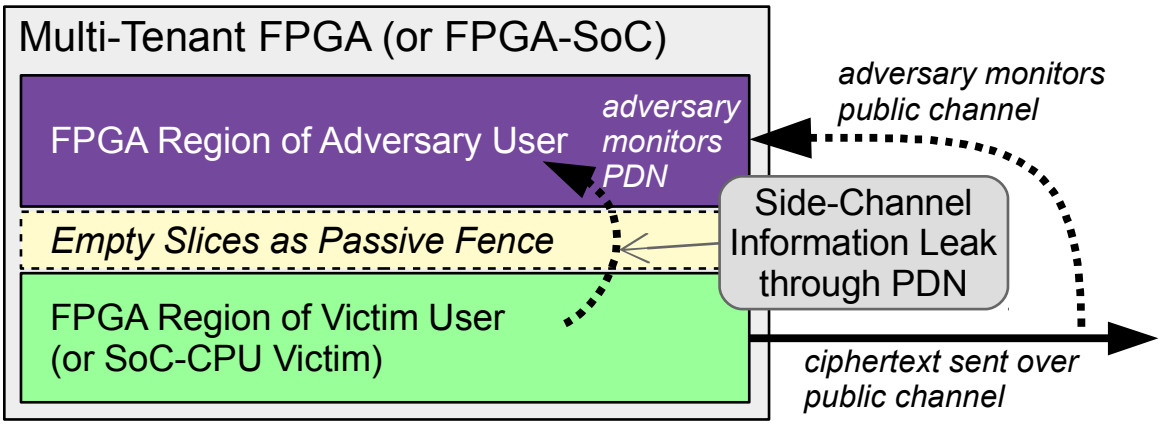


Figure 6.1.: Adversary model for a multi-tenant FPGA or FPGA with shared SoC Logic, such as ASIC-logic CPUs. Based on the models previously shown in [12, 14].

fluctuations. These voltage fluctuations can be analyzed to extract secret keys from cryptographic circuits, implemented in another part of the FPGA or SoC. In a possible real-world scenario, a victim user encrypts plaintext messages using any cryptographic core in a part of the FPGA, and sends out the encrypted ciphertext messages over a public communication channel. The adversarial user which monitors the public channel can then use the measured voltage fluctuations together with the ciphertext to extract the secret encryption key, and subsequently sensitive encrypted information.

6.1.2. Side-Channel Attack Countermeasures

There is ongoing research to explore potential countermeasures against side-channel attacks, following the two directions of *masking* or *hiding*: Masking schemes are implemented on the algorithmic level and separate internal values into multiple shares [108]. Only the combination of all shares will again reveal the correct secret value. Ideally, the operations are performed individually on each share. Since this is not possible for non-linear functions, ensuring the correctness of the computation requires a large algorithmic overhead. Furthermore, such a protection has to be developed individually for each algorithm to be protected, and its actual implementation is challenging as well. Masking techniques can be attacked with higher order attacks, i.e., using higher statistical moments [109]. However, for such attacks, the noise is amplified exponentially with increasing order, resulting in an increased number of required traces.

On the other hand, hiding aims at reducing the SNR at the electrical level. One option is to increase the noise level through additional sources of noise. Especially on FPGAs, implementing generic noise generators has been studied using shift registers, BRAM write collisions, or short-term short circuits [58]. Correlated noise generation was suggested to hide leakage from an FPGA AES implementation in [71]. Likewise, [58] proposes clock randomization to temporally spread the side-channel information. Another variant of hiding reduces the strength of the signal by equalizing the instantaneous power consumption. In the past, hiding was often implemented using a variant of dual-rail precharge logic [104]. While in theory, this results in a balanced power consumption, perfectly balanced computation paths are hard to achieve in practice due to

manufacturing process variations. This problem is amplified on standard FPGAs as the physical realization of elements is unknown. Some recent work proposed duplicating and inverting a cryptographic circuit [66]. Yet, this is still difficult to achieve on FPGAs, since on-chip resources can only be used very coarse grained [105]. More lightweight and feasible approaches achieve power equalization on FPGAs to some limited extent using ROs. However, previous approaches were not designed with side-channel attacks in mind and are thus not evaluated in a security context so far [110].

Summarizing the existing solutions, they are often specific to a particular cryptographic algorithm and require significant changes to its implementation. Our active fence aims to overcome this disadvantage but can also be applied in addition to any of the more sophisticated, specific countermeasures, to further increase protection against side-channel attacks.

6.1.3. Methodology

On-chip power analysis side-channel attacks are based on the fact that a cryptographic module injects voltage fluctuations into the power grid, depending on its data-dependent activity. These fluctuations travel very quickly through the on-chip power grid [11, 111], and reach other parts of the chip. The effect they have on path delays can then be observed by an attacker using the delay-line sensors described in Section 2.6.

The same underlying mechanisms can be used to add additional voltage fluctuations into the system, to change what can be observed at the side of the adversary. Our proposed methods try to reduce the overall usable SNR in the voltage fluctuations traveling from the cryptographic victim module to the adversarial sensor by putting active fences between the modules. We base these active fences on ROs, as controlled primitives, that can be enabled and disabled. During the time they are enabled, they have a high switching activity f_{sw} that leads to a high current $I(t)$. Thus, with only small area use they can efficiently inject a high voltage drop into the power grid to equalize the outgoing voltage fluctuations (i.e., the information leakage) from the cryptographic module. Just like other countermeasures from the *hiding*-category, we evaluate these active fences to either increase the noise, or reduce the signal. Overall, this is evaluated based on the number of traces required for a successful CPA attack.

In Figure 6.2, we present a general overview of the proposed method. The active fence is placed as a RO region around a design, that is critical to security of an application, to prevent side-channel leakage to partitions of other users. Various strategies of placing and activating the ROs within the fence region are feasible. ROs can be activated randomly or by using a TDC sensor for controlled activation. We will discuss those strategies in the following subsections and show experimental results for some combinations.

6.1.3.1. Placing Ring Oscillators

To equalize the voltage fluctuations from the cryptographic module, various choices to place ROs are possible. For external power analysis, an arbitrary placement of ROs may be sufficient. However, due to the strong tempo-spatial dependency of on-chip voltage fluctuations [11, 111], it is important how the ROs are placed, and in which

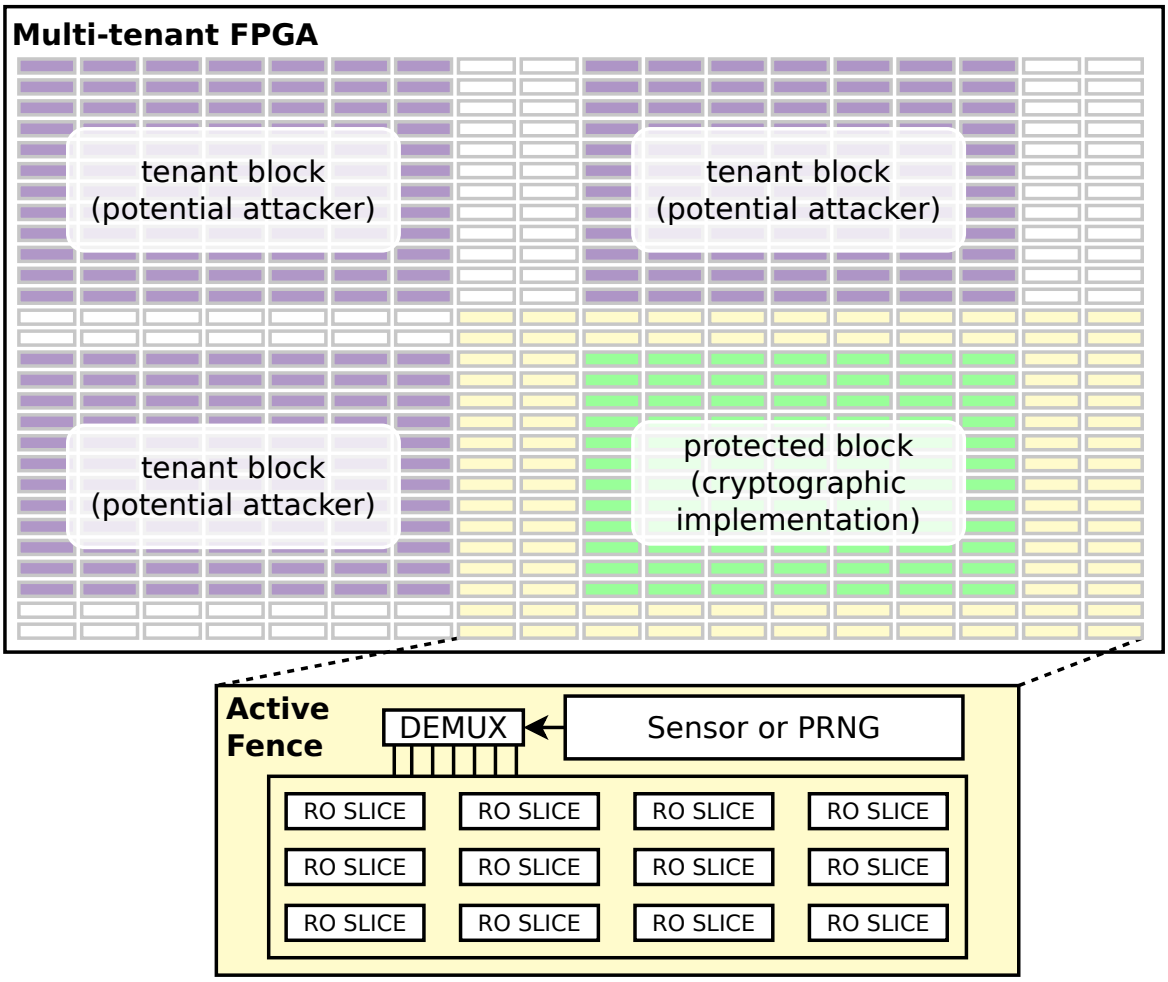


Figure 6.2.: Overview of the active fence as a protective countermeasure between a design, which is used in a security-related application, and other users on a multi-tenant FPGA. A specific amount of ROs is activated dependent on the value of a sensor or a PRNG.

order they get activated. This plays an important role in suppressing the side-channel leakage an adversary can observe.

Additionally, we should consider reducing the impact this countermeasure has on all benign users of the multi-tenant FPGA, i.e. ideally it should be placed outside the user-assigned reconfigurable regions. With all these considerations, our choices can be broken down to two basic options:

1. Constrain ROs to a region between attacker and victim in which the place-and-route toolchain places them heuristically.

2. Map the ROs precisely into a densely packed uniform array between attacker and victim, to activate them in a specific spatial order.

When placing the RO array heuristically into the constrained region, activation happens at arbitrary locations within that region. For the second option, the ROs are activated row-by-row or column-by-column, depending on the layout of the entire system.

In Figure 6.3 we show a simplified example of the entire design when allowing the place-and-route toolchain placing ROs heuristically. Depending on the input value, we activate a larger or smaller amount of ROs in the active fence region to mitigate side-channel leakage from the victim partition at the bottom of the figures into the attacker region at the top. Figure 6.4 outlines the principle of a constrained row-by-row RO placement and activation.

6.1.3.2. Activation Strategies

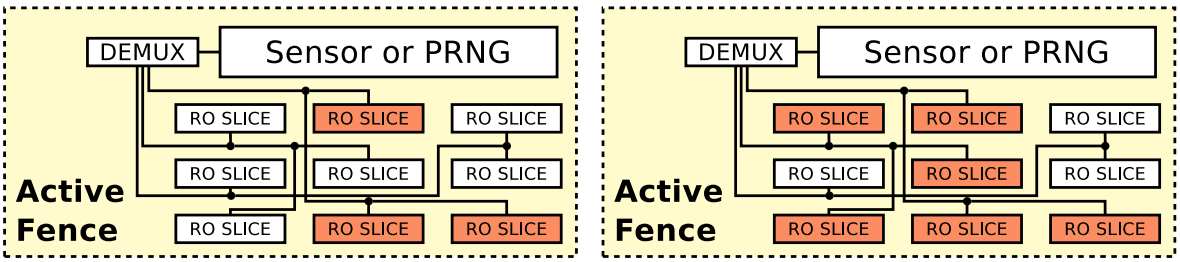
In addition to the placement layout of the ROs regarding victim and adversarial blocks, it is also important to decide on an activation strategy. Generally, we intend to lower the supply voltage when activating a larger amount of ROs and raise the voltage when activating a smaller amount. The exact amount is technology dependent and can be experimentally evaluated, such that enabling all ROs can sufficiently out-balance the worst-case voltage fluctuations emitted from the cryptographic module. It must be decided how these ROs will be activated at runtime — both spatially and temporally. An activation strategy can fulfill one or both goals of either canceling out the fluctuations caused by the cryptographic module, or to add overall more noise to the system.

In our experiments we consider two strategies as depicted also in Figure 6.2:

1. Random activation patterns, where the amount of activated ROs is based on a PRNG output, implemented as a Linear Feedback Shift Register (LFSR).

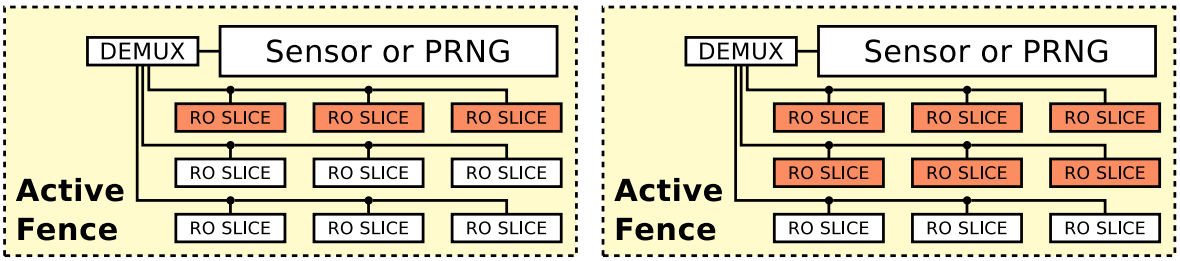
2. Activating an amount of ROs depending on the value of a voltage-fluctuation sensor, similar to the sensor used by the attacker, however at the victim side.

A random activation fulfills the objective of increasing the noise in the system, increasing the amount of required traces for a successful attack. When activating according to a sensor value, we aim to flatten the voltage fluctuations caused by the cryptographic module, thus reducing the leakage to the attacker as well. In our experiments, we find a sensor-based activation to be slightly more successful in weakening the attack.



(a) One third of ROs activated (b) Two thirds of ROs activated

Figure 6.3.: RO activation pattern when letting the toolchain decide the placement heuristically



(a) One third of ROs activated (b) Two thirds of ROs activated

Figure 6.4.: RO activation pattern when fixing RO placement as a row-by-row grid

6.1.4. Implementation and Experimental Setup

Our experimental setup is based on the open-source FPGA development board *Radiona ULX3S* [112] in a version that integrates a Lattice ECP5 12F FPGA with 12K LUT elements with various other components such as an Espressif ESP32 IoT microcontroller module. Although Xilinx and Intel FPGAs are used more widely, we choose to initially develop our design on a Lattice device, which has great open-source support [113] that will be useful in future research. Nevertheless, our proposed methodology is assumed to be easily adaptable to other FPGAs and platforms.

Figure 6.5 shows the overview block diagram of the setup. The Lattice ECP5 on the left side contains three modules. In the bottom part, a hardware AES module is integrated, which is connected to the outside to receive plaintexts, encrypt them, and send the ciphertext back to the connected workstation PC. This module resembles the victim user, out of two users of the system. For the connection, we use a simple UART module on the FPGA, connected to a USB-to-UART Bridge on the Radiona ULX3S Board. In the top part of the ECP5, the adversary user implements his logic, based on a delay-line sensor, specifically tailored to the ECP5 primitives, as we describe in Section 6.1.4.2. Between the two users, we implement the different variants of our active fence, as described in Section 6.1.3. Both the AES module and the active fence to protect it are clocked from the same on-chip PLL clock generator, whereas the adversary uses another PLL. On the board level, both attacker and victim PLLs are connected to the same onboard 25 MHz clock generator.

In Figure 6.6 we show how the design is mapped onto the ECP5 FPGA as seen in the Floorplan View of the Lattice Diamond design software. To make our results

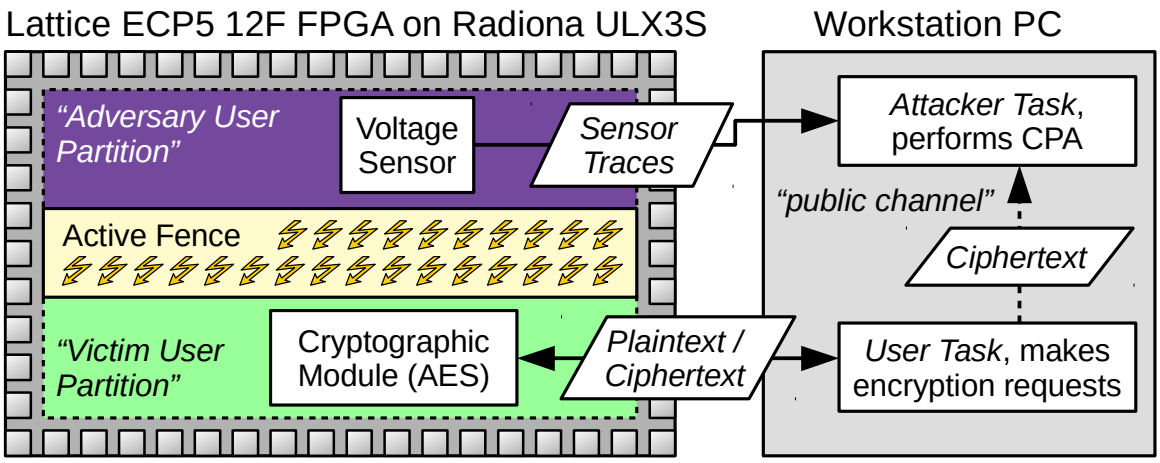


Figure 6.5.: Experimental setup overview showing the Lattice ECP5 FPGA connected to a workstation PC. The AES module is used to encrypt messages, while an attacker can use the ciphertexts together with sensor traces to perform CPA.

comparable, we fix the placement of the AES module as well as the attacker TDC sensor. This experimental setup follows roughly what has been presented in [12] and [14], except for the added active fences.

6.1.4.1. AES implementation

We use industry's standard symmetric encryption scheme AES in its 128-bit variant to evaluate the performance of our protection against side-channel analysis. However, note that our proposed active fence is independent of the exact algorithm or implementation to be protected. To keep comparability, we use an AES module that follows the same design principle as explained in [12], using a 32-bit datapath. In each clock cycle, four of the 16 state bytes are processed in parallel, as visualized in Figure 6.7. One round thus takes four clock cycles and one additional clock cycle to perform the key schedule sharing the Sboxes (not depicted). The module is operated at a clock speed of 12.5 MHz from the same clock generator as the active fence.

6.1.4.2. Sensor Implementation in Lattice ECP5

Previous publications have mostly used Xilinx FPGAs [12, 14, 114], but we also report in Section 5.3.1 how the attack is successful on a Lattice iCE40. We implement TDC sensors as explained in Section 2.6 for both offensive and defensive design parts on the Lattice ECP5, using the available carry-chain primitives.

Carry-chains on the ECP5 are instantiated through *CCU2C* primitives, each of which represents a 2-bit carry element with carry-in, carry-out and two bits of summation signals. Connecting the carry-in of one *CCU2C* element to the carry-out of another element automatically enforces an adjacent placement of the two elements.

We route the sensor clock signal through an initial delay carry-chain into the main chain, which is eventually sampled into registers at the falling sensor clock edge. A specific location is set for the first elements of the initial and the main carry-chain

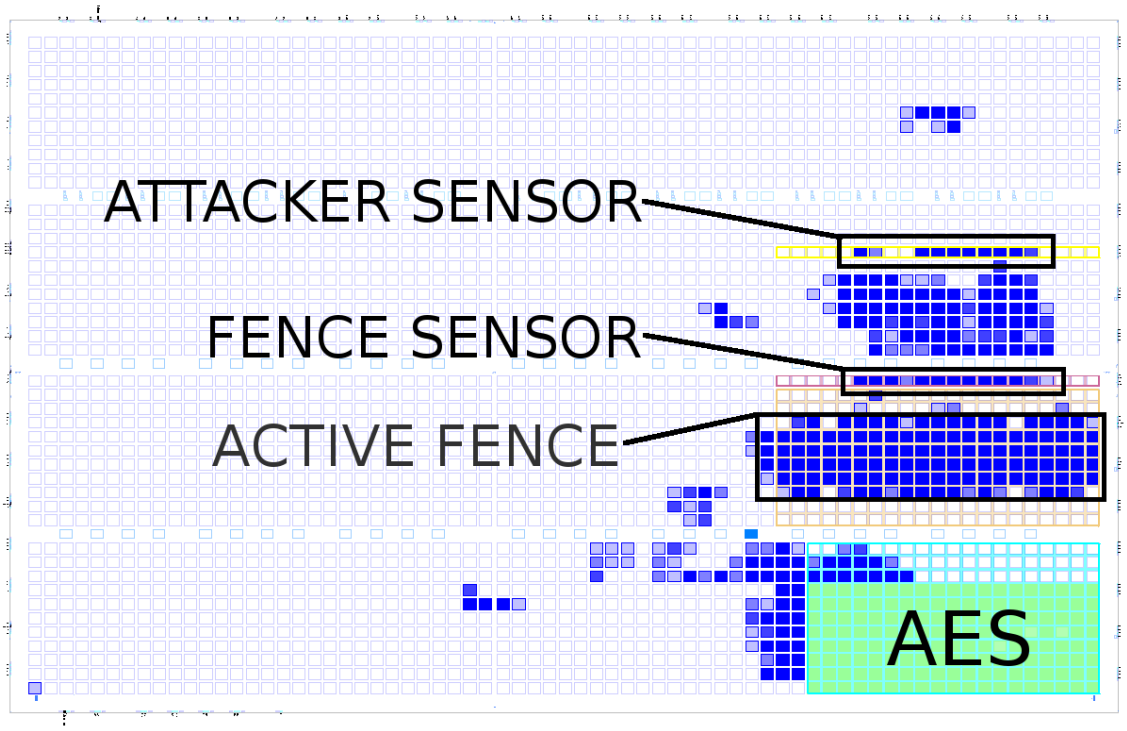


Figure 6.6.: The complete layout of the design on the ECP5 FPGA as seen in the Lattice Diamond Floorplan Viewer

respectively, which fixes the placement of the entire sensor due to the described adjacent placement of *CCU2C* elements. Although the initial delay length varies depending on the sensor clock, we fix the output carry-chain at a length of 62. When the supply voltage is high, the sensor clock signal propagates into the delay line further than at low voltage until the falling edge causes the sensor registers to capture the state of the line. We count the number of set bits in the main delay line as the sensor output, which gives us a 6-bit sensor value.

In Figure 6.8, we show 16 bits of a TDC delay line on the ECP5, as seen in the Floorplan View of the Lattice Diamond design software. For our experiments we sample at 100 MHz on the attacker side sensor, whereas the TDC sensor on the Active Fence part was restricted to 50 MHz to account for a realistic scenario. We assume that in general the attacker is able to sample the side-channel information at a higher rate compared to the clock speed of the attacked module. Thus, the attacked AES module is driven at 12.5 MHz clock speed, giving the attacker 8 samples per AES clock. As shown in the next section, the amount of traces required to attack an unprotected design is rather small.

6.1.4.3. Active Fence Implementation

In Section 6.1.3, we explained the general principle of an active fence as a hiding countermeasure. The implementation on the Lattice ECP5 FPGA is based on single LUT ROs, which are instantiated using the *ND2* primitive. A *ND2* element represents a two-input NAND-gate to realize a RO with an enable signal. To prevent any synthesis

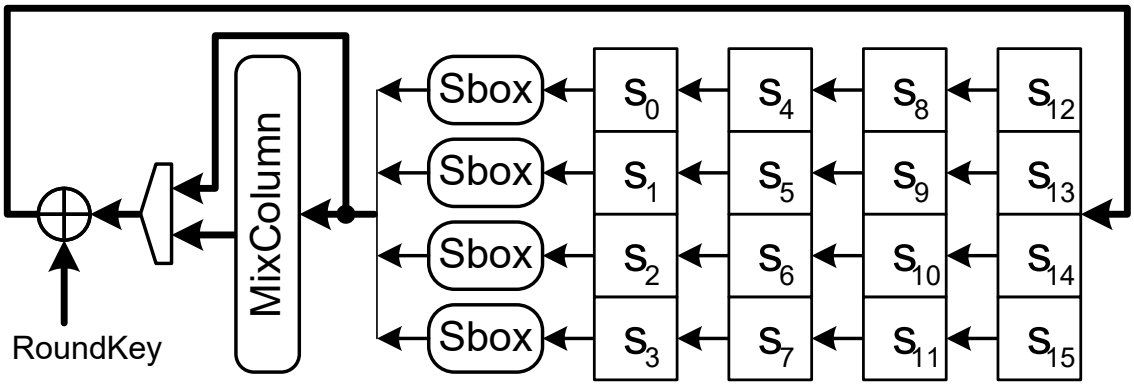


Figure 6.7.: Schematic of the used AES encryption core design as shown in [12]

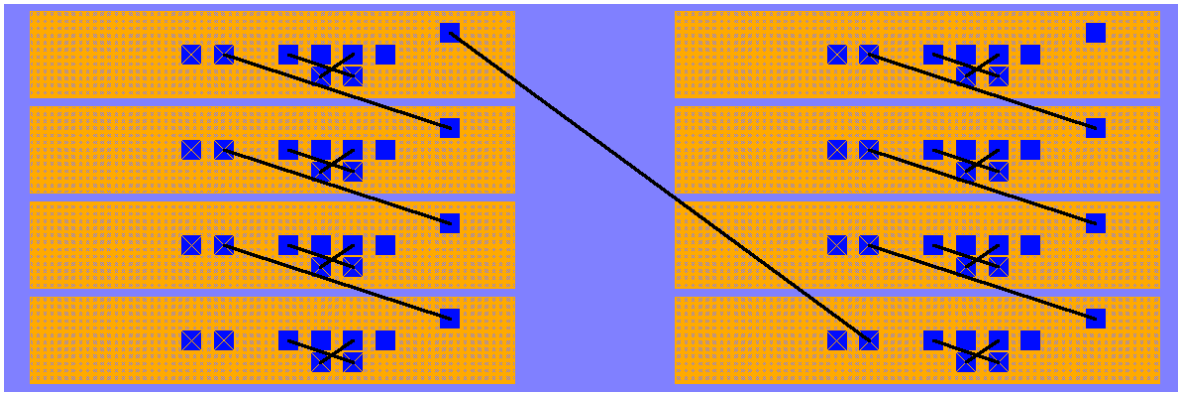


Figure 6.8.: Partial sensor carry-chain on the ECP5 as displayed in the Floorplan View of the Lattice Diamond design software; Each block contains a 2-bit carry element and the connection between those elements enforces adjacent placement as a fast carry chain on the ECP5

optimizations from removing the RO array, we combine all RO signals with an XOR operation into a single signal, which is mapped to an open pin on the FPGA.

The ROs are placed according to one of the placement strategies described in Section 6.1.3 and a certain amount is activated depending on a value x. We use a total amount of $21 \times 32 = 672$ ROs, where for each increase in x an additional amount of 21 ROs is activated. This total amount of ROs is chosen to correspond roughly to the amount of slices occupied by the AES module for the fence to be able to generate voltage fluctuations in the range of the cryptographic implementation itself. Furthermore, a row of 21 adjacent ROs is required to spatially cover the AES module when using a row-by-row RO layout. 32 corresponds to the maximum fluctuation range in the TDC sensor we observed in our experiments. Thus, a value of $x = 20$ activates a total amount of 420 ROs.

Overall, the overhead of the active fence in terms of area is about the same as the area occupied by the AES module. Analyzing the required additional power consumption by the design with active fence as reported by the Lattice Diamond Power Calculator is 178 μW. This corresponds to slightly more than half of the additional power consumption reported for the AES module, which is 320 μW. However, the tool does most likely not account for the dynamic power consumption of the ROs adequately and thus further measurements would be required.

In Section 6.1.3.2 we explain different activation methods. We can choose x either as a PRNG output or a TDC sensor value. For the sensor-based activation, we scale the sensor value into the range of 0 to 32 by subtracting the observed minimum in the first encryptions. To randomly activate the RO grid, we simply use x directly as the output of a PRNG module from OpenCores [115].

6.1.5. Results

Our results are based on performing a CPA on an AES encryption module in different scenarios using the voltage traces collected through the attacker TDC sensor. For each scenario, we deploy variations of our active fence implementation or omit any protection for the baseline experiment. The success of any defensive method is evaluated as the increase in traces required for the attacker to successfully recover the first secret key byte of the last AES round key. For CPA we use a bitwise correlation model as described in Section 2.3. After selecting the best correlating bit, we plot the key hypothesis correlations over the number of evaluated traces with the correct key marked red to show the minimum traces required for a successful attack.

6.1.5.1. Baseline, without mitigation

Initially, we synthesize a design without any countermeasures, to compare the efficiency of our active fence countermeasure against. In Figure 6.9, the result of a CPA on the unprotected AES is shown as described previously. We see that the attacker is able to successfully distinguish the correct key byte hypothesis after about 1 800 traces. Comparing these results to those reported before in [12, 14, 22], we need a similar amount of traces as Schellenberg et al.

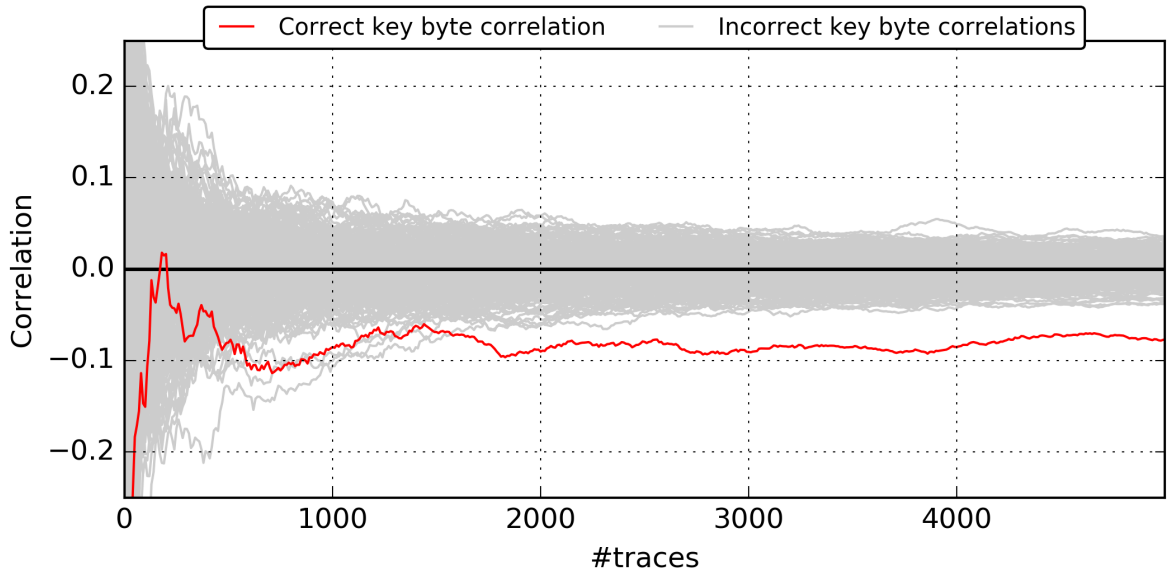


Figure 6.9.: Baseline results for CPA on AES without any countermeasure. The correlation for the correct key is marked red, and the attack is successful after about 1 800 traces.

6.1.5.2. Area-restricted ROs, randomly enabled

For a first experiment, we naively let the design tool place ROs into a constrained area between victim and attacker partitions and randomly activate different amounts of them. The amount of activated ROs is determined by the output of an LFSR-based PRNG. Figure 6.10 shows the result of a CPA on AES in this scenario. We see that the amount of traces required for a successful attack increases from 1800 to 80k. In general, that makes this approach already a valid defense mechanism, but we show in further experiments, that we are able to improve the defense by specific placement and activation.

6.1.5.3. RO-array, randomly enabled

The hiding effect becomes much stronger, when placing ROs in a specific manner, to prevent the attacker from profiting from placement convenience. As described in Section 6.1.3 and Section 6.1.4, we place ROs in a row-by-row scheme between AES module and attacker sensor. Again using a PRNG output to determine the amount of activated ROs, we show in Figure 6.11 how CPA performs in this scenario.

The correlation with the correct key byte is now much lower in absolute value. A successful attack is possible after 120k traces, where the absolute correlation with the correct key is larger than the correlations with the incorrect keys. In our last experiment, we show how the attack success can be decreased more reliably with a sensor-based active fence.

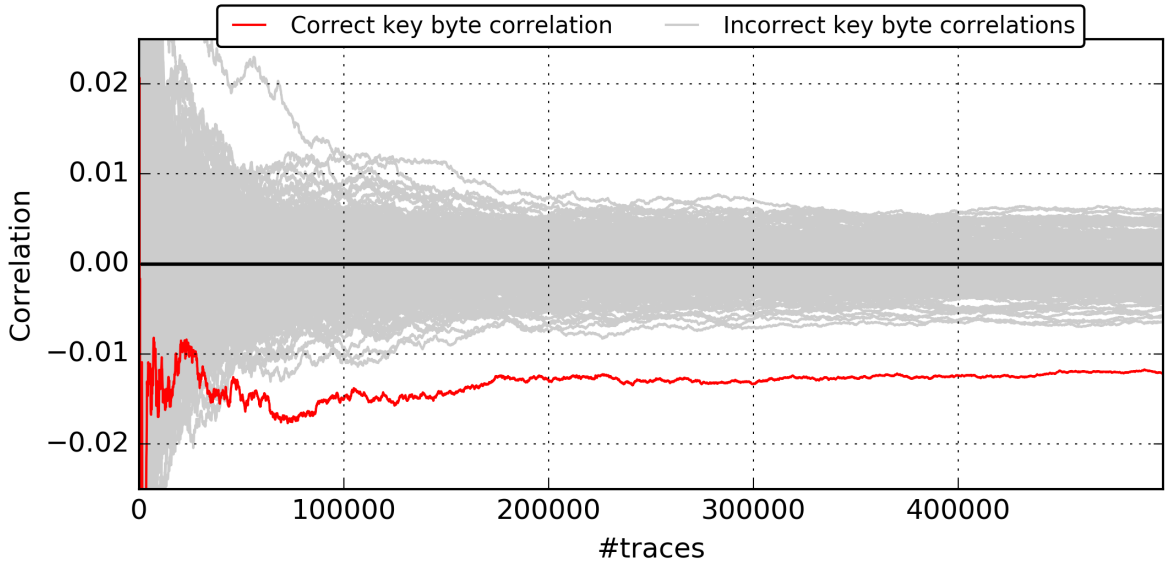


Figure 6.10.: Results for CPA on AES with an arbitrarily placed active fence, activated based on a PRNG output. The correlation for the correct key is marked red, and the attack is successful after about 80k traces.

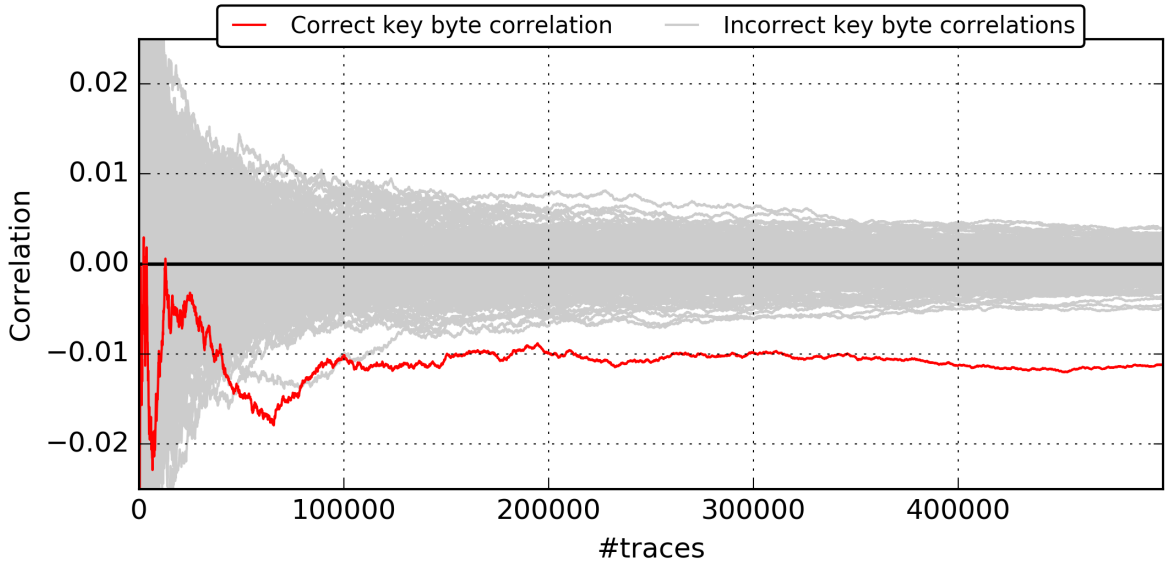


Figure 6.11.: Results for CPA on AES with an active fence, that is placed row-by-row and activated based on a PRNG output. The correlation for the correct key is marked red, and the attack is successful after about 120k traces.

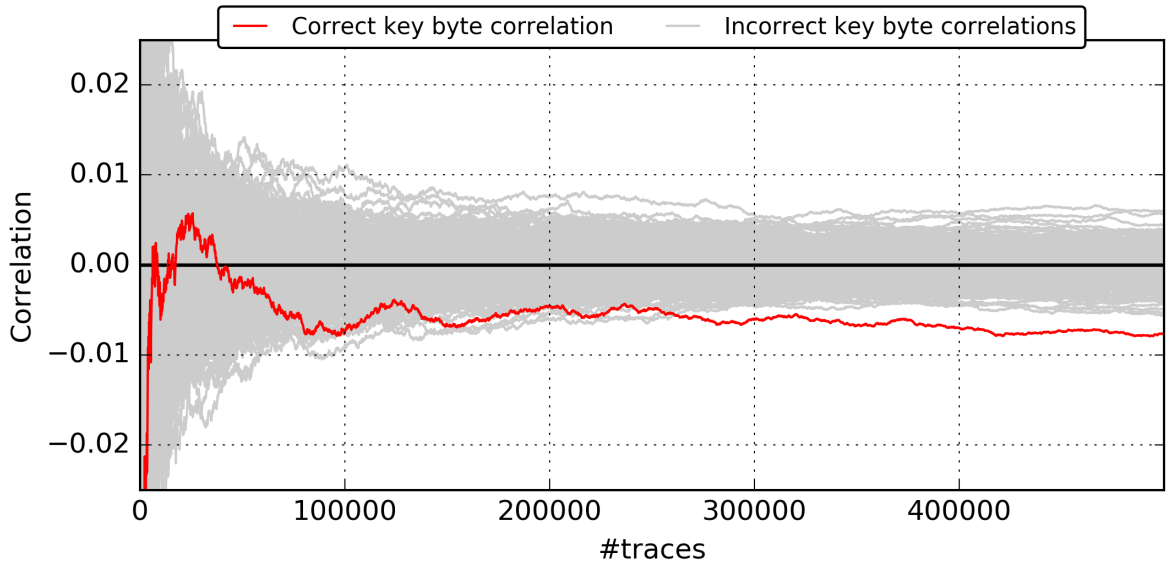


Figure 6.12.: Results for CPA on AES with a sensor-based active fence, that is placed row-by-row. The correlation for the correct key is marked red, and the attack is successful after about 300k traces.

6.1.5.4. RO-array, sensor-based enabled

Lastly, we determine the amount of ROs to be activated using the output value of a TDC sensor, clocked at half the speed of the sensor used by the attacker. The goal of this activation strategy is to directly mitigate the voltage fluctuations caused by the AES encryption. In Figure 6.12, we show the results of a CPA on the AES module in this setup. As in the previous experiment, the absolute correlation value decreases significantly. Additionally, the attack is only successful after about 300k traces, which corresponds to two magnitudes of additional traces being required when compared to attacks on the unprotected module. Further improvements to the sensor feedback will most certainly lead to an even better leakage mitigation in future works.

6.1.6. Discussion

We develop a first approach for an on-chip hiding countermeasure against side-channel attacks on multi-tenant FPGAs. In this scenario, the limitations of the attackers are known, whereas an external attacker can employ any expensive measurement equipment. The evaluation of different placement and activation strategies shows that the general approach of active fences between different designs on a multi-tenant FPGA is feasible.

To significantly improve the hiding effect further investigation of the spatial dependencies and improvements of the defensive sensor are certainly required. In Chapter 4, we investigate the impact of physical design parameters on side-channel vulnerability and find it to be highly relevant for deploying countermeasures as well. Placing the active fence inconveniently might significantly decrease the protection level. However, the

dynamic noise generation through local sensors might actually help in counteracting the effect of module placement.

Currently, we employ a clocked sensor, similar to the attacker sensor to mitigate fluctuations caused by an AES module. A better approach would be an entirely combinational circuit to enable an almost instantaneous reaction of the active fence.

The benefit of the presented method lies mostly within its generality and simplicity as leakage can be prevented independently of the underlying cryptographic module, implementation, and mapping.

6.1.7. Conclusion

We are able to show how a specifically placed and activated active fence made out of ring oscillators can effectively reduce the on-chip side-channel leakage by two orders of magnitude. Additionally, the approach is simple to deploy and independent of the cryptographic module that should be protected. In future research, the specific implementation may be improved, for example through more advanced on-chip sensors which can react to voltage fluctuations instantaneously. Moreover, the connection between placement of the fence elements and success of the countermeasure should be investigated further.

Hiding schemes like what we propose here are known to be unable to prevent side-channel attacks entirely. Instead, they make the attacks harder to mount, i.e., a higher number of measurements is required. Therefore, masked implementations can benefit more from additionally integrated hiding countermeasures.

Hence, examining the effect of our developed on-chip hiding technique on higher-order side-channel attacks when the underlying cryptographic module is a masked implementation is among our future plans. Our method is also orthogonal to the approach of checking bitstreams for malicious signatures before uploading them to a multi-tenant FPGA which has been proposed previously as a countermeasure to security threats in multi-tenant FPGAs.

6.2. Neural Networks as a Side-Channel Countermeasure: Challenges and Opportunities

Just like in the previous section, we explore another hiding countermeasure against side-channel attacks here, which leverages the advantages of heterogeneous computing systems such as FPGAs. FPGAs are currently heavily utilized for accelerating Deep Neural Network (DNN) inference in the cloud [116, 117].

DNNs have become a game changer in many applications from speech recognition and image classification to medical diagnosis and finance. Through a trainable model of a biological neural network, DNNs can be used to approximate almost any known or unknown function, be it for classification or regression tasks. Moreover, many software frameworks [118, 119] and specific hardware accelerators, make the development and deployment of artificial neural networks easy and fast like never before. With the increasing use of DNNs in many fields, they have also gained interest in the field of implementation security, mainly as a way to improve attacks.

We propose the use of DNNs as a computational abstraction, to further drive the development of SCA countermeasures. Whereas hiding [57, 58] and masking [59] countermeasures exist, they are often tailored to specific hardware or algorithms respectively. By training a DNN to perfectly fit a cryptographic function or a sensitive part of it, we can optimize the security of the neural network as an intermediate layer between hardware and algorithm through choice of hyperparameters during both training and inference. This approach is in line with other works on using DNNs as a means to improve resilience against fault attacks of critical algorithms [120].

As a proof of concept, we replace the AES S-Box with different variants of a trained DNN and evaluate its vulnerability against CPA [38]. An initial naive DNN design is still highly vulnerable, so we aim to mitigate the leakage by evaluating different parameters, such as the input and output data encoding, application of dropout during training, or the activation function. Eventually, we find the choice of activation function to have a significant impact on the attack success.

The flexibility of the neural network abstraction allows us to achieve partial side-channel resilience in our setup, which is based on a Xilinx Zynq-7000 FPGA platform. We evaluate the design against both external as well as remote (internal) attackers, the latter making use of the sensors explained in Section 2.6. Our final design is not vulnerable against an attacker with external measurement equipment, even when collecting up to one million traces. When estimating supply voltage fluctuations through internal delay lines, however, our design can still be attacked.

In the next section, we provide some general background on neural networks, as well as an overview on related works. A more detailed motivation and our methodology are described in Section 6.2.2, whereas the actual implementation on our experimental platform is explained in Section 6.2.3. Our results are presented in Section 6.2.4, and we draw some conclusions in Section 6.2.5.

6.2.1. Preliminaries

Before describing our motivation and methodology in more detail, we provide the necessary background information on artificial neural networks as well as an overview on existing work related to the topics in this section.

6.2.1.1. Artificial Neural Networks

Deep learning [121] has improved algorithms in almost every field of our everyday life and has become a widely discussed topic in academia as well. Neural networks have arrived down to consumer electronics and with the widespread application, the interest in specialized hardware accelerators has been rising as well. In this subsection, we describe the general principles of artificial neural networks as far as they are relevant to this work.

Artificial neural networks are composed of neurons, which mathematically model the principle of their biological equivalent: A neuron computes a weighted sum of its inputs, which is then transformed through a nonlinear activation function $f(x)$, modelling the firing of the neuron when the input exceeds a certain threshold. The weight matrix is adapted during training through backpropagation of the output error, when comparing

to a known ground truth in the training set. Usually, multiple neurons together form layers, where – in the fully connected case – each neuron is connected to each input. A DNN is composed of multiple, concatenated neuron layers.

In our implementations, we employ fully connected layers – which are also called dense layers – in a multilayer architecture of varying size. Usually, the training dataset is randomly selected from the respective application domain and then the network trained on the dataset in order to minimize a loss function. In our case of replicating the 8-bit AES substitution function with a neural network, we have only 256 possible input and output values.

6.2.1.2. Neural Networks and Implementation Security

Artificial neural networks have been targeted by SCA as well [122–124] and countermeasures against such attacks have been proposed [124]. The proposed mitigations in [124] make use of both hiding and masking schemes to protect hardware implementations on FPGAs and Application Specific Integrated Circuits (ASICs). Those works are orthogonal to our approach, as they consider classical DNN applications, where the weights are part of the secret data, which must be protected from an attacker. In our approach, a known and well-defined function is replicated as a DNN and the attacker is assumed to know the function, making the protection of weights irrelevant. However, combining generic countermeasures for neural networks with our approach of replicating classical cryptographic primitives might lead to a very generic mitigation against SCA.

Moreover, DNNs have recently been proposed as a generic countermeasure against fault attacks [120]. Interestingly, artificial neural networks can be optimized for improved fault-tolerance during training. Similar to our proposal for SCA mitigation, the authors in [120] replicate the AES S-Box including the preceding XOR operation using a DNN, achieving a higher fault tolerance and less vulnerability to fault attacks. To target fault tolerance, their DNN architecture makes use of a much larger amount of neurons and is trained with specific constraints, namely L2-regularization, on the network weights.

Although many existing hiding and masking countermeasures can mitigate side-channel leakage very efficiently, they usually lack generality as they are targeted at specific implementations, i.e. specific hardware or a specific algorithm. In light of the developments described above and the ever-increasing support for neural network acceleration in hardware and software nowadays, it seems not far-fetched to also leverage this versatile computational model for SCA mitigation.

6.2.2. Optimizing DNNs for SCA Resilience

Here, we explain our motivation for using DNNs as a side-channel countermeasure as well as our general approach and the evaluation of training and implementation parameters to optimize the resilience of a DNN implementation.

6.2.2.1. Motivation

Whereas the protection of neural networks in classical AI applications has been explored, and they have been used on the attacker side for side-channel analysis as well, their use for side-channel mitigation is yet to be explored. We find several points, that motivate the use of DNNs against side-channel leakage:

- Any data-dependency in the control flow is automatically removed: The DNN always performs the same amount of computations.
- A DNN can be trained to exactly fit a known function and offers a generic representation, that is independent of the (cryptographic) algorithm. Unlike, for example, masking countermeasures, which are tailored to a specific algorithm, the DNN allows addressing the sources of leakage on a lower level, independent of the function it implements.
- Data can be processed by the DNN in almost arbitrary numerical representations, which can be beneficial, when mitigating attacks that target, for instance, a binary data representation.
- To represent a specific function, an exponentially large amount of different DNNs with different topology or weights exists to implement this function. Thus, attacks on one instance do not necessarily scale to other instances. Moreover, the ability to generate an arbitrary amount of randomized implementations of the same cryptographic primitive can be combined with other countermeasures which are, for instance, based on implementation diversity [125].
- Neural network implementations have been leveraged to increase the resilience against fault attacks as well [120], which is why the addition of side-channel resilience can potentially secure a design against both implementation attacks in a unified approach.

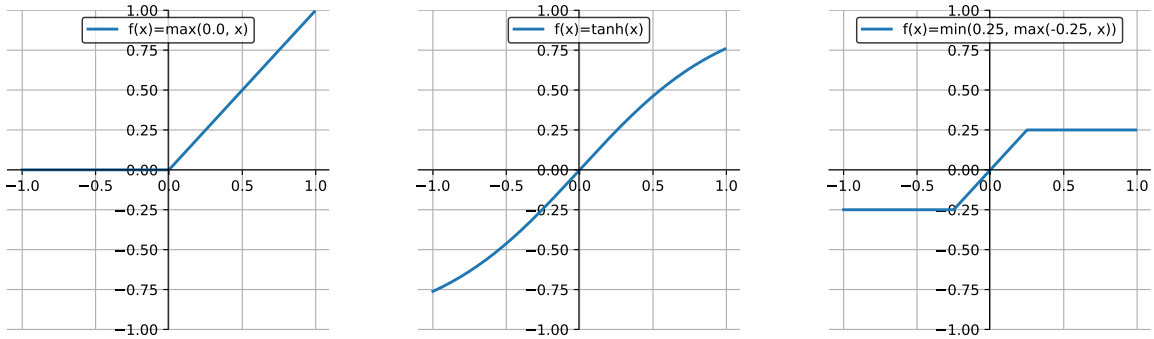
As an intermediate layer between a specific hardware and the cryptographic algorithm, DNNs can abstract both sides, and we prove their flexibility, when optimizing a function towards side-channel resilience.

6.2.2.2. Optimization Approach and Parameters

We investigate a multitude of different parameters w.r.t. how they can improve side-channel resilience of a DNN implementation of a cryptographic function. Here, we describe our general neural network architecture as well as the examined parameters and why we assume them to have an impact on the vulnerability.

General Design:

Our initial design is a fully-connected feed-forward DNN with eight input and output neurons as well as two hidden layers of 32 and 64 neurons each. The network size has been empirically determined to be as small as possible, without incorrectly calculating S-Box values. The weights are initialized randomly using the Glorot initializer [126],



(a) ReLU function (b) *Tanh* function (c) Ramp function

Figure 6.13.: An overview of different activation functions. In our FPGA implementation, we evaluated the ReLU function as well as ramp activation, which is sometimes called hard *tanh*.

which takes the amount of input and output nodes of each neuron into account. We map the 8-bit AES S-Box into input and output vectors of size 8, where each vector element represents a bit of the S-Box input or output and each bit is converted such that -0.5 corresponds to 0 and 0.5 to 1. As a nonlinear activation function, we initially employ the commonly used Rectifier Linear Unit (ReLU) function. Training the described network for about 5 000 epochs with the gradient-based Adam optimizer [127] and the Mean Squared Error (MSE) loss function, we achieve 100% accuracy in replicating the AES S-Box function. During inference, we only compare the sign bit of each output neuron to transform back to a 0 or 1 binary output.

Activation Function:

Although the ReLU activation function works well with replicating the AES S-Box function and is easy to implement, it may not be beneficial for mitigating SCA attacks. This is due to its behaviour of clipping the output values at 0, which may lead to significant differences in the power consumption of subsequent computations, where values are sometimes zeroed depending on the input data. Instead, we suggest the use of *tanh* or the ramp function $f(x) = min(0.25, max(-0.25, x))$, which is also called hard *tanh* sometimes, as both have a smoother behaviour with the output value hardly ever being exactly zero. In Figure 6.13, we present the three different suggested activation functions. Whereas *tanh* is easy to implement in software, the ramp function is more suited for hardware implementations, which is why we choose to implement it in our FPGA-based setup.

Input/Output Encoding:

As many SCA attacks including CPA are based on the binary representation or hamming weight of the data, encoding the data in a different base is another parameter, which we explore for leakage prevention. In any base k, we need $n = \lceil \log_k(256) \rceil$ neurons in the input and output layers, corresponding to the digits of the input and output data in base-k encoding. We evaluate a neural network to compute the S-Box function on base-5 inputs and produce base-5 outputs, which has four input and four

output nodes. The digits in the range $0, 1, ..., 4$ are scaled into a range of -0.5 to 0.5, such that 0 is represented as -0.5 and 4 as 0.5. In order to still fit the S-Box function exactly, the network needs to be slightly larger than our base two-layer network. Instead, we employ four hidden layers of 64 neurons each.

Neural Balancing:

Aiming to balance the power consumption of layer computations for all inputs, we evaluate different training modifications. One option is to introduce regularizers, which penalize unbalanced layer outputs during training. More specifically, we add the standard deviation of the sum of all neuron outputs per layer over all 256 possible inputs as a penalty term to the loss function. Let a_i be the number of neurons with an output > 0 for input i into a layer, then the loss for that layer is $l = \text{MSE}(y_{\text{pred}}, y_{\text{true}}) + \sigma_{\text{act}}$, where the first term is the standard MSE loss function and $\sigma_{\text{act}} = \sqrt{\frac{1}{255} \cdot \sum_{i=0}^{256}(a_i - \bar{a_i})^2}$. This additional loss term causes the optimizer to aim for the same amount of neurons being activated for each possible input in order to potentially equalize power consumption.

Extending the DNN Function:

By incorporating the preceding XOR with the AES round key into the function that is computed by the DNN, we extend the part of the encryption that is covered by neural network computation. Here, the input to the S-Box, which is the XOR of the previous round state byte with the corresponding secret round key, is never present in binary form, but only as weights and activations inside the DNN. Since this value is relevant for the power consumption targeted by the CPA, we expect a higher resilience against the attack. As for the network to perform computations on base-5 encoded data, we require four hidden layers with 64 neurons each. Other than that, we employ the base design here with binary input and output as well as the ReLU activation function.

Binary Neural Network:

We also deploy a network with binary weights $(-1/1)$ using the approach from [128]. For a correctly implemented AES S-Box function, the Binary Neural Network (BNN) has three hidden layers of 512, 1 024 and 2 048 binary neurons. Although this approach uses the input data in binary form and produces a binary output, we expect a large amount of uncorrelated noise, as the computations are based on many XOR operations with binary weights that are unrelated to the actual input value.

6.2.3. Implementation and Setup

In this section, we describe how we train neural networks to replicate the AES S-Box function and we detail in which way they are deployed on our FPGA evaluation platform.

6.2.3.1. Framework and Training

To train DNNs we make use of the widespread Tensorflow framework, which offers a Python interface and stores trained models in the HDF5 format. Moreover, Tensorflow-Lite allows storing networks in a binary format for fast inference on many embedded

architectures. For implementing the DNN in hardware, we make use of quantization, to map the floating-point weights into an integer range and use integer computations instead. The floating-point values are transformed into the range of $[-128, 127]$ and stored as 8-bit signed integers. The neural network computations are then performed on the quantized values, which are transformed back in the input and output layers respectively. We make use of the Tensorflow-internal quantization algorithm and deploy quantized networks replicating the S-Box function on an FPGA.

6.2.3.2. Device and Measurement

Our evaluation platform is the Pynq-Z1 board, which incorporates a Xilinx 7-Series Zynq XC7Z020-1CLG400C FPGA. We implement the networks ourselves, without using the Xilinx IP cores, to maintain maximum control over the experiments and store the network weights in BRAM, using the dedicated DSP blocks of the FPGA for performing the computations. The BNN design does not make use of the DSP blocks, but is instead implemented entirely in standard logic LUTs. As explained in the introduction, we evaluate our design against both an external attacker, who uses an oscilloscope to measure power, and a remote, internal attacker in a multi-tenant FPGA setup, who introduces internal sensors. The internal sensors are based on estimating supply voltage fluctuations indirectly through measuring data propagation delay through long delay lines as explained in Section 2.6.

The external attacks are performed using a Keysight Technologies MSO9104A oscilloscope, where we make use of a Linux system, running on the integrated ARM hard processor system in the Zynq FPGA to control the oscilloscope via USB. The side-channel analysis is performed separately on different machines, using a GPU-based CPA implementation. We remove all decoupling capacitors on the board to improve the quality of our measurements and sample up to $1M$ traces for each attack. The measurements are collected by measuring the supply voltage $V_{DD} \approx 1.0V$ over the FPGA chip.

For evaluating against a malicious tenant on a multi-tenant FPGA, we make use of the self-calibrating delay-line sensors, which have been used in Chapter 4. Due to a much stronger impact of on-chip voltage fluctuations onto the sensor values compared to the external measurements, all implementations can be attacked with a lower amount of traces in this setup. However, as shown in Chapter 4, the sensor placement and mapping can have a significant impact on the attack success, which is why we collect only up to $100K$ traces but with 4 different sensors distributed throughout the FPGA fabric. Again the internal ARM hard processor is used to coordinate the measurements and the evaluation is performed on a separate machine.

6.2.4. Results

Here, we present the results of performing CPA attacks on different implementations of DNNs replacing the AES byte substitution function. First, as a baseline to the following results on DNN implementations, we attack a regular S-Box implementation, which is implemented as combinational logic on the FPGA, without using the BRAM. For all attacks we target the last encryption round and use the respective DNN implementation

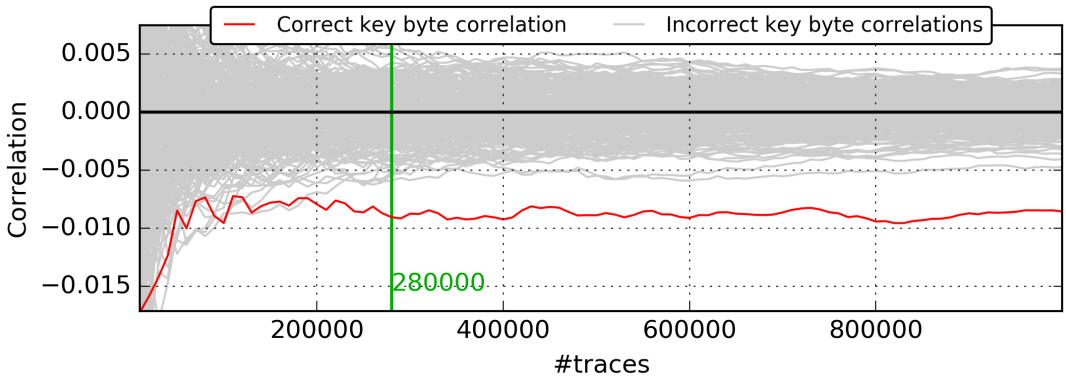


Figure 6.14.: Correlation over the amount of collected traces, when attacking the AES S-Box implemented as standard LUT-based logic through external measurements with an oscilloscope.

for substituting one byte of the AES state. The amount of traces required for key recovery is determined as defined in Section 2.3.

6.2.4.1. Baseline Attack on a Standard Implementation

For comparison, we attack a conventional S-Box implementation on our FPGA evaluation platform. Exemplary we show the result of attacking the design through external measurements, which is presented in Figure 6.14. We are able to distinguish the correct key here after around 280 000 traces. When using on-chip sensors as described in Section 6.2.3.2, only 6 000 traces are already enough to determine the correct key byte value.

6.2.4.2. Attack on the base DNN Architecture

In the previous section, Section 6.2.2.2, we presented the first basic DNN with the ReLU activation function and a binary representation of input and output to replicate the AES S-Box function. The results of attacking the DNN substitution on our FPGA implementation can be seen in Figure 6.15. Here, we exemplary present the final correlation values after $100K$ traces when estimating supply voltage fluctuations through on-chip sensors.

We conclude that the initial DNN implementation is highly vulnerable against CPA attacks, as $\leq 1\,000$ traces are required for a successful key recovery, both when using external measurement equipment and when estimating voltage fluctuations internally. In fact, we are even able to reconstruct the DNN architecture, when looking at the final correlation values over the sampling period, where the layer structure is clearly visible.

6.2.4.3. Evaluation of DNN parameters

In Table 6.1, we present the results of performing CPA attacks on our previously explained DNN design variants. We note that many of the parameter choices fail to

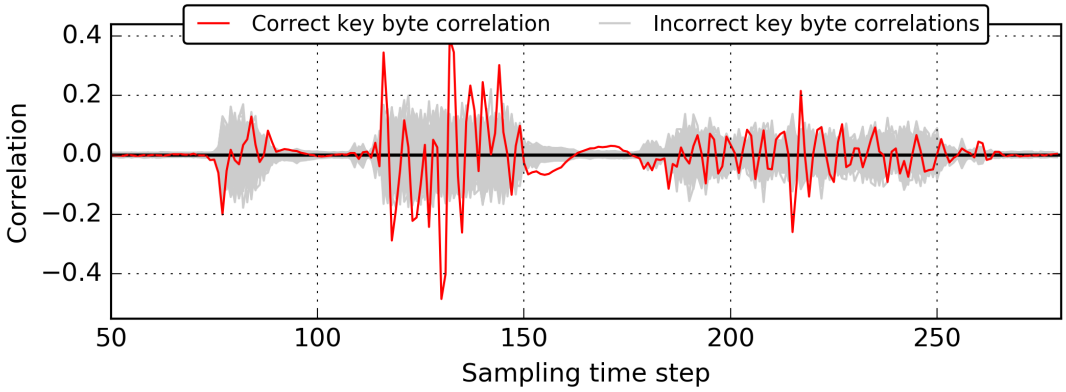


Figure 6.15.: Final correlations at each point in time during the sampling period, when attacking the AES S-Box implemented as a neural network using our base DNN design through measurements with on-chip sensors.

Design variant	Traces for key recovery External measurement	 Internal measurement
Standard implementation	280 000	6 000
Basic DNN design	$\leq$ 1 000	$\leq$ 1 000
Base-5 encoding	$\leq$ 1 000	$\leq$ 1 000
Neuron balancing	$\leq$ 1 000	$\leq$ 1 000
With key XOR	$\leq$ 1 000	$\leq$ 1 000
Binary weights	$\leq$ 1 000	$\leq$ 1 000
Ramp activation	$>$ **1 000 000**	**3 000**

Table 6.1.: An overview of the amount of traces required for a successful CPA attack, reflecting the side-channel vulnerability of each design variant.

suppress the leakage and mitigate the attack, making them just as vulnerable as the base design. In fact, all evaluated approaches are unable to prevent a malicious tenant from recovering the secret key.

However, when applying a ramp function $f(x) = min(0.25, max(-0.25, x))$ instead of the default ReLU activation, we observe a significant decrease in leakage during the layer computations. The hard clipping to 0 output values with the ReLU activation has indeed a significant impact on the leakage of the layer computations. In fact, we are unable to recover the secret key even with up to $1M$ traces when measuring externally with an oscilloscope, which shows in the final correlation values over the sampling period in Figure 6.16. On the other hand, even for an attacker who uses on-chip sensors to estimate fluctuations, the amount of traces for a successful attack is higher when using a ramp activation. Nevertheless, our designs are unable to fully mitigate internal attacks, which could be related to an increased amount of information about voltage fluctuations in a specific region of the FPGA, which is visible to a single on-chip sensor, but invisible to an outside attacker who can only measure the sum of fluctuations.

We confirm the significance of the activation function, by training three different

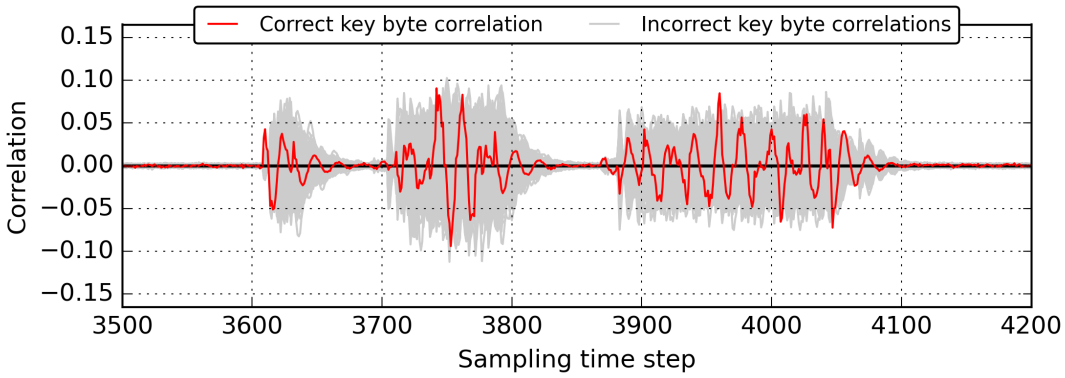


Figure 6.16.: Final correlations at each point in time during the sampling period, when attacking the AES byte substitution, which is implemented as a neural network with a ramp activation function instead of the default ReLU activation.

DNNs with randomly initialized weights and repeating the attack using external measurement equipment. For each design, we are unable to recover the secret key with up to $1M$ measurement traces. This result also demonstrates the ability to generate an arbitrary amount of randomized implementations, which can be combined with other countermeasures [125], as explained in Section 6.2.2.1.

We identify the activation function as a critical parameter for side-channel resilience and – despite not being able to mitigate an internal on-chip attack – exemplarily show how the flexibility and the abstraction level of a DNN implementation can greatly benefit the development of generic side-channel countermeasures.

6.2.4.4. Overhead

We evaluate the overhead of the FPGA implementation in terms of speed and resources for a fair comparison. The entire design including AES and the communication with the ARM processor requires 12% of LUTs, 12% of BRAM and 29% of DSP resources. When using only standard AES S-Boxes and removing the DNN, the design needs 4% of LUTs, no BRAM and no DSPs. Thus, the DNN introduces both area and power overhead, but the design still does not require an unrealistic amount of resources on the rather small Zynq-7000 FPGA and could potentially be deployed to low-power edge hardware as well.

For comparing the encryption time overhead, we run 100k encryptions on the FPGA with all S-Box operations replaced by the DNN implementation. With only standard S-Box implementations, the total duration for 100k encryptions is about $2.2s$ which corresponds to $22\mu s$ per encryption. When using a 25 MHz clock for both AES core and DSPs, the total duration is $118,6s$, corresponding to $1.186ms$ per encryption. Therefore, we have an overhead of about $50\times$ in speed, when using the DNN implementation with the same clock as the standard AES core.

6.2.5. Conclusion

In this contribution, we show how artificial deep neural networks provide an intermediate abstraction layer between countermeasures against side-channel attacks on the algorithmic and the hardware layer. As a proof of concept, we train neural networks to replicate the AES byte-substitution and perform CPA attacks on the implementations, which are deployed on a Xilinx Zynq-7000 FPGA. The designs are evaluated against both an attacker with external measurement equipment, and a malicious tenant using on-chip sensors on a multi-tenant FPGA. We vary multiple parameters during training and inference and learn that the choice of activation function has a significant impact on the side-channel leakage. In the case of an external attacker, a standard implementation of the AES S-Box needs about 280 000 measurements for key recovery. Whereas a naive DNN mapping with a ReLU activation needs less than 1 000 traces, the modified design is protected against an external CPA attack with up to $1M$ measurement traces. However, attackers performing internal measurements with delay-line-based on-chip sensors are still able to recover the secret key, most likely due to a more localized view on the voltage fluctuations in specific regions of the FPGA. Nevertheless, DNN implementations prove to be a versatile and generic instrument to address side-channel leakage in a matter agnostic to hardware and overlying algorithm. Future work will continue to investigate parameters and training approaches, which affect the vulnerability of implementations. Moreover, application specific hardware accelerators and different cryptographic algorithms will decrease and justify the overhead of DNN-based countermeasures.

Part III.

Related Work and Summary

7. Related Work and Future Perspectives

The topic of remote fault and side-channel attacks has been explored before, mostly in the form of cache timing side-channel attacks [129, 130], where memory accesses based on secret values are timed by an attacker to determine whether the accessed data was cached or not. On the other hand, the well-known Rowhammer attack on Dynamic Random Access Memory (DRAM), which is based on crosstalk causing bitflips when nearby rows are continuously accessed, has been successfully deployed remotely as well [131, 132]. However, the first remote *power-based* side-channel and fault attacks have been found in 2017 and 2018 [10, 12–14] only, and the topic has received much attention since then. Many have presented similar attacks on FPGA-based systems [15, 42, 43, 91, 133, 134], but also on microcontrollers [17, 18] and even modern x86-CPUs [32]. Likewise, the concept of remote side-channel attacks based on electromagnetic (EM) has been explored in recent works as well [135, 136].

In this chapter, we present other works that highlight the importance of considering this threat model in today's tightly integrated and miniaturized systems. Furthermore, we discuss potential future research directions both for additional exploration of attack vectors and countermeasures against them.

7.1. Timing-based Remote Side-Channel Attacks

As mentioned, remote side-channel attacks that are not based on (indirect) measurement of the device's power consumption or supply voltage have predominantly been using timing as a means to leak secret information. Modern CPU architectures with their cache hierarchy and their highly optimized utilization of shared resources across security boundaries are often designed with optimization instead of security in mind. For end-user applications, this may not be problematic, but especially cloud computing providers are threatened by high financial loss, when an attacker can escape a virtualized container and eavesdrop on other clients or compromise integrity.

Initially, attacks were mostly focussed on leveraging the cache as a shared resource between processes from different users and also between different CPUs [129, 130]. The basic principle of cache-timing attacks is to put the cache to a state that is known to the attacker, then run the victim code and finally perform timing measurements to determine in which way the state of the cache was changed by the victim. An attacker can, for instance, fill the entire cache with data and trigger the victim's execution. Afterwards, the cache line that was evicted, i.e. which addresses were accessed during the execution, can be determined by timing the accesses to the previously loaded attacker data. If the victim's code contains memory accesses, where secret data is part of the

memory address (for example, when using a lookup table), the leaked information will enable the deduction of said secret data.

The cache is only one point of attack, where secret information is leaked through a timing side-channel. Generalizing the concept of timing the access to a resource which is shared across security boundaries has lead to the notion of *microarchitectural side-channels* [137]. For instance, the penalty for a misprediction by a branch predictor, can leak information from implementations with a data-dependent control flow [138].

A somewhat orthogonal line of research on attacks that rely on the same timing side-channels (or covert-channels) for data leakage is that of Microarchitectural Data Sampling (MDS), where memory contents can be leaked from a speculative context. The recently discovered Spectre [54] and Meltdown [55] have received significant attention by media and industry and mitigation approaches have considerable impact on the system performance [139]. MDS is based on side effects, such as data being loaded to the cache, during speculative execution. If the speculative execution happens due to a branch misprediction, for instance, after an array bounds check, the side effects can be used as a covert-channel to access data across security boundaries.

To mitigate these timing-based remote side-channel attacks, countermeasures have been directed mostly at the timing side-channel itself. Implementations can avoid memory accesses and control-flow dependencies on secret data, direct timing measurements can be prohibited, or the data-dependent timing can be avoided at the cost of performance [137]. However, direct and indirect measurement of effects on the electrical level, such as the attacks presented in this thesis, bypass the majority of these countermeasures.

7.2. Remote Fault and Side-Channel Attacks on the Electrical Level

In the previous section, we exemplarily listed some of the most prominent remote side-channel attacks, which all have in common that they are based on measuring execution timing. Countermeasures can thus be specifically targeted at the timing side-channel, through software- or hardware-side mitigations [137]. Moreover, whereas on x86-based systems cycle counters or flush operations can be easily exploited, timing attacks on other architectures may be significantly more challenging [56]. Here, we present some attack schemes that fit the line of work presented also in this thesis in terms of leveraging measurements on the electrical level to estimate voltage fluctuations or power consumption. As the majority of schemes has been demonstrated on FPGAs, we will distinguish between FPGA-based attacks and attacks that exploit other (unintended) internal sensors or fault injection mechanisms.

7.2.1. Attacks in FPGA-based Systems

Since the initial works on FPGA-based side-channel [12, 14] and fault attacks [10, 13], many subsequent works explore the attack space, finding more ways to exploit the design freedom given by FPGA resources. Extending the existing attacks is essential

to understanding the impact of the threat and developing adequate countermeasures, which is therefore a major part of this thesis and presented in Chapter 3.

Regarding FPGA-based fault attacks, research has been mostly directed towards stealthier attacker designs, that can avoid detection by the hypervisor. In a first follow-up work, it was suggested to replace the initially used Ring Oscillators (ROs) with oscillators that are hidden through a transparent latch or a self-clocked flip-flop [42]. This approach would already bypass the simple design rule checks currently employed by cloud providers [91], but could still be detected by more elaborate bitstream/design checking.

Excessive heat and voltage-drops based on an entirely different mechanism were also demonstrated in [15], where researchers were able to cause transient short-circuits using true dual-port block RAM (BRAM) modules on Xilinx FPGAs. The short circuits occur, whenever data is written with identical addresses but complementary values through the two ports of a BRAM instance, leading to timing faults and even bit-flips in configuration memory.

In [133], an XOR tree was used in a new kind of *glitch amplification* for power-hammering. The basic principle of glitch amplification is to delay the inputs to the XOR tree in a way that maximizes the internal switching inside the structure. The increased power consumption was used to successfully crash Xilinx Ultrascale FPGAs.

Finally, in [43], researchers abused seemingly benign IP cores, such as shift registers and AES modules to cause a Denial-of-Service (DoS). This approach is also extended in this thesis, where we demonstrate the use of AES modules and benchmark circuits for precise fault injection attacks on a recent Intel Stratix 10 FPGA, as explained in Section 3.1.

Concerning FPGA-based side-channel attacks, we evaluate the extent of the leakage through the power supply hierarchy in Section 3.2 and present a thorough analysis of physical design impact in Chapter 4. Here, we present some related work, where the initial attacks from [12, 14] were extended to achieve a better understanding of their nature. Worth mentioning is also leakage occurring through crosstalk between adjacent wires inside FPGAs, which was shown in [89]. However, this leakage can be easily prevented by following industry standards for design placement, such as the Xilinx Isolation Design Flow [81].

In [114] it was first examined, whether side-channel leakage occurs between two FPGAs on the same PCB. The results showed that a CPA attack on the AES is possible when the encryption module is placed on one chip, whereas the sensor is located on the other. In fact, even with a standard configuration of the SAKURA-G board that was used in the experiments (i.e. without removal of capacitors), the attack was successful, albeit at the cost of requiring a higher amount of measurement traces.

A covert-channel between different dies on a high-end multi-die FPGA has been demonstrated in [76]. Such FPGAs are composed of multiple dies, so-called Super Logic Regions (SLRs), which are integrated in a 2.5D manner. The convert-channel showed reliable information transfer ($\approx$ 97% accuracy) between SLRs both on local devices and devices in the Amazon and Huawei cloud.

Lastly, the evaluated scenario in [19] was similar to the one we present in this work in Section 3.2. Covert-channels were successfully demonstrated between CPUs, GPUs,

and a PCIe FPGA accelerator in a standard PC setup, where the leakage occurs only through the system-level ATX PSU.

In addition to the work presented in this thesis, the above research proves the importance of securing FPGA-based systems against fault and side-channel attacks on the electrical level. Side-channel and fault attacks are elevated from being local attacks on one specific device, performed by an attacker with extensive measurement equipment, to attacks that can be deployed remotely and simultaneously on millions of devices. In the next section, we present further attacks that do not rely on FPGA-based sensors or power wasting logic, but instead exploit other security flaws in today's highly complex integrated systems.

7.2.2. Attacks through non-FPGA Mechanisms

Whereas all works presented in the previous section and in this thesis in general exploit the design freedom of FPGAs to deploy side-channel and fault attacks, some recent works have also explored the possibility of using different means of measurement, such as actual internal sensors or indirect measurement through analog components. In this section, we present some of the work that fits this threat model of remote fault and side-channel attacks on the electrical level on systems without an FPGA.

In [17, 18], researchers showed that noise from ADC pins in a mixed-signal SoC actually contains leakage information that can be used to successfully perform a CPA attack on a concurrent AES encryption. Although requiring a significant amount (in the range of 10^6) of measurement traces, the attacks are successful even when the pin is connected to GND or VDD as often recommended for unused pins. Leakage assessment in [17] also show the potential to perform similar attacks on modular exponentiation, which is the main operation in RSA encryption.

The exposure of any kind of power sensor data to low privilege processes can have a severe impact on the system security as demonstrated in [32]. Through power averages at very low sampling rates from the Intel Rolling Average Power Limit (RAPL) interface, CPA attacks against the hardware AES (AES-NI) in modern Intel CPUs have been deployed successfully. Additionally, the work presents further attack vectors such as a derandomization of the kernel address space, which is usually employed as a countermeasure against attacks based on memory corruption.

A different line of research exposed the vulnerability of mixed-signal SoC against remote side-channel attacks, which are exploiting leakage that is modulated as noise onto radio signals emitted from the device while performing encryption [135, 140]. In [135], successful CPA attacks against the AES were demonstrated through measurements of a Bluetooth signal, emitted from an SoC, which performed AES encryptions on an integrated ARM Cortex-M4 microcontroller. The leakage in these attacks happens due to the proximity of the two components of the chip, where one is performing security critical operations that procure electromagnetic emissions, and the other one is transmitting radio signals, which in turn are mixed with the EM leakage from the first component. Recently, researchers also demonstrated the feasibility of EM-based side-channel attacks through integrated sensor components, such as microphones in standard end-user PCs. In [136], ECDSA keys of a remote party could be unveiled from the audio feed of a voice call.

On the other hand, remote fault attacks go back to the famous Rowhammer attacks in 2014 [131], which are still a major unresolved issue in modern DDR4 memory implementations [141]. Through continuous accesses to neighboring memory rows, bitflips can be caused in victim row of the memory system, which can even be possible from a JavaScript sandbox when the victim visits a malicious website [132]. The cause of the bitflips is attributed to voltage fluctuations and electromagnetic interaction in the highly integrated memories through the hammering causing retention failures in neighboring rows.

Many recent works have explored the possibility of using Dynamic Voltage and Frequency Scaling (DVFS) parameters for fault injection [33–36]. In [33], fault injection has been successfully used to recover secret AES keys stored in an ARM TrustZone enclave, by increasing the operating frequency of the device at a specific time to provoke timing faults. Similarly, TrustZone can be broken in multicore scenarios through decreasing the core voltage and simultaneously decreasing the frequency of the attacker core to prevent faults in the attacker execution [35]. On x86 systems, attackers are able to attack Intel SGX enclaves by exploiting hidden Machine State Registers (MSRs) to manipulate the core voltage [34, 36].

Through careful consideration of user privileges and access to interfaces that can be used to manipulate operating conditions or estimate power consumption, many of the above attacks can and have been mitigated, for instance, through microcode updates. However, attacks that rely on unintended effects on the electrical level, such as Rowhammer [131], ADC-based side-channel measurements [17, 18] or EM measurements through analog system components [135, 136, 140] are significantly harder to prevent, as the unsuccessful attempts at counteracting Rowhammer attacks prove [141].

7.3. Countermeasures

In this section, we present countermeasures that have been proposed aside from the ones presented in this thesis in Chapter 5 and Chapter 6. As side-channel and fault attacks have been known for more than 20 years [8, 9], many approaches have been discussed to address the issue from a perspective of the respective hardware or software implementation. For instance, faults can often be addressed by redundancy in similar ways as addressing faults for safety concerns [142]. Likewise, side-channel leakage can be reduced by *hiding* [57, 71] or *masking* [59] countermeasures, as also explained in Section 4.1. However, we present some countermeasures here, which explicitly target the threat model of multi-tenant FPGAs in the cloud and FPGA-based systems in general.

The proposed approaches can be generally classified as offline countermeasures, which attempt to detect malicious logic before it is loaded to the FPGA, and online countermeasures, which detect and neutralize attacks through sensors and other additional logic when the attacker design is already active on the chip. Our proposed offline bitstream checking countermeasure, which we present in Chapter 5, was refined and extended to actual cloud FPGAs in [24]. The authors greatly increase the amount of malicious signatures for detecting novel self-oscillating circuits and glitch amplification designs that would be most likely used for creating excessive voltage drops.

Online countermeasures have mostly been developed with the detection and mitigation of fault attacks in mind [143–146]. In both [143] and [144], methods for locating a fault attacker's design through a sensor network are proposed. When it comes to mitigation of the attacks, [144] and [145] suggest clock-gating and quick disabling of interconnect respectively, to stop the excessive power consumption from impeding other tenants on the same chip. The latter has the potential to even stop attacks that rely on internal clocking, for instance, through self-oscillating clocks in the attacker FPGA region. Alternatively, the operating frequency of the victim design can be automatically lowered, when a critical voltage undershoot is detected, as proposed in [146]. This dynamic frequency scaling, however, is intrusive to the non-malicious tenants on the FPGA.

Similar to how we propose the use of physical design parameters (cf. Chapter 4) and neural networks (cf. Section 6.2) as a hiding countermeasure in this thesis, other works have considered to leverage the advantages of reconfigurable hardware for side-channel countermeasures [72, 147, 148]. In [72], a countermeasure based on implementation variety is proposed, where different implementations of the AES S-Box are randomly interchanged at runtime, increasing the difficulty for a side-channel attacker due to the randomized power profile. A generalization of programmable ROs as a countermeasure against both side-channel and fault attacks is presented in [147]. On one hand, ROs can be employed as sensors for detecting fault attacks, on the other hand, they serve as random noise generators to hide data-dependent leakage against side-channel attacks. Lastly, randomization of the clock driving the victim circuit using a delay line is proposed in [148], which can induce noise at very low overhead.

The presented mitigation strategies in this thesis and in other works can prevent many of the known attacks. However, as the underlying problem is the shared PDN across security boundaries, future development will most likely lead to more advanced attacks, requiring constant adaption to the continuously changing threat.

7.4. Perspectives

The threats to today's highly complex heterogeneous computing platforms presented in this thesis and in the related work poses an important challenge to industry and academia alike. Future research will need to continuously develop new countermeasures against remote fault and side-channel attacks and chip engineers will have to keep the presented threat model in mind when designing security critical hardware.

Most current work is focused on attacking cryptographic hardware to recover secret encryption keys. However, the new hardware component are often also intended to accelerate novel applications, such as artificial intelligence. Thus, we already observe the emergence of attacks, which threaten the integrity and confidentiality of neural networks, specifically on FPGAs [122, 123, 134] and also through remote attacks [149]. Future countermeasures will need to tackle the novel threats but can also leverage the distinctive features of the attacked application, like we presented in Section 6.2 for the case of a neural network implementation.

Considering very recently demonstrated attacks as in [136], where electromagnetic side-channel leakage into internal microphones was exploited through a voice call, we

can safely assume that the extent of the issue is still not fully explored yet. This goes for side-channel measurements, but also for fault injection from software. In [19], GPU-based covert-channels were demonstrated, which suggests that – with a specific instruction sequence – the power consumption of highly parallelized hardware accelerators might be raised to a level where crashes, damage, or fault injection may be possible.

Whereas the work in this thesis and the presented related work is based on attacker circuits, which were developed through hardware description languages directly, many future applications will most likely rely on highly dynamic and automatized translation schemes. High-Level Synthesis (HLS) [150] will compile FPGA designs from C-code directly, or warp processors [151] might dynamically deploy sections of executables with hardware accelerators on an FPGA. In such scenarios, fault and side-channel attacks may be possible by attackers carefully choosing the right sequence of software code.

Summarizing future research directions, both the exploration of new attack vectors and the development of adequate countermeasures will be required, to ensure the security of upcoming heterogeneous systems. In Table 7.1, we provide an overview of the attacks and countermeasures in FPGA-based systems presented in this thesis and in the related works.

Attacks	*Bitstream checking (Chapter 5)*	*Active fences (Section 6.1)*	*NNcrypt (Section 6.2)*	*FPGAdefender ([24])*	*SPREAD ([72])*	*Monitoring ([144])*	*Frequency scaling ([146])*	*LoopBreaker ([145])*	*UCloD ([148])*	*PRO ([147])*
	Countermeasures									
Inside Job ([12])	✓	✓	✓	–	✓	–	–	–	✓	✓
Inter-chip side-channels ([114])	✓	✓	✓	–	✓	–	–	–	✓	✓
Cross-SLR covert-channels ([76])	✓	✓	✓*	–	✓*	–	–	–	✓*	✓
C^{3}APSULe ([19])	✓	–	–	–	–	–	–	–	–	–
Code classification (Section 3.2)	✓	–	–	–	–	–	–	–	–	–
SCA on Neural Networks ([149])	✓	✓	✓	–	✗	–	–	–	✓	✓
FPGAhammer ([13])	✓	–	–	✓	–	✓	✓	✓	–	✓
Sequential oscillator faults ([42])	✓	–	–	✓	–	✓	✓	✓	–	✓
BRAM collision faults ([15])	✗	–	–	✗	–	✓	✓	✓	–	✓
Glitch amplification faults ([133])	✗	–	–	✓	–	✓	✓	✓	–	✓
Benign logic faults ([43],Section 3.1)	✗	–	–	✗	–	✓	✓	✓	–	✓

Table 7.1.: A comprehensive overview of novel attacks on the electrical level as well as countermeasures, specifically in FPGA-based systems. If an attack can in principle be mitigated by a specific countermeasure, we mark it as '✓' in the respective column. In [76], a covert-channel has been demonstrated, which is why we denote countermeasures as '✓*', if the countermeasure would be able to mitigate a side-channel attack in the same scenario. If an attack definitely evades mitigation, we mark it as '✗'. A '—' signifies that the countermeasure is not meant to target the respective attack category.

8. Conclusion

In this thesis, recently discovered vulnerabilities through remote side-channel and fault attacks on the electrical level of heterogeneous FPGA-based computing platforms are thoroughly analyzed in various aspects, regarding their severity, extent and potential mitigation. We find that fault attacks on FPGAs can be performed through seemingly benign logic and show how instruction sequences on a normal x86 CPU can be detected from an FPGA accelerator within the same system, where leakage occurs only through a standard system-level power supply. A systematic analysis of side-channel vulnerability with respect to the physical design parameters, such as placement and routing of attacker and victim design, unveils a high impact of said parameters on the amount of traces required for a successful key recovery attack. The findings highlight the importance of considering design parameters when deploying actual countermeasures and may enable future countermeasures based on placement and routing at zero overhead.

On the other hand, we present a promising approach for offline analysis of user bitstreams to detect potentially malicious attacker designs based on specific known signatures. Evaluating the approach on a large amount of benchmark and attacker designs, we show its effectiveness in preventing attacks without being too restrictive on the benign user designs. Finally, we propose novel hiding countermeasures against side-channel attacks that leverage the flexibility of programmable hardware. We introduce the concept of active fences, where ring oscillators around an encryption module are dynamically activated based on on-chip sensor readings. Moreover, we evaluate the usage of neural networks as a side-channel countermeasures by mapping the byte-substitution of AES into a neural network implementation and adapting architecture and training parameters.

We conclude that the emerging hardware introduces both new threats, the extent of which is not fully explored yet, and entirely new opportunities for countermeasures. Remote power-based fault and side-channel attacks remain an important topic for future research and must be considered for secure hardware design and system integration.

Part IV.

Appendix

Bibliography

[1] F.-Q. Xie, L. Nittler, C. Obermair, and T. Schimmel, "Gate-Controlled Atomic Quantum Switch," *Phys. Rev. Lett.*, vol. 93, p. 128303, Sep 2004.

[2] S. Jiang, D. He, C. Yang, C. Xu, G. Luo, Y. Chen, Y. Liu, and J. Jiang, "Accelerating Mobile Applications at the Network Edge with Software-Programmable FPGAs," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 55–62.

[3] Intel's Edge-Centric FPGA Solutions. [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/edge-centric-fpga-solutions.html

[4] Amazon EC2 F1 Instances. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1

[5] Deep Dive into Alibaba Cloud F3 FPGA as a Service Instances. [Online]. Available: https://www.alibabacloud.com/blog/deep-dive-into-alibaba-cloud-f3-fpga-as-a-service-instances_594057

[6] Deploy ML models to field-programmable gate arrays (FPGAs) with Azure Machine Learning. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-fpga-web-service

[7] A. Vaishnav, K. D. Pham, and D. Koch, "A Survey on FPGA Virtualization," in *28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 131–1317.

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

[9] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," in *Advances in Cryptology — EUROCRYPT '97*, 1997, pp. 37–51.

[10] D. R. E. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[11] D. R. E. Gnad, F. Oboril, S. Kiamehr, and M. B. Tahoori, "An Experimental Evaluation and Analysis of Transient Voltage Fluctuations in FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, 2018.

[12] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An Inside Job: Remote Power Analysis Attacks on FPGAs," in *Proceedings of Design, Automation & Test in Europe (DATE)*, 2018.

[13] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, vol. 2018, no. 3, 2018.

[14] M. Zhao and G. E. Suh, "FPGA-Based Remote Power Side-Channel Attacks," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 805–820.

[15] M. M. Alam, S. Tajik, F. Ganji, M. Tehranipoor, and D. Forte, "Ram-jam: Remote temperature and voltage fault attack on FPGAs using memory collisions," in *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2019, pp. 48–55.

[16] D. Mahmoud and M. Stojilović, "Timing violation induced faults in multi-tenant FPGAs," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1745–1750.

[17] D. R. E. Gnad, J. Krautter, and M. B. Tahoori, "Leaky Noise: New Side-Channel Attack Vectors in Mixed-Signal IoT Devices," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 3, pp. 305–339, 2019.

[18] C. O'Flynn and A. Dewar, "On-Device Power Analysis Across Hardware Security Domains.: Stop Hitting Yourself." *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 4, pp. 126–153, Aug. 2019.

[19] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C^{3}APSULe: Cross-FPGA Covert-Channel Attacks through Power Supply Unit Leakage," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1728–1741.

[20] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "Remote and Stealthy Fault Attacks on Virtualized FPGAs," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1632–1637.

[21] J. Krautter, D. Gnad, and M. Tahoori, "CPAmap: On the Complexity of Secure FPGA Virtualization, Multi-Tenancy, and Physical Design," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, pp. 121–146, Jun. 2020.

[22] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "Mitigating Electrical-Level Attacks towards Secure Multi-Tenant FPGAs in the Cloud," vol. 12, no. 3, Aug. 2019.

[23] D. R. E. Gnad, S. Rapp, J. Krautter, and M. B. Tahoori, "Checking for Electrical Level Security Threats in Bitstreams for Multi-tenant FPGAs," in *2018 International Conference on Field-Programmable Technology (FPT)*, 2018, pp. 286–289.

[24] T. M. La, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch, "FPGADefender: Malicious Self-Oscillator Scanning for Xilinx UltraScale + FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 3, Sep. 2020.

[25] J. Krautter, D. R. Gnad, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Active Fences against Voltage-based Side Channels in Multi-Tenant FPGAs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[26] J. Krautter and M. B. Tahoori, "Neural Networks as a Side-Channel Countermeasure: Challenges and Opportunities," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021, pp. 272–277.

[27] D. R. E. Gnad, J. Krautter, M. B. Tahoori, F. Schellenberg, and A. Moradi, "Remote Electrical-level Security Threats to Multi-Tenant FPGAs," *IEEE Design and Test*, vol. 37, no. 2, pp. 111–119, 2020.

[28] S. Meschkov, D. R. E. Gnad, J. Krautter, and M. B. Tahoori, "Is your secure test infrastructure secure enough? : Attacks based on delay test patterns using transient behavior analysis," in *2021 IEEE International Test Conference (ITC)*, 2021, pp. 334–338.

[29] J. Krautter, M. Mayahinia, D. R. E. Gnad, and M. B. Tahoori, in *Asia and South Pacific Design Automation Conference 2022*, 2022.

[30] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[31] D. J. Bernstein, "Cache-timing attacks on AES," 2005.

[32] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "PLATYPUS: Software-based Power Side-Channel Attacks on x86," in *2021 IEEE Symposium on Security and Privacy (SP)*, May 2021.

[33] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1057–1074.

[34] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based Fault Injection Attacks against Intel SGX," in *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[35] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "VoltJockey: Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-Core Frequencies," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 195–209.

[36] Z. Kenjar, T. Frassetto, D. Gens, M. Franz, and A.-R. Sadeghi, "V0LTpwn: Attacking x86 Processor Integrity from Software," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1445–1461.

[37] "Advanced encryption standard (AES)," Tech. Rep., Nov. 2001.

[38] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, M. Joye and J.-J. Quisquater, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–29.

[39] G. Piret and J.-J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad," in *Cryptographic Hardware and Embedded Systems - CHES 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 77–88.

[40] K. Arabi, R. Saleh, and X. Meng, "Power Supply Noise in SoCs: Metrics, Management, and Measurement," *IEEE Design Test of Computers*, vol. 24, no. 3, pp. 236–244, 2007.

[41] L. Zussa, J.-M. Dutertre, J. Clédière, and A. Tria, "Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism," in *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*, 2013, pp. 110–115.

[42] T. Sugawara, K. Sakiyama, S. Nashimoto, D. Suzuki, and T. Nagatsuka, "Oscillator without a combinatorial loop and its threat to FPGA in data centre," *Electronics Letters*, vol. 55, no. 11, pp. 640–642, 2019.

[43] G. Provelengios, D. Holcomb, and R. Tessier, "Power Wasting Circuits for Cloud FPGA Attacks," in *30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 231–235.

[44] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine, "Moats and Drawbridges: An Isolation Primitive for Reconfigurable Hardware Based Systems," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007.

[45] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems, 1989*, May 1989, pp. 1929–1934 vol.3.

[46] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a Side Channel Based Disassembler," in *Transactions on Computational Science X*. Springer Berlin Heidelberg, 2010, pp. 78–99.

[47] J.-J. Quisquater and D. Samyde, "Automatic Code Recognition for Smartcards Using a Kohonen Neural Network." in *CARDIS*, vol. 2, 2002, p. 6.

[48] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "SCANDALee: A Side-ChANnel-based DisAssembLer using Local Electromagnetic Emanations," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. IEEE Conference Publications, 2015.

[49] M. Christ, A. W. Kempa-Liehr, and M. Feindt, "Distributed and parallel time series feature extraction for industrial big data applications," *CoRR*, vol. abs/1610.07717, 2016.

[50] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache Attacks on Mobile Devices," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 549–564.

[51] G. E. A. P. A. Batista, E. J. Keogh, O. M. Tataw, and V. M. A. de Souza, "CID: an efficient complexity-invariant distance for time series," *Data Mining and Knowledge Discovery*, vol. 28, no. 3, pp. 634–669, apr 2013.

[52] D. Page, "Defending against cache-based side-channel attacks," *Information Security Technical Report*, vol. 8, no. 1, pp. 30–44, mar 2003.

[53] I. Corporation. Intel Processors and FPGAs—Better Together. [Online]. Available: https://itpeernetwork.intel.com/intel-processors-fpga-better-together/

[54] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," *ArXiv e-prints*, Jan. 2018.

[55] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *ArXiv e-prints*, Jan. 2018.

[56] M. Green, L. Rodrigues-Lima, A. Zankl, G. Irazoqui, J. Heyszl, and T. Eisenbarth, "AutoLock: Why Cache Attacks on ARM Are Harder Than You Think," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1075–1091.

[57] R. P. McEvoy, C. C. Murphy, W. P. Marnane, and M. Tunstall, "Isolated WDDL: A Hiding Countermeasure for Differential Power Analysis on FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 1, 2009.

[58] T. Güneysu and A. Moradi, "Generic Side-Channel Countermeasures for Reconfigurable Devices," in *Cryptographic Hardware and Embedded Systems – CHES 2011*. Springer Berlin Heidelberg, 2011.

[59] T. D. Cnudde, M. Ender, and A. Moradi, "Hardware Masking, Revisited," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2018.

[60] N. Mentens, B. Gierlichs, and I. Verbauwhede, "Power and Fault Analysis Resistance in Hardware through Dynamic Reconfiguration," in *Cryptographic Hardware and Embedded Systems – CHES 2008*. Springer Berlin Heidelberg, 2008.

[61] L. Lin and W. Burleson, "Analysis and Mitigation of Process Variation Impacts on Power-Attack Tolerance," in *Proceedings of the Design Automation Conference (DAC)*, ser. DAC '09. ACM, 2009.

[62] M. Renauld, F.-X. Standaert, N. Veyrat-Charvillon, D. Kamel, and D. Flandre, "A formal study of power variability issues and side-channel attacks for nanoscale devices," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2011.

[63] Jianwei Dai and Lei Wang, "A Study of Side-Channel Effects in Reliability-Enhancing Techniques," in *International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2009.

[64] Massimo Alioto, Simone Bongiovanni, Milena Djukanovic, Giuseppe Scotti, and Alessandro Trifiletti, "Effectiveness of Leakage Power Analysis Attacks on DPA-Resistant Logic Styles Under Process Variations," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 2, 2014.

[65] X. Wang, W. Yueh, D. B. Roy, S. Narasimhan, Y. Zheng, S. Mukhopadhyay, D. Mukhopadhyay, and S. Bhunia, "Role of power grid in side channel attack and power-grid-aware secure design," in *Proceedings of the 50th Annual Design Automation Conference on - DAC 2013*. ACM Press, 2013.

[66] A. Wild, A. Moradi, and T. Guneysu, "GliFreD: Glitch-Free Duplication Towards Power-Equalized Circuits on FPGAs," *IEEE Transactions on Computers*, vol. 67, no. 3, 2018.

[67] H. Yu, Q. Xu, and P. H. W. Leong, "Fine-grained characterization of process variation in FPGAs," in *International Conference on Field-Programmable Technology (FPT)*. IEEE, 2010.

[68] K. M. Zick and J. P. Hayes, "Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 5, no. 1, 2012.

[69] B. Gojman, S. Nalmela, N. Mehta, N. Howarth, and A. DeHon, "GROK-LAB: Generating Real On-chip Knowledge for Intra-cluster Delays Using Timing Extraction," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '13. ACM, 2013.

[70] P.-C. Liu, H.-C. Chang, and C.-Y. Lee, "A Low Overhead DPA Countermeasure Circuit Based on Ring Oscillators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 7, 2010.

[71] N. Kamoun, L. Bossuet, and A. Ghazel, "Correlated power noise generator as a low cost DPA countermeasures to secure hardware AES cipher," in *2009 3rd International Conference on Signals, Circuits and Systems (SCS)*. IEEE, 2009.

[72] N. Bete, F. Saqib, C. Patel, R. Robucci, and J. Plusquellic, "Side-channel Power Resistance for Encryption Algorithms using Dynamic Partial Reconfiguration (SPREAD)," *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019.

[73] K. M. Zick, M. Srivastav, W. Zhang, and M. French, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '13*, 2013.

[74] Xilinx, "7 Series FPGAs Configurable Logic Block - User Guide (v1.8)," 2016.

[75] J. Wu, "Several Key Issues on Implementing Delay Line Based TDCs Using FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 3, pp. 1543–1548, 2010.

[76] I. Giechaskiel, K. Rasmussen, and J. Szefer, "Reading Between the Dies: Cross-SLR Covert Channels on Multi-Tenant Cloud FPGAs," in *IEEE International Conference on Computer Design (ICCD)*, 2019.

[77] C. Beckhoff, D. Koch, and J. Torresen, "Short-Circuits on FPGAs Caused by Partial Runtime Reconfiguration," in *Field Programmable Logic and Applications (FPL)*. IEEE, 2010, pp. 596–601.

[78] S. Bhunia, M. Abramovici, D. Agrawal, P. Bradley, M. S. Hsiao, J. Plusquellic, and M. Tehranipoor, "Protection Against Hardware Trojan Attacks: Towards a Comprehensive Solution," *IEEE Des. Test*, vol. 30, no. 3, pp. 6–17, Jun. 2013.

[79] M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Des. Test. Comput.*, vol. 27, no. 1, pp. 10–25, Jan. 2010.

[80] M. Fyrbiak, S. Wallat, P. Swierczynski, M. Hoffmann, S. Hoppach, M. Wilhelm, T. Weidlich, R. Tessier, and C. Paar, "HAL- The Missing Piece of the Puzzle for Hardware Reverse Engineering, Trojan Detection and Insertion," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2018.

[81] D. Corbett, "The Xilinx Isolation Design Flow for Fault-Tolerant Systems," 2012.

[82] C. Albrecht, "IWLS 2005 Benchmarks," Tech. Rep., Jun. 2005.

[83] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR project," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays - FPGA '12*. ACM Press, 2012.

[84] J. Luu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, V. Betz, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, and T. Liu, "VTR 7.0," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 7, no. 2, pp. 1–30, jul 2014.

[85] SpinalHDL contributors. (2019) A FPGA friendly 32 bit RISC-V CPU implementation. [Online]. Available: https://github.com/SpinalHDL/VexRiscv

[86] Cobham Gaisler. (2019) LEON3 Processor. [Online]. Available: https://www.gaisler.com/index.php/products/processors/leon3

[87] A. L. Masle and W. Luk, "Detecting power attacks on reconfigurable hardware," in *Field Programmable Logic and Applications (FPL)*. IEEE, 2012, pp. 14–19.

[88] C. Ramesh, S. B. Patil, S. N. Dhanuskodi, G. Provelengios, S. Pillement, D. Holcomb, and R. Tessier, "FPGA side channel attacks without physical access," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2018.

[89] I. Giechaskiel, K. B. Rasmussen, and K. Eguro, "Leaky Wires: Information Leakage and Covert Communication Between FPGA Long Wires," in *Asia Conference on Computer and Communications Security (ASIACCS)*, 2018, pp. 15–27.

[90] V. M. Alvarez, "YARA - The pattern matching swiss knife for malware researchers," virustotal.com, 2018. [Online]. Available: http://virustotal.github.io/yara/

[91] T. La, K. Pham, J. Powell, and D. Koch, "Denial-of-Service on FPGA-based Cloud Infrastructures — Attack and Defense," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 3, p. 441–464, Jul. 2021.

[92] E. Mones, L. Vicsek, and T. Vicsek, "Hierarchy Measure for Complex Networks," *PLoS ONE*, vol. 7, no. 3, 2012.

[93] T. P. Peixoto, "The graph-tool python library," *figshare*, 2014. [Online]. Available: http://figshare.com/articles/graph_tool/1164194

[94] K. A. Hawick and H. A. James, "Enumerating Circuits and Loops in Graphs with Self-Arcs and Multiple-Arcs," in *Proc. 2008 Int. Conf. on Foundations of Computer Science (FCS'08)*. Las Vegas, USA: CSREA, 14-17 July 2008, pp. 14–20.

[95] L. Shang, A. S. Kaviani, and K. Bathala, "Dynamic power consumption in Virtex™-II FPGA family," in *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays - FPGA '02*. ACM Press, 2002.

[96] A. Czutro, M. Sauer, I. Polian, and B. Becker, "Multi-conditional SAT-ATPG for power-droop testing," in *2012 17TH IEEE EUROPEAN TEST SYMPOSIUM (ETS)*. IEEE, may 2012.

[97] Claire Wolf and Mathias Lasser, "Project IceStorm," 2015. [Online]. Available: https://github.com/YosysHQ/icestorm

[98] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An Open Source Software for Exploring and Manipulating Networks," 2009. [Online]. Available: https://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154

[99] C. Wolf, "Yosys Open SYnthesis Suite," 2013. [Online]. Available: https://www.yosyshq.com

[100] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *SOFTWARE - PRACTICE AND EXPERIENCE*, vol. 30, no. 11, pp. 1203–1233, 2000.

[101] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," in *Proceedings of IEEE Int'l Symposium Circuits and Systems (ISCAS 85)*. IEEE Press, Piscataway, N.J., 1985, pp. 677–692.

[102] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *Design Test of Computers, IEEE*, vol. 17, no. 3, pp. 44–53, Jul 2000.

[103] "OpenCores – The reference community for Free and Open Source gateware IP cores," 2018. [Online]. Available: https://opencores.org/

[104] J.-L. Danger, S. Guilley, S. Bhasin, and M. Nassar, "Overview of Dual rail with Precharge logic styles to thwart implementation-level attacks on hardware cryptoprocessors," in *3rd International Conference on Signals, Circuits and Systems (SCS)*. IEEE, 2009.

[105] A. Moradi and A. Wild, "Assessment of hiding the higher-order leakages in hardware," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 453–474.

[106] S. Trimberger and S. McNeil, "Security of FPGAs in data centers," in *IEEE 2nd International Verification and Security Workshop (IVSW)*. IEEE, 2017.

[107] M. McLean and J. Moore, "FPGA-based single chip cryptographic solution," *Military Embedded Systems*, pp. 34–37, 2007.

[108] Y. Ishai, A. Sahai, and D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks," in *Advances in Cryptology - CRYPTO 2003*. Springer, 2003, pp. 463–481.

[109] A. Moradi and F.-X. Standaert, "Moments-correlating DPA," in *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*. ACM, 2016, pp. 5–15.

[110] H. Homulle, S. Visser, B. Patra, and E. Charbon, "Design techniques for a stable operation of cryogenic field-programmable gate arrays," *Review of Scientific Instruments*, vol. 89, no. 1, p. 014703, 2018.

[111] R. Bertran, A. Buyuktosunoglu, P. Bose, T. J. Slegel, G. Salem, S. Carey, R. F. Rizzolo, and T. Strach, "Voltage noise in multi-core processors: Empirical characterization and optimization opportunities," in *47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 368–380.

[112] (2019) Radiona ULX3S. Radiona.org / Zagreb Makerspace and FER – Faculty of Electrical Engineering and Computing – University of Zagreb. [Online]. Available: https://radiona.org/ulx3s/

[113] (2019) SymbiFlow – open source FPGA tooling for rapid innovation. [Online]. Available: https://symbiflow.github.io/

[114] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "Remote inter-chip power analysis side-channel attacks at board-level," in *Proceedings of the International Conference on Computer-Aided Design - ICCAD '18*. ACM Press, 2018.

[115] J. Castillo Villar. (2019) SystemC/Verilog Random Number Generator. OpenCores. [Online]. Available: https://opencores.org/projects/systemc_rng

[116] Xilinx, "White Paper: Accelerating DNNs with Xilinx Alveo Accelerator Cards (WP504)."

[117] Intel, "Solution Brief: Efficient Implementation of Neural Network Systems Built on FPGAs, and Programmed with OpenCL." [Online]. Available: https://www.intel.de/content/dam/www/programmable/us/en/pdfs/literature/solution-sheets/efficient_neural_networks.pdf

[118] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[119] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[120] M. Alam, A. Bag, D. B. Roy, D. Jap, J. Breier, S. Bhasin, and D. Mukhopadhyay, "Enhancing fault tolerance of neural networks for security-critical applications," *arXiv preprint arXiv:1902.04560*, 2019.

[121] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[122] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I Know What You See: Power Side-Channel Attack on Convolutional Neural Network Accelerators," in *ACSAC*, 2018.

[123] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *DAC*, 2018.

[124] A. Dubey, R. Cammarota, and A. Aysu, "MaskedNet: A Pathway for Secure Inference against Power Side-Channel Attacks," *arXiv preprint arXiv:1910.13063*, 2019.

[125] I. Bow, N. Bete, F. Saqib, W. Che, C. Patel, R. Robucci, C. Chan, and J. Plusquellic, "Side-Channel Power Resistance for Encryption Algorithms Using Implementation Diversity," *MDPI Cryptography*, 2020.

[126] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *PMLR*, 2010.

[127] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2014.

[128] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *NIPS*, 2016.

[129] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *Cryptographers' track at the RSA conference*. Springer, 2006, pp. 1–20.

[130] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 719–732.

[131] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *International Symposium on Computer Architecture (ISCA)*, 2014, pp. 361–372.

[132] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript," *CoRR*, 2015.

[133] K. Matas, T. M. La, K. D. Pham, and D. Koch, "Power-hammering through Glitch Amplification – Attacks and Mitigation," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 65–69.

[134] A. Boutros, M. Hall, N. Papernot, and V. Betz, "Neighbors From Hell: Voltage Attacks Against Deep Learning Accelerators on Multi-Tenant FPGAs," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 103–111.

[135] G. Camurati, S. Poeplau, M. Muench, T. Hayes, and A. Francillon, "Screaming Channels: When Electromagnetic Side Channels Meet Radio Transceivers," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 163–177.

[136] D. Genkin, N. Nissan, R. Schuster, and E. Tromer, "Lend Me Your Ear: Passive Remote Physical Side Channels on PCs," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022.

[137] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, vol. 8, no. 1, pp. 1–27, 2018.

[138] O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert, "Predicting Secret Keys Via Branch Prediction," in *Topics in Cryptology – CT-RSA 2007*, M. Abe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 225–242.

[139] A. Prout, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Klein, P. Michaleas, L. Milechin, J. Mullen, A. Rosa, S. Samsi, C. Yee, A. Reuther, and J. Kepner, "Measuring the Impact of Spectre and Meltdown," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, 2018, pp. 1–5.

[140] G. Camurati, A. Francillon, and F.-X. Standaert, "Understanding Screaming Channels: From a Detailed Analysis to Improved Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, p. 358–401, Jun. 2020.

[141] P. Frigo, E. Vannacc, H. Hassan, V. v. der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 747–762.

[142] T. G. Malkin, F.-X. Standaert, and M. Yung, "A Comparative Cost/Security Analysis of Fault Attack Countermeasures," in *Fault Diagnosis and Tolerance in Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 159–172.

[143] S. S. Mirzargar, G. Renault, A. Guerrieri, and M. Stojilović, "Nonintrusive and Adaptive Monitoring for Locating Voltage Attacks in Virtualized FPGAs," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 288–289.

[144] G. Provelengios, D. Holcomb, and R. Tessier, "Mitigating Voltage Attacks in Multi-Tenant FPGAs," *ACM Trans. Reconf. Technol. Syst. (TRETS)*, vol. 14, no. 2, 2021.

[145] H. Nassar, H. AlZughbi, D. Gnad, L. Bauer, M. Tahoori, and J. Henkel, "LoopBreaker: Disabling Interconnects to Mitigate Voltage-Based Attacks in Multi-Tenant FPGAs," in *International Conference on Computer-Aided Design (ICCAD)*, 2021.

[146] Y. Luo and X. Xu, "A Quantitative Defense Framework against Power Attacks on Multi-tenant FPGA," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–4.

[147] Y. Yao, P. Kiaei, R. Singh, S. Tajik, and P. Schaumont, "Programmable RO (PRO): A Multipurpose Countermeasure against Side-channel and Fault Injection Attack," *arXiv preprint arXiv:2106.13784*, 2021.

[148] D. Jayasinghe, A. Ignjatovic, and S. Parameswaran, "UCloD: Small Clock Delays to Mitigate Remote Power Analysis Attacks," *IEEE Access*, vol. 9, pp. 108 411–108 425, 2021.

[149] S. Moini, S. Tian, J. Szefer, D. Holcomb, and R. Tessier, "Remote Power Side-Channel Attacks on BNN Accelerators in FPGAs," *Design, Automation and Test in Europe Conference (DATE)*, 2021.

[150] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[151] R. Lysecky, G. Stitt, and F. Vahid, "Warp Processors," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, no. 3, p. 659–681, jun 2004.

Acronyms

ADC Analog-to-Digital Converter.
AES Advanced Encryption Standard.
AI Artificial Intelligence.
ASIC Application Specific Integrated Circuit.
BNN Binary Neural Network.
CMOS Complementary Metal Oxide Semiconductor.
CPA Correlation Power Analysis.
CPU Central Processing Unit.
DFA Differential Fault Analysis.
DNN Deep Neural Network.
DoS Denial-of-Service.
DRAM Dynamic Random Access Memory.
DRP Dual-Rail Precharge.
DSP Digital Signal Processing.
DVFS Dynamic Voltage and Frequency Scaling.
EM electromagnetic.
FF Flip Flop.
FPGA Field Programmable Gate Array.
GPU Graphics Processing Unit.
IC Integrated Circuit.
IoT Internet of Things.
IP Intellectual Property.
LDA Linear Discriminant Analysis.
LFSR Linear Feedback Shift Register.
LUT Look-Up Table.
MDS Microarchitectural Data Sampling.
MSE Mean Squared Error.
MSR Machine State Register.
PCA Principal Component Analysis.
PCB Printed Circuit Board.
PDN Power Distribution Network.
PL Programmable Logic.
PLL Phase Locked Loop.
PR Partial Reconfiguration.
PRNG Pseudo-Random Number Generator.
PSU Power Suppy Unit.
PV Process Variation.
ReLU Rectifier Linear Unit.
RO Ring Oscillator.

SCA Side-Channel Analysis.
SNR Signal to Noise Ratio.
SoC System-on-Chip.
SPA Simple Power Analysis.
STA Static Timing Analysis.
TDC Time-to-Digital Converter.
VM Virtual Machine.
VRM Voltage Regulator Module.

List of Figures

List of Tables

A. Classification and Detection of Code Patterns through the Power Supply

A.1. Assembly Code Patterns for the ARMv7 architecture

Here, we present the assembly code that has been used in this work to evaluate classification and live detection on the Xilinx Zynq 7000 FPGA-SoC, which is an ARMv7 platform. The trigger code before each pattern is omitted here and can be examined in Listing 3.1 in Section 3.2.3.2.

A.1.1. CPU1 Code Pattern

```
mov r10, #3
mov r11, #5
.rept 2000
smlal r5, r6, r10, r11
ror r10, #3
ror r11, #5
.endr
```

A.1.2. CPU2 Code Pattern

```
mov r10, #3
mov r11, #5
.rept 2000
add r5, r5, r10
sub r6, r5, r11
eor r6, r6, r5
and r6, r6, r11
.endr
```

A.1.3. CPU3 Code Pattern

```
vmov.f32 s0, #3.5
vmov.f32 s1, #1.5
.rept 2000
vmul.f32 s0, s0, s1
vadd.f32 s0, s0, s1
vsqrt.f32 s0, s0
vsub.f32 s0, s0, s1
.endr
```

A.1.4. MEM Code Pattern

In the following code, *mem* corresponds to an array of size $32 \cdot 300$ bytes, where 32 is the cache-line size of both L1 and L2 cache on the ARM core of the SoC. When evaluating cached and uncached memory accesses, we can evict the entire array or access its elements such that all subsequent accesses during measurement are either cached or uncached. In Section 3.2.4.4, we reduce the code to only a single memory access.

```
mov r4, %[mem]
.rept 300
ldr r6, [r4]
add r5, r5, r6
add r4, r4, #32
.endr
```

A.1.5. NOP Code Pattern

```
.rept 3000
mov r4, r4
.endr
```

A.2. Assembly Code Patterns for the x86 architecture

Here, we present the assembly code that has been used in this work to evaluate classification in an x86-64 Intel Xeon desktop/server setup. Again, the trigger code before each pattern is omitted here.

A.2.1. CPU1 Code Pattern

```
.rept 2048
mull %%ebx
rorl $0x3, %%eax
.endr
```

A.2.2. CPU2 Code Pattern

```
.rept 2048
addl %%edx, %%eax
orl  %%edx, %%eax
subl %%eax, %%edx
andl %%edx, %%eax
.endr
```

A.2.3. CPU3 Code Pattern

```
.rept 2048
fld (%[fpu])
fld1
faddp
fsqrt
fstp (%[fpu])
.endr
```

A.2.4. MEM Code Pattern

As in the code pattern for ARMv7, *mem* is an array of size $64 \cdot 1\,024$ bytes, where 64 is the cache-line size on x86 systems.

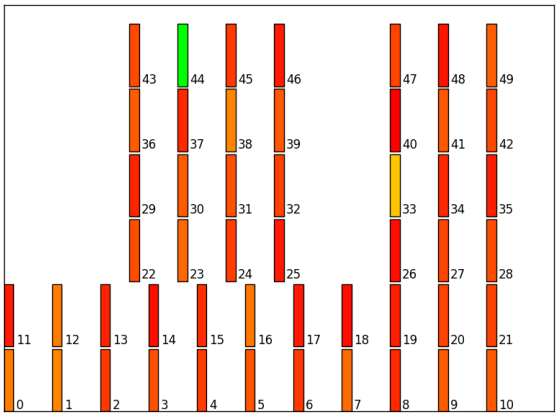
```
.rept 1024
xorb (%[mem]), %%ah
add $64, %[mem]
.endr
```

A.2.5. NOP Code Pattern

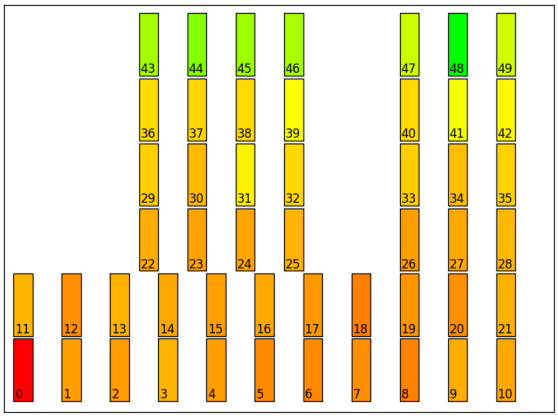
```
.rept 3000
nop
.endr
```

B. CPAmap: On the Complexity of Secure FPGA Virtualization, Multi-Tenancy, and Physical Design

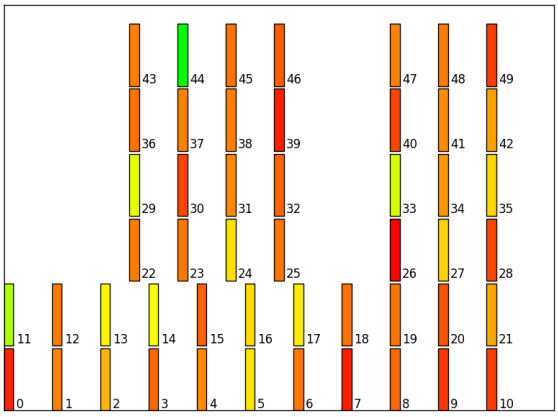
B.1. Influence of Global Placement on Board B



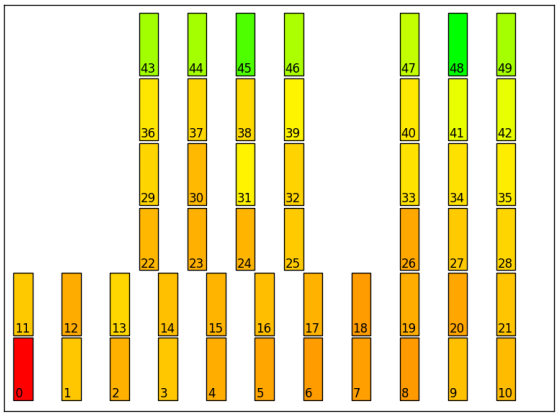
(a) Sensitivity based on sensor average $S^{\mu}(i)$ in different sensor locations i.



(b) Impact based on sensor average $I^{\mu}(i)$ caused by noise generation grids in different locations i.



(c) Sensitivity based on sensor variance $S^{\sigma}(i)$ in different sensor locations i.



(d) Impact based on sensor variance $I^{\sigma}(i)$ caused by noise generation grids in different locations i.

Figure B.1.: Influence of global placement of FF-based noise generators and TDC sensors evaluated and averaged over 1 000 measurements on board B. Locations that are colored orange/red correspond to sensors that are more sensitive or noise generators that cause a higher impact respectively. Green colored locations are less sensitive or cause less impact.

B.2. Evaluating Attacks on the second AES Key Byte on Board A

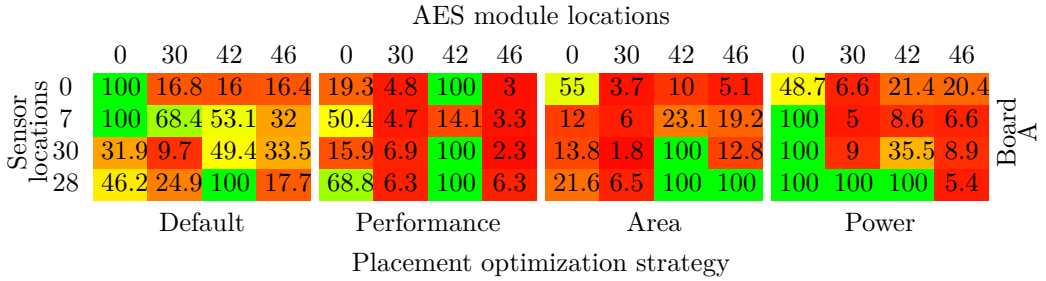


Figure B.2.: Minimum amount of traces ($\times 1\,000$) required to attack the second round key byte on board A. Red colored cells correspond to combinations where an attack is easy, green colored to parameters resulting in a difficult attack. A value of $100k$ means that we were unable to recover the key with $100k$ traces.