EPFL

Thèse n° 9918

Evaluating, Exploiting, and Hiding Power Side-Channel Leakage of Remote FPGAs

Présentée le 29 août 2023

Faculté informatique et communications
Laboratoire d'architecture de systèmes parallèles
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Ognjen GLAMOČANIN

Acceptée sur proposition du jury

Prof. P. Thiran, président du jury
Prof. B. Falsafi, Dr M. Stojilović, directeurs de thèse
Prof. R. Kastner, rapporteur
Dr M. Macchetti, rapporteur
Prof. D. Atienza, rapporteur

École polytechnique fédérale de Lausanne

2023

The moment you doubt whether you can fly,

you cease for ever to be able to do it.

— J. M. Barrie, Peter Pan

To Teodora, the love of my life.

Acknowledgements

I came to EPFL in September 2018 to start my PhD journey, expecting it to be challenging. After all, both my bachelor's and master's studies were extremely difficult, but with dedication and hard work, I managed to finish them successfully. What I did not foresee at the time was just how challenging that journey would be, and if it were not for the support of all the wonderful people around me, I would not have made it to the finish line. I owe my deepest gratitude to all these people. Thus, it is fitting to start this thesis by extending my heartfelt thanks to them.

First, I am grateful to my advisors, Dr. Mirjana Stojilović and Prof. Babak Falsafi. It goes without saying that my supervisor, Mirjana, was the most influential presence throughout this journey. Since the beginning, she pushed me to be better and more efficient in everything I do. She taught me how to articulate and present my ideas clearly and confidently, both in writing and when presenting. She made me question everything, making me the researcher I am today. With her constant dedication and drive, there were moments when I felt like she cared more about my PhD than myself, and I thank her for that. Babak was my mentor throughout the PhD journey. He was always there when I needed advice and provided support whenever I needed it. He taught me the importance of clarity: clarity of thought, clarity in writing, and clarity in presentations. Besides work, Babak taught me about the other vital things in life: food and drinks. Most notably, he sparked my love for espresso and later entrusted me to be the coffee tsar at PARSA, which I will forever be grateful for.

Next, I would like to thank the members of my thesis committee, Prof. Patrik Thiran, Prof. David Atienza, Prof. Ryan Kastner, and Dr. Marco Macchetti, for their guidance and feedback during and after my thesis exam. Ryan is one of the kindest people I ever met, and his feedback

during the end stages of my PhD was instrumental in shaping my research and thesis. I also wanted to thank Prof. Mathias Payer, my EDIC mentor. I would not be here without his kind support. He was always happy to take time and listen to me, and his advice reshaped how I look at research.

I am very grateful to have had an extraordinary set of close collaborators. Prof. Francesco Regazzoni was my go-to hardware security expert, whose cool composure toward research always calmed me down during incredibly stressful paper deadlines. Prof. Mathias Payer always provided very insightful comments that made me rethink my approach for the better, leading to a significantly better paper. The other closest collaborators I am incredibly thankful to are Dina G. Mahmoud, Shashwat Shrivastava, David Spielmann, Louis Coulon, Hajira Bazaz, Anđela and Staša Kostić, Nour Ardo, Jinwei Yao, Kostas Papagiannopoulos, Dorian Ros, Melissa Azouaoui, Andrea Caforio, Daniel Collins, and Subhadeep Banik. I immensely appreciate their contribution to enriching my doctoral journey. I also want to thank all the students I have worked with throughout my PhD, who motivated me to be a better teacher and supervisor. Thank you to Alexandre Abbey, Dorian Ros, Léa Micheloud, David Spielmann, Hajira Bazaz, Arthur Passuello, Jinwei Yao, Nour Ardo, David Dervishi, and Anđela and Staša Kostić.

Next, I want to thank the people who did not directly participate in my PhD journey but nevertheless left a significant impact on me and my overall academic path. First, Prof. Laslo Nađ, whose enthusiasm to understand *why and how* an idea came to be, shaped how I look at science and engineering during my Bachelor's studies. Then, Prof. Staniša Dautović, whose passion for science and research motivated me to pursue a PhD and do research. And finally, Prof. Rastislav Struharik, who gave me a strong technical background in FPGAs I heavily relied on during my PhD journey. Finally, I would like to thank my middle school teachers, Senja Stavrić, who taught me what it means to have and meet high expectations, and Danica Gavrilović, who sparked my interest in technical sciences and engineering.

Next, I would like to acknowledge all the PARSA members, who added much-needed joy and color to my PhD journey. First and foremost, I want to thank Siddharth Gupta. Sid and I shared most of our PhD journeys and bonded very closely in my second year, probably because we were the only ones coming to the office at the time. I want to thank him for being a good

listener and giving me some of the best advice I have gotten during my PhD. He always listened to my angry rants about random and most minor stuff. Above all, I want to thank him for listening to all my, as he would call it *hyperventilating*, when my anxiety took over, and I would spiral on negative things. I think Sid has singlehandedly prevented an early end of my PhD more times than I can count. Thank you, Sid, for being the best friend I needed throughout this journey. Next, I want to thank Mario Drumond. Even though Mario and I overlapped for only one year at PARSA, he is still a very important person in my path. He may not realize it, but he saved me from "financial ruin" during my PhD by helping me and my then-unemployed wife get subsidies for health insurance in my first year. I started hanging out with Mario more after he graduated, and he significantly helped me handle the stress of the PhD and job search with concise and constructive advice. Also, he was my right-hand man in making fun of poor Sid. Next, I want to thank Arash Pourhabibi. He was my first office mate and one of the warmest and most supportive people I ever had the chance to meet. He was always ready to take time, listen, and give sage advice that would completely change how I look at my problems, suddenly making them manageable. When he graduated, he also became my gaming buddy, and we shared many discussions about our favorite games. When I came to Switzerland, I was a big softy and could not tolerate spicy food. I am incredibly grateful to Arash and Sid, who taught (forced?) me to eat spicy food, and opened my eyes to a whole new world of fantastic tastes. While not in the lab, I always felt Negar Foroutan was part of the PARSA gang, at least spiritually. I am very grateful to have met her, as she always helped with the most selfless intentions. I always enjoyed having amazing history/linguistics conversations with her, through which I learned much about Iran.

I am also grateful to have spent much time with my two Turkish friends, Ahmet Caner Yüzügüler and Simla Burcu Harma. Simla and Ahmet are complete and total opposites, but both are amazing people. One comes from the *cool* part of Turkey, and the other from the *lame* part. They both agree on this, though I am not sure they mean the same thing. Simla's optimism always helped me focus on the *amazing* things in life, and I thank her for that. Ahmet's relaxed attitude helped me realize that life is not all about work and stress; that we should relax from time to time, be it by chilling at the lake, or eating some spicy crackers

and watching Pineapple Express. I am also really grateful for having the SOL team, Shashwat Shrivastava and Louis Coulon. Shashu and Lulu have been very supportive throughout the last stages of my PhD, and were always ready to listen to my rants. They are both very grounded and positive people who helped me in my research, but also pushed me to stand up for myself. I also want to thank Dina for all her help and support. We were working in the same area of research, and discussions with her helped me clarify and understand many things. She was always happy to help with both technical, but also personal issues. Even though for a short time, I am incredibly grateful to have met Hussein Kassir in my first year. We both quickly realized we shared the same weird sense of humor and later became Reddit buddies. Hussein and I stayed in regular contact, and he was extremely helpful with my job search, advising me and providing me with resources to prepare for hardware interviews. I also want to thank Atri Bhattacharyya. We started our PhDs at the same time, and connected on a lot of topics, both technical and non-technical. Atri always amazed and motivated me with his ability to ask really good questions at every talk I attended with him. He organized great board game nights, where I would usually miserably lose to either him or his friend. I am also grateful to have spent time with Mark Sutherland, maybe the biggest overthinker I know. I want to thank him for all the long discussions, both philosophical and technical. But also, for sending me all those amazing Key and Peele videos; these guys are really cool. I would also like to thank all the other PARSA members whom I enjoyed interacting with and that made PARSA the amazing and lively place it is: Rafael Pizarro Solar, Yuanlong Li, Shanqing Lin, Ali Ansari, Ayan Chakraborty, and Buğra Eryılmaz. Finally, I would like to thank Stéphanie Baillargues, for helping me in anything and everything related to EPFL administration. She was always ready to go above and beyond to do the crazy things that came to our minds, from renovating the lab, to organizing trips and reimbursing receipts, and doing it all faster than the speed of light. Next, I also want to thank my EPFL PhD colleagues from other labs that I was very close to throughout my PhD journey. First, I want to thank the *Kashmiri* gang: Mahyar Emami, Sahand Kashani, Xinrui Jia, and Rishabh Iyer. Mahyar and Sahand were my FPGA buddies, and we shared a lot of the sweat and blood needed to use FPGAs. Mahyar and I started the PhD at the same time and immediately bonded on gaming, movies, and TV shows. Mahyar lent me

his PS4 one day, and doomed me for the rest of my life, for which I am unironically incredibly thankful, as he revived my long-lost love for gaming. Whenever somebody asks me about Sahand, I say he is the greatest FPGA wizard I ever knew. He is one of the most disciplined and driven people I have ever known. If you give Sahand enough time, he will make miracles possible. He and I share an intense need for cleanness which we have first bonded on, but we later realized we have much more in common, and I really enjoyed spending time with him and discussing various technical and non-technical topics. I also wish to thank Sahand and Mahyar for graciously letting me use their Alveo U200 FPGA board. Their scripts and guidance helped significantly reduce my workload, and I used their infrastructure for most of the papers in my senior years. I am also very grateful to have met Xinrui Jia. We started our PhD at the same time, and she has been an incredibly good friend throughout the five years of our PhD journeys. She was always ready to listen and help whenever needed. Xinrui is one of the most reliable people I have met. If she agrees to do something, it will be done perfectly. She was always happy to water my plants whenever I went out of town, and was one of the very few people I trusted enough to do this surprisingly complicated procedure (maybe because we share a love for our beautiful orchids). Finally, I want to thank Rishabh. He is probably one of the most competitive people I have ever seen. I will never forget the look on his face when I won an unbelievable poker hand. However, I incredibly admire that he uses his competitiveness to push himself and improve, instead of pushing others down. Like Mario, he was an amazing accomplice for pranking Sid.

Next, I want to thank the LAP gang, including Stefan Nikolić, Grace Zgheib, Sahand Kashani, Lana Josipović, Mikhail Asiatici, Louis Coulon, and Andrea Guerrieri. While PARSA was my home, my research community was not in computer architecture. My research centers on FPGAs, in which the people from the LAP gang are the experts. Whenever I went to FPGA conferences, they made me (and my wife) feel like a part of their group, invited me to dinners and trips, and introduced me to various experts in the community. Thank you all for adopting me into your FPGA family and providing me with the fondest memories during my conference trips.

I am also incredibly indebted to the amazing people I have known for years in Serbia, who

supported my path towards a PhD from 1,000 km away. First, I would like to thank Vojin Jovičić, whom I met when I was three, and who has been a constant support throughout my life. We went through a lot together, from kindergarten to my marriage, at which he was the best man. We went everywhere on roller skates together, and played Yu-Gi-Oh cards and video games. We shared our love for music, and even played in a band together. We played pretty bad, but we had a blast, like we always do. He was the one that drove me to Lausanne by car the day I started my PhD. Throughout my PhD, the stories about his (mis)adventures have made me smile on many a dark day. He was always there when I returned to visit Serbia, and bent over backwards to spend as much time with me as possible. He is the best friend anyone could ask for, and I am proud to have shared most of my life with him. I am also very grateful for my middle school group of friends, including Vojin Jovičić (again), Dragan Gobla Špančić, Bojan Jagetić, Nikola Kričković, and the latecomer, Miloš Vlaškalić. I spent so much time with these people throughout middle and high school, just chilling on the bench, drinking coke or beer, and talking about the most random stuff ever. Thank you for filling my teenage years with amazing memories.

I am also incredibly grateful to my two great friends at EPFL from my high school choir days, Stefan Nikolić and Viktor Šanca. Stefan and I met in a mosh pit at a punk metal concert at high school. Ever since then, we have been great friends. We quickly discovered we share many similar interests, from music to our love of engineering, steam trains, sailships, and hiking. After high school, we attended the same university and followed all the classes together. He always had an encyclopedic knowledge, and discussions with him really helped me gain a deeper understanding of the material we were working on. Stefan is the one who convinced me to apply to EPFL, and I am incredibly grateful for that. Like Stefan, Viktor and I sang in the same high school choir. We were both tenors and classical music geeks, so we quickly bonded. In Switzerland, Viktor was my go-to guy for anything law- or regulation-related. Somehow, he always knew exactly where the answer to my question was, and pointed me to the precise line of the exact law that had the answer. I should add that he is not Swiss, at least not officially. He was always a great listener, and we enjoyed ranting to each other about various things throughout our PhD journeys.

I am also very grateful to my two high school friends, Ognjen Stanojev and Aleksa Nikolić. Ogi has always been a cool and composed presence in my life, always there to listen and try to help. He has lodged me more times than I can count, both in Zurich and the US. Whenever I hang out with Aleksa (usually with Ogi, too), we always end up having unexpected fun. We end up in crazy places doing crazy things, and I will always remember our trips together.

I would also like to thank my *Jules Verne* family, with which I learned French and had amazing adventures since I was a kid: Vojin Jovičić, Lena Pucarević, Anja Jurić, Jelisaveta Džigurski, Nemanja Jevtović, Jelena Begić, Marija Drapšin, Miloš Ubović, Marko Jurić, and Natalija Čomić. I made sure to see them whenever I returned to Serbia, and every time, even though years and months passed, it was as if not a single day had passed. I am very grateful for their support throughout my PhD journey. I am also very grateful to my amazing group of friends from my university days in Novi Sad, including Stefan Nikolić, Dajana Milićević, Jovana Zoranović, Stefan (Stena) Stanić, Kosta Ivančević, and Predrag Kovačević. Our many discussions and study sessions helped me better master the material, and hanging out with them made my university days incredibly fun. I am also incredibly grateful to all the choir friends that I met and sang with throughout the years, from both the "Jovan Jovanović Zmaj" high school, led by Jovan Travica and Jasmina Nešković, but also the "Klapa Capa" a capella group, including Viktor Šanca, Nikola Jovanović, Miloš Stojanović, Staša Stajić, Branislav Glogovac, Zoran Otrupčak, Stefan Nikolić, Goran Stanić, and Uroš Dobrić. Singing in these two choirs not only allowed me to meet my wife and fall in love, but also allowed me to express my love for singing with many like-minded friends.

Next, I want to thank all my friends from Sorbonne Université, including Yohan Fargeix, José-Paul Dominguez, David Bananier, and Arthur Hennequin, who helped me catch up when I joined the second year of the two-year master's program. I learned a lot from them, and working and hanging out was a pleasure. I am also grateful to Aleksandar Miladinović, whom I met when I first came to our dorm in Paris. He is an amazing person I enjoyed cooking with and visiting Paris, making my time in Paris super fun.

I am also incredibly grateful for the people from ARM I used to hang out with during my internship. Abdelhadi Moustafa is an amazing guy who taught me much about Lebanese food

and culture. I also enjoyed discussing TV shows with him, and I think he is the only person that *truly* understands what binge-watching a whole TV show means. Yohan Fargeix has unknowingly helped me deal with a lot of problems during the internship with his unbelievably positive attitude. I think he is one of the rare people I know who almost never worries about anything, and his attitude rubbed on me and made my time at ARM much more fun. Even though I have not talked to Maria Teresa Bevivino much during my internship, I greatly enjoyed spending time with her later on, during many of our trips around Europe with Yohan and Abdelhadi. JP is one of the kindest and tamest people I have ever met, and I also greatly enjoyed spending time with him during my internship. He drove us to so many beaches that summer that his poor car just decided to give up at the end of August. We are still waiting for the gas bill from JP.

Finally, I would like to extend my immeasurable gratitude to my family. They are the main reason I am where I am right now, and without their unconditional love and everlasting support throughout my entire life, I would not have reached the end of my PhD journey. I want to thank my parents, Jelena and Zoran, for believing in me since day one and investing so much time, energy, and resources in me, my education, and my well-being. I am incredibly thankful for all the sacrifices that they made. I am grateful for all the little (and big) details: the hugs, the walks in nature, the travel to imaginary worlds and real places all over Europe, the support in my education, English, French, and piano lessons, the encouragement to study abroad, and the countless other selfless things they did for me that I am maybe not even aware of. I also want to thank my two brothers, Dušan and Nikola, for always being there for me and sharing their childhood with me. I always fondly remember our time together, building LEGOs, watching movies, playing video games, and laughing. I am proud of what you are now, and I am looking forward to spending more time with you in the future. I am also very grateful to my grandma Cecilija, grandpa Dušan, and aunt Jelena, who have always been there for me since I was a child, and took me on wild adventures all over Serbia. Some of my fondest memories are from spending time with them, at their home or our country-side house, and I will always be thankful for their support. I would like to acknowledge another very important person in my life, my grandmother Katica, who passed away shortly before I started my PhD. I am

very grateful to have had her in my life. She used to spend so much time with me when I was little, taking me on adventures all around Novi Sad. When I was having trouble in elementary school, she sat me down and taught me how to study, and to this day, I use her advice, for which I am forever grateful. I also want to extend my gratitude to my in-laws, Valerija, Relja, and Fedor Popović, who have always considered and treated me as their own. I would like to thank them for their love and support throughout my PhD journey.

Last but not least, I would like to thank my wife Teodora, the love of my life, and my *giving tree*. She is the main reason I am where I am right now, and I would not have had the strength to finish this journey without her. Doing a PhD was the hardest thing I have ever done in my entire life, by far. I think my PhD journey was also one of the hardest things she did, too. She left everyone she knew and loved to move across the Alps to be with me, for which she will forever have my gratitude. And during the difficult periods of my PhD—and there were *a lot* of them—she put her problems aside and always unconditionally supported and encouraged me. She has been my rock, my safe place, the first person I want to see in the morning, and the last person I want to see before I go to sleep. I am forever grateful to have a person in my life that unconditionally believes in me, and constantly reminds me to stop doubting that I can fly. I love her more than anything else in the world, and I look forward to spending the rest of my life with her.

I would also like to thank the people and organizations that financially supported my PhD journey. I am thankful to EPFL, the Swiss National Science Foundation, and finally, the Swiss people for supporting me through their taxes.

Lausanne, July 17, 2023 Ognjen Glamočanin

Abstract

The pervasive adoption of field-programmable gate arrays (FPGAs) in both cyber-physical systems and the cloud has raised many security issues. Being integrated circuits, FPGAs are susceptible to fault and power side-channel attacks, which require physical access to the victim device. However, recent work demonstrated that physical proximity is no longer required for these attacks, as FPGA logic can be misused to create on-chip voltage sensors or power-wasting circuits. The work in this thesis explores the impact of FPGA-based voltage sensors on FPGA security and shows that sensors create new opportunities to evaluate, exploit, and hide power side-channel leakage in remote FPGAs.

In the first part of this thesis, we demonstrate that voltage sensors can increase power side-channel security. In the case of deployed and no longer accessible cyber-physical devices, we show that FPGA-based voltage sensors allow designers to evaluate the power side-channel leakage after deployment, ensuring constant power side-channel security monitoring. Our results, comparable to state-of-the-art measuring equipment, move the leakage evaluation boundary from controlled lab environments to the field, allowing future work to combine leakage evaluation with other security measures such as tamper detection.

In the second part of the thesis, we focus on evaluating the security impact of FPGA-based voltage sensors on multitenant FPGAs, and show that voltage sensors can evaluate, exploit, and hide power side-channel leakage. We demonstrate that a remote attacker can mount both statistical (correlation power analysis) and machine learning based attacks with the voltage sensors, emphasizing the need to deploy countermeasures in multitenant FPGAs. The work in this thesis was the first to show a successful remote power analysis attack on cloud FPGA instances and the first to provide an instruction-level leakage analysis of soft-core CPUs in a

shared FPGA scenario. Motivated by the exploits, this thesis proposes a novel hiding technique against remote power side-channel analysis attacks: active wire fences. Our results show that active wire fences outperform the state-of-the-art hiding techniques in shared FPGAs.
In the last part of the thesis, we explore more efficient and stealthy techniques for sensing on-chip voltage. We present the first stealthy routing-based FPGA sensor that outperforms the state of the art in remote power analysis attacks. With our stealthy sensor architecture, we show that detecting sensor circuits is not a scalable solution for guaranteeing security. Finally, this thesis evaluates the impact of external factors, specifically temperature, on FPGA-based voltage sensors and the success of remote power side-channel attacks in multitenant FPGAs. Our work shows that, if ignored, temperature effects on voltage sensors can lead to misleading attack results.

Keywords: reconfigurable logic and FPGAs, multitenancy, power analysis attacks, on-chip sensors, cloud, CPU instruction identification, hiding, temperature impact

Résumé

L'adoption généralisée de systèmes FPGA dans les systèmes cyber-physiques et dans le cloud a soulevé de nombreux problèmes de sécurité. En tant que circuits intégrés, les FPGAs sont sensibles aux attaques d'analyse de consommation, qui nécessitent un accès physique au dispositif victime. Cependant, des travaux récents ont démontré que la proximité physique n'est plus nécessaire pour l'éxécution de ces attaques, car les ressources internes des FPGAs peuvent être utilisées à mauvais escient pour créer des capteurs de tension directement sur la puce, ou des circuits qui gaspillent de l'énergie. Cette thèse explore l'impact que peut avoir, sur la sécurité des FPGAs, de capteurs de tension basés ces derniers. Nous montrons que ces capteurs créent de nouvelles opportunités pour évaluer, exploiter et dissimuler des fuites issus de canaux latéraux d'alimentation dans les FPGA distants.

Dans la première partie de cette thèse, nous démontrons que les capteurs de tension peuvent augmenter la sécurité. Dans le cas de dispositifs cyber-physiques déployés et devenus inaccessibles, nous montrons que les capteurs de tension basés sur les FPGA permettent aux concepteurs d'évaluer la présence de fuites d'information à travers un canal latéral de puissance après que le système FPGA ait été déploié, assurant ainsi une surveillance constante de la sécurité. Nos résultats, comparables à ceux obtenus avec de l'équipement de mesure de pointe, déplacent la limite d'évaluation des fuites depuis un environnement de laboratoire contrôlé vers le terrain, ce qui permettra à l'avenir de combiner l'évaluation des fuites avec d'autres mesures de sécurité telles que la détection de sabotage.

Dans la deuxième partie de la thèse, nous nous concentrons sur l'évaluation de l'impact sécuritaire des capteurs de tension on-chip sur les FPGAs multi-utilisateur, et nous montrons que les capteurs de tension peuvent évaluer, exploiter et dissimuler des fuites de canaux

latéraux de puissance. Nous démontrons qu'un assaillant distant peut déployer des attaques statistiques (analyse de puissance par corrélation) et des attaques basées sur l'apprentissage automatique avec les capteurs de tension, soulignant la nécessité de déployer des contre-mesures dans les FPGAs multi-utilisateur. Les travaux de cette thèse ont été les premiers à montrer une attaque réussie d'analyse de puissance à distance sur des instances FPGA dans le cloud, et les premiers à fournir une analyse de fuite au niveau des instructions des CPU soft-core dans un scénario FPGA partagé. Motivée par ces exploits, cette thèse propose une nouvelle technique de dissimulation contre les attaques d'analyse de canal latéral de puissance à distance : les clôtures de fils actives. Nos résultats montrent que les clôtures de fils actives sont plus performantes que les techniques de dissimulation de pointe dans les FPGAs partagés.

Dans la dernière partie de la thèse, nous explorons des techniques plus efficaces et plus furtives pour détecter la tension sur une puce. Nous présentons le premier capteur FPGA furtif basé sur le routage qui surpasse l'état de l'art dans les attaques d'analyse de puissance à distance. Avec notre architecture de capteur furtif, nous montrons que la détection des circuits de capteurs n'est pas une solution qui peut être mise à l'échelle pour garantir la sécurité. Enfin, cette thèse évalue l'impact des facteurs externes, en particulier de la température, sur les capteurs de tension basés sur les FPGAs et le succès des attaques par canal latéral de puissance à distance dans les FPGAs multi-utilisateur. Notre travail montre que, s'ils sont ignorés, les effets de la température sur les capteurs de tension peuvent conduire à résultats trompeurs en ce qui concerne la réussite ou l'échec d'une attaque.

Mots clés : logique reconfigurable et FPGAs, multi-utilisateur, attaques d'analyse de puissance, capteurs on-chip, cloud, identification des instructions CPU, dissimulation, impact de la température

Contents

List of Figures

List of Tables

List of Abbreviations

AES advanced encryption standard

ASIC application-specific integrated circuit

BRAM block RAM

CLB configurable logic block

CNN convolutional neural network

CPA correlation power analysis

CPU central processing unit

CSP cloud service provider

CWT continuous wavelet transform

DCM digital clock manager

DL deep learning

DoS denial-of-service

DPA differential power analysis

DSP digital signal processing

EM electromagnetic

FF flip-flop

FIFO first-in-first-out

FPGA field-programmable gate array

FSM finite state machine

GDM Gaussian diffusion model

GPU graphics processing unit

HD Hamming distance

ISA instruction set architecture

ITD inverse temperature dependence

k-NN k-nearest neighbors

KL Kullback-Leibler

LDA linear discriminant analysis

LFSR linear-feedback shift register

LSTM long short-term memory

LUT look-up table

ML machine learning

MLP multi-layer perceptron

MMCM mixed-mode clock manager

PCA principal component analysis

PCB printed circuit board

PDN power delivery network

PID proportional-integral-derivative

PLL phase-locked loop

PRNG pseudo-random number generator

QDA quadratic discriminant analysis

RDS routing-delay sensor

ResNet residual network

RFC random forest classifier

RNN recurrent neural network

RO ring oscillator

RR routing resources

RSA Rivest-Shamir-Adleman

SB switch box

SCA side-channel analysis

SLR super-logic region

SNR signal-to-noise ratio

SPA simple power analysis

SVM support vector machine

TDC time-to-digital converter

TPU tensor processing unit

TRNG true random number generator

TVLA test vector leakage assessment

VM virtual machine

VTA versatile tensor accelerator

1 Introduction

Due to the end of Moore's law and the breakdown of Dennard's scaling, compute-intensive systems are transitioning from homogeneous and processor-dominated systems toward more heterogeneous architectures which rely on specialized hardware to improve performance and energy efficiency. As a result, today's computing systems feature not only central processing units (CPUs) but also graphics processing units (GPUs) and special-purpose integrated circuits such as application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs). Compared to ASICs, using FPGAs significantly reduces design time, development, and deployment costs. The flexibility of FPGAs, coupled with their highly-parallel architecture and energy efficiency, has led to their integration into various heterogeneous systems—from small embedded and cyber-physical systems to datacenters and, recently, the public cloud of major cloud service providers (Microsoft Azure [2], Amazon AWS [3], and Alibaba Cloud [4]).

The growing awareness of the importance of security and privacy and the increased regulatory and legal attention to data makes security not only desired but, in many cases, required in modern systems. Industry and research trends have seen an increase in attention to cybersecurity, trusted execution environments, and confidential and trusted computing [5, 6]. Therefore, apart from satisfying the short design time and ever-growing complexity of the systems, designers of FPGA-based systems face a new challenge: ensuring security and privacy for the system end users. Consequently, previous work has explored FPGA security in heterogeneous systems,

covering vulnerabilities such as fault injection or side-channel analysis (SCA) attacks [7]. In particular, power SCA attacks—targeting information inadvertently leaked through the power consumption or supply voltage fluctuations—can be easily mounted by attackers with physical access to the device [8, 9]. In power SCA attacks, an adversary collects a number of power measurements, typically with an oscilloscope, and applies statistical or machine learning techniques to infer algorithmically secure information. These attacks represent a critical class of side-channel attacks, as even a limited number of power measurements can contain leakage sufficient to extract confidential data, such as encryption keys, from unprotected devices [10].

Recent research demonstrated that physical proximity is no longer required for SCA power measurements on FPGA, as voltage fluctuations occurring during the computation of a hardware module, for example, a cryptographic accelerator, can also be captured by dedicated circuits realized in the FPGA logic itself, such as time-to-digital converter (TDC) converter sensors or ring oscillators [11–14]. Data collected with these voltage-drop sensors could then be used instead of the classical power measurements collected by an oscilloscope to successfully recover confidential information. Successful examples of these attacks directly exploited the relation between circuit activity and the voltage fluctuations on the power delivery network (PDN) to extract encryption keys of a cryptographic core [12, 13] or data from machine learning (ML) accelerators [15, 16].

In this thesis, motivated by the possibility of performing remote voltage measurements, we investigate the implication of voltage-drop sensors on the security of remotely accessible FPGAs. We identify two prominent use cases of remote FPGAs—cyber-physical devices deployed in the field and cloud FPGAs—and we show that on-chip voltage sensors can be used to both evaluate and compromise the security of FPGA-based systems.

1.1 Thesis Goals

Often accessible to attackers, deployed cyber-physical devices remain susceptible to physical attacks. In the case of both protected and unprotected devices in the field, adversaries can

resort to more invasive methods to increase the power side-channel leakage and reduce the attack effort, such as tampering with the PDN [12, 17]. Moreover, effects such as aging, malfunctioning, or even maintenance can also impact the leakage. However, the device leakage is only verified before deployment, in a controlled scenario, and is almost never reevaluated after deployment. As its **first objective**, this thesis explores the possibility of leveraging on-chip sensors to provide continuous monitoring and evaluate the device's vulnerability to first-order power SCA attacks [18].

In the cloud, a multitenant ecosystem is already in place, and the available computational resources—namely CPUs—are shared between system users. To enable better management and use of available datacenter resources, increased efforts are being made to support secure virtualization and sharing of FPGA hardware acceleration fabric [19–31]. However, spatial sharing, i.e., co-locating multiple users on the same FPGA, implies many security issues, primarily due to the access to low-level hardware primitives and electrical-level coupling via the shared PDN or long wires [32, 33]. As a **second objective**, this thesis tackles power side-channel security in shared, i.e., multitenant FPGAs. It uncovers and analyzes new security vulnerabilities introduced by attackers with access to low-level FPGA logic—power SCA attacks on encryption and soft processor cores using on-chip voltage sensors [14]—and explores new, more efficient ways of mitigating these vulnerabilities specialized for shared FPGAs.

As a **third objective**, this thesis aims to explore new, more efficient, and stealthy techniques of sensing on-chip voltage in shared FPGAs [34]. Finally, this thesis aims to evaluate the impact of external factors, specifically temperature, on on-chip voltage sensors and the success of remote power side-channel attacks in multitenant FPGAs [35].

Thesis statement:

Access to fine-grained FPGA hardware in modern heterogeneous systems allows constructing efficient on-chip voltage sensors, which create new opportunities to evaluate, exploit, and hide power side-channel leakage in remote FPGAs.

1.2 Thesis Contributions

As indicated in the thesis title and statement, this work extends the state of the art alongside three main pillars of power side-channel leakage in remote FPGAs: evaluating, exploiting, and hiding. Fig. 1.1 illustrates the main contributions of this work. Alongside the evaluating axis, this thesis leverages FPGA-based voltage sensors to enhance security by providing continuous leakage monitoring and methodologies to evaluate the power side-channel security of deployed devices. Our work uses FPGA-based voltage sensors to demonstrate new vulnerabilities in FPGA-based systems, enhancing the state of the art alongside the exploiting axis. Finally, our work contributes to the hiding axis by leveraging low-level FPGA logic to reduce the power side-channel leakage, while using the FPGA-based sensor to record voltage traces.

We organize the thesis contributions into three main parts to align with the thesis goals outlined in Chapter 1.1. Part I centers around evaluating the security of deployed cyber-physical devices, contributing to the evaluating axis. Part II, contributing to all three axes of power side-channel security, centers around multitenant FPGA security. Lastly, Part III, contributing to the evaluating and exploiting axis, centers around advanced sensor architectures and temperature impact on sensors. In the following subsections, we describe the contributions of each of the three main parts of the thesis.

1.2.1 Evaluating the Security of Deployed Cyber-Physical Devices

The leakage of cyber-physical devices can change during their service life for various reasons (tampering, aging, or malfunctioning). However, until our thesis, the device leakage was typically evaluated in laboratory environments before deployment and not reevaluated. Previous work demonstrated that FPGA-based voltage sensors require an order of magnitude more traces than an oscilloscope for a successful power analysis attack [12]. Our work shows that, despite lower-quality power measurements, FPGA-based voltage sensors can be successfully leveraged to evaluate power side-channel leakage in deployed devices. The work in this thesis was the first to present and validate a built-in test for self-evaluation of power side-channel

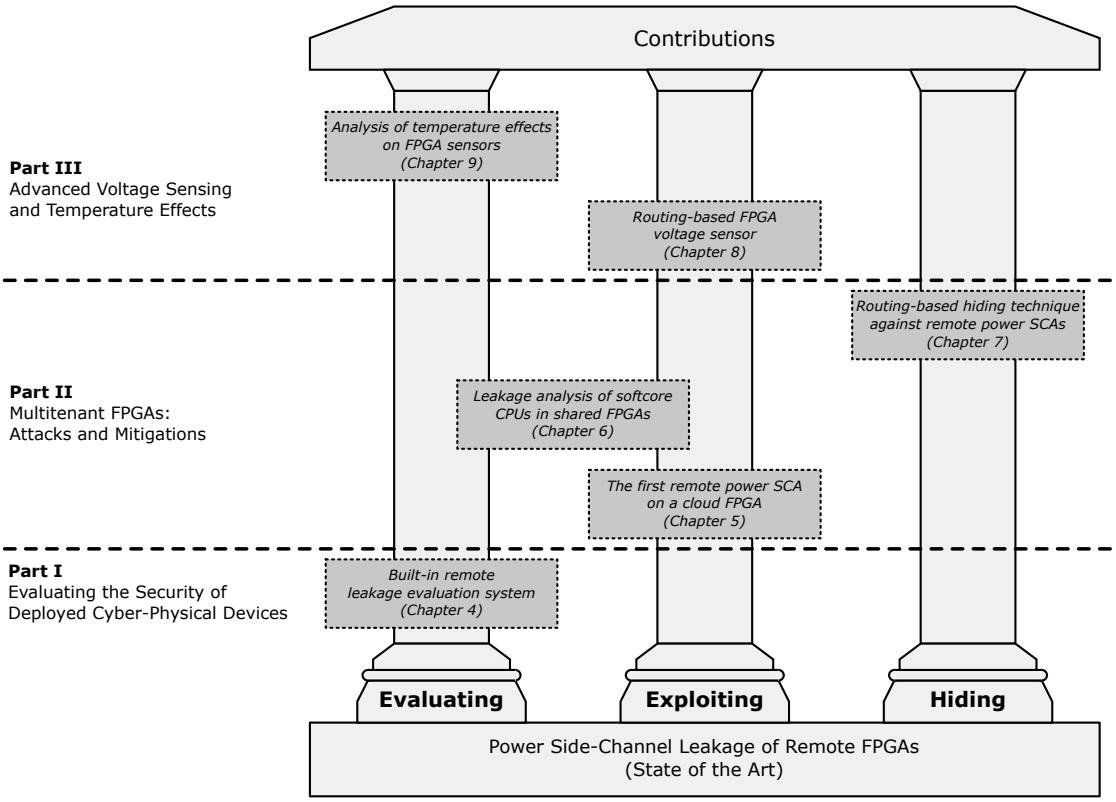


Figure 1.1: Thesis contributions supported by three main pillars: evaluating, exploiting, and hiding power side-channel leakage of remote FPGAs. Our work in Chapters 4, 6, and 9 presents the contributions along the evaluating pillar, extending the state of the art with new techniques for measuring leakage of deployed devices, measuring leakage of softcore CPUs, and analyzing the temperature impact on FPGA sensors. Our work in Chapters 5, 6, and 8 builds the contributions of the exploiting pillar, demonstrating statistical and ML-based attacks on cloud FPGAs and a new FPGA sensor architecture. Finally, the work in Chapter 7 contributes to the hiding pillar, improving the state of the art hiding techniques with new routing-based power-wasting circuits.

leakage, suitable for FPGAs in cyber-physical devices [18]. Our methodology is based on the t-test, the *de facto* industry standard for security assessments. The system consists of an FPGA-based voltage sensor [11] to measure the on-chip voltage fluctuations and an engine to calculate the t-test statistic on the fly and thus verify the presence of information leakage. Using an advanced encryption standard (AES) cryptographic accelerator implemented on FPGA, we show that our built-in test achieves results comparable to those obtained with state-of-the-art laboratory oscilloscope equipment. The system proposed in this thesis allows, for the first time, a real-time assessment of power side-channel leakage during the device

operation in the field. This work opens new frontiers for ensuring the security of safe and robust cyber-physical systems, which are currently verified only before deployment.

1.2.2 Multitenant FPGAs: Attacks and Mitigations

Previous work demonstrated that on-chip sensors and remote power SCA attacks present a potential threat to the security of multitenant FPGAs [12, 13]. These experiments were, however, mostly carried out on low- to mid-end FPGAs in a controlled environment and not on a real system deployed on the cloud. In contrast, most modern cloud FPGAs have updated resource architectures, 10–20× more resources, and a higher quality PDN. While giving the intuition that the threat of remote power analysis attacks could also be exploitable in a real cloud environment and high-end cloud FPGAs, previous work did not provide any clear evidence of this fact.

The work in this thesis demonstrates, for the first time, a successful key recovery attack on a cryptographic accelerator running on an Amazon EC2 F1 instance. Our research shows that the FPGA-based voltage sensors used in previous work should be carefully ported to new FPGA families to reduce the attack effort. The results in this thesis demonstrate that a correlation power analysis (CPA) attack can recover the full AES key using less than 0.5 million traces, which is two orders of magnitude higher compared to a controlled environment. We show that even though the effort for a successful attack on the cloud increases, the security concerns raised by multitenant FPGAs are valid, and countermeasures should be put in place to mitigate them. Moreover, our work motivated a growing amount of subsequent studies to showcase their attacks not only on local FPGA boards, but also on datacenter-scale FPGAs.

Previous work on remote power side-channel attacks, including our attack on Amazon F1 instances, requires many victim traces for statistical attacks on cryptographic circuits or attacks large, power-consuming victim circuits such as machine learning accelerators. However, confidential information extends beyond a user's bitstream, cryptographic key, or neural network accelerator parameters and architecture. If the user's design includes a soft-core

CPU, the executed code may contain secrets or be proprietary, and identifying the executed instructions can compromise the confidentiality of the code. Therefore, this thesis analyzes the instruction-level power side-channel leakage of a small open-source RISC-V soft processor core. We profile a soft-core CPU and implement various instruction opcode classifiers based on classical machine learning algorithms used in disassembly attacks and novel deep learning approaches. We explore how parameters such as placement, trace averaging, profiling templates, and different FPGA families (including a cloud-scale FPGA) impact the classification accuracy. Our results show that in a worst-case scenario for the evaluator, i.e., an attacker breaching physical separation, we can identify the opcode of executed instructions with high average accuracy. The results in this thesis also show that the instruction-level leakage is significantly reduced in a cloud-scale FPGA scenario with higher soft-core CPU frequencies. However, our results show that even small circuits, such as soft-core CPUs, leak potentially exploitable information through on-chip power side channels, and users should deploy mitigation techniques against disassembly attacks to protect their proprietary code and data.

Previous work proposed various mitigation techniques to protect against power SCA attacks on shared FPGAs, such as detecting and preventing the deployment of voltage-sensing circuits, reducing the signal-to-noise ratio (SNR) by hiding, and masking the victim design using random shares [33]. On the one hand, masking methods require modifying the victim design while detecting voltage-sensing circuits is prone to false positives and negatives. On the other hand, hiding techniques can be independent of the victim's application and do not require modifications to the victim's design, making them appealing to both the cloud service providers (CSPs) and users. In a mitigation context, this thesis presents a design of an active wire fence and demonstrates its ability to provide hiding protection against remote power side-channel attacks in shared FPGAs. The novel wire fence uses FPGA routing resources to draw more current and generate more noise than the state-of-the-art fence built solely with ring oscillators (ROs). Comparing the voltage drop resulting from the activation of the wire fence and the RO fence, this thesis shows that the RO fence compares to the wire fence of

approximately half the logic resources. Therefore, when the space is limited, active wire fences are a better alternative to their RO counterparts. Additionally, our results show that the wire fence provides stronger side-channel security than the state of the art: when comparing the attack effort to break a 128-bit AES key, we demonstrate that at least 6× more traces were required to break the key with the wire fence compared to the RO fence.

1.2.3 Advanced Voltage Sensing and Temperature Effects

The state-of-the-art FPGA voltage sensors, i.e., TDCs, rely on CARRY elements, which require careful vertical placement for optimal sensitivity. However, using dedicated CARRY elements limits the sensitivity, requires complex placement constraints, restricts the portability of the TDC sensors, and, more importantly, makes TDCs easily detectable by scanning tools. To further push the limits of remote power analysis attacks, this thesis presents a novel FPGA-based voltage sensor design, different from TDC and RO sensors. Our new routing-delay sensor (RDS), free of a tapped delay line, is the first sensor that leverages routing resources for sensing voltage variations. We evaluate RDS using the CPA attack and the key rank estimation metric, in an attack against an AES-128 module. The results in this thesis show that RDS outperforms TDC, and that for many different sensor and AES placements on the Sakura-X board, an attack with RDS requires, on average, 35% fewer traces to break the secret key. Our results confirm that RDS is considerably superior to TDC even on cloud-scale FPGAs (AMD Alveo U200): the key rank estimation in 25 experiment runs with approx. 2 million traces per run shows that, on average, RDS breaks half the key bits while TDC breaks fewer than 16 bits.

Finally, this thesis is the first to evaluate the temperature impact on TDC sensors in a multitenant FPGA scenario. The operating temperature of the TDC sensor affects the acquired traces, but its impact on the success of remote power SCA attacks has largely been ignored in the literature. Our results show that the temperature impact can vary across FPGA families and that the temperature variations during trace acquisition can considerably impact the success of the CPA attack. Our analysis also reveals that, if ignored, temperature effects on TDC sensors can lead to misleading and overly optimistic results of ML-based profiling attacks.

This thesis emphasizes the importance of following power SCA trace acquisition guidelines for minimizing the temperature effects and, consequently, obtaining a more realistic measure of success for remote ML-based profiling attacks: in some cases, ML models of incorrectly recorded traces can learn temperature variations instead of leakage, resulting in a misleadingly higher accuracy of up to 15%.

1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 motivates the research on electrical-level security issues caused by access to FPGA logic in heterogeneous systems. Chapter 3 provides the background by introducing FPGA architecture, power analysis attacks, and FPGA-based sensor architectures. Then, we present the contributions of this thesis, organized in three parts:

- **Part I** and Chapter 4 demonstrate how FPGA-based voltage sensors can be used to evaluate power side-channel leakage in remote FPGAs, by presenting the first built-in remote leakage evaluation methodology for deployed FPGA-based cyber-physical devices.

- **Part II** shows how FPGA-based voltage sensors and other FPGA logic can be used to evaluate, exploit, and hide power side-channel leakage of multitenant FPGAs. Chapter 5 focuses on exploiting power side-channel leakage, and demonstrates the first remote power analysis attack on a cloud FPGA. Chapter 6 evaluates and exploits the power side-channel leakage of soft-core CPUs in shared FPGAs. Finally, Chapter 7 focuses on hiding the power side-channel leakage, and shows a novel mitigation technique against remote power analysis attacks.

- **Part III** focuses on evaluating and exploiting the power side-channel leakage in more depth. Chapter 8 shows a novel voltage sensor design based on routing resources, outperforming the state-of-the-art TDC sensors, while Chapter 9 provides an analysis of the temperature effects on FPGA voltage sensors and remote power analysis attacks.

Finally, in **Part IV**, we present the related work in Chapter 10, and we conclude the thesis in Chapter 11.

1.3.1 Bibliographic Notes

This thesis was conducted under the supervision of my advisor, Dr. Mirjana Stojilović. Portions of this document are based on the following publications:

- **O. Glamočanin**, L. Coulon, F. Regazzoni, and M. Stojilović, *Built-In Self-Evaluation of First-Order Power Side-Channel Leakage for FPGAs*, Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'20) [18],
- **O. Glamočanin**, L. Coulon, F. Regazzoni, and M. Stojilović, *Are Cloud FPGAs Really Vulnerable to Power-Analysis Attacks?*, Proceedings of the 2020 Design, Automation and Test in Europe Conference and Exhibition (DATE'20) [14],
- **O. Glamočanin**, Shashwat Shrivastava, Jinwei Yao, Nour Ardo, Mathias Payer, and M. Stojilović, *Instruction-Level Power Side-Channel Leakage Evaluation of Soft-Core CPUs on Shared FPGAs*, under review for the Springer Journal of Hardware and Systems Security,
- **O. Glamočanin**, Anđela Kostić, Staša Kostić, and M. Stojilović, *Active Wire Fences for Multitenant FPGAs*, Proceedings of the 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS'23), **Best Paper Award nomination** [36],
- David Spielmann, **O. Glamočanin**, and M. Stojilović, *RDS: FPGA Routing Delay Sensors for Effective Remote Power Analysis Attacks*, IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES'23) [34],
- **O. Glamočanin**, H. Bazaz, M. Payer, M. Stojilović, *Temperature Impact on Remote Power Side-Channel Attacks on Shared FPGAs*, Proceedings of the 2023 Design, Automation and Test in Europe Conference and Exhibition (DATE'23) [35].

2 Security Implications of FPGAs in Modern Heterogeneous Systems

In this chapter, we motivate the use of FPGA-based heterogeneous systems and provide an overview of electrical-level security vulnerabilities arising from access to low-level FPGA logic.

2.1 Towards Heterogeneous Systems

Today's computing systems face limits that require fundamental changes in system architectures. Performance improvement is significantly hindered by both physical limitations, such as the end of Moore's law and Dennard scaling, and architectural limitations, including instruction-level parallelism and Amhdal's law. With the ever-growing need for higher performance—especially in the era of big data and machine learning—system architects need new computing paradigms to ensure latency, throughput, and power requirements.

Fig. 2.1 shows the difference between the transistor density predicted by Moore's law and state-of-the-art technologies. As can be observed, the technology scaling is in the slowdown phase of Moore's law, with 10× fewer transistors than predicted by Moore's law. Additionally, small transistor sizes led to the breakdown of Dennard scaling in 2008, as shown in Fig. 2.2. With an exponential increase in power consumption, power and energy have become key design constraints. Therefore, in addition to having a slower increase in transistors, system designers must use them more efficiently.

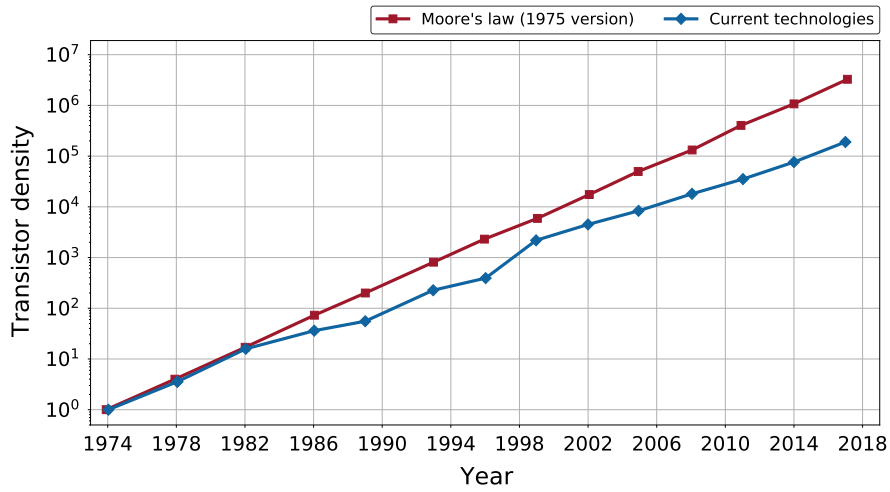


Figure 2.1: Difference between the transistor density predicted by Moore's law and the current state of the art [1].

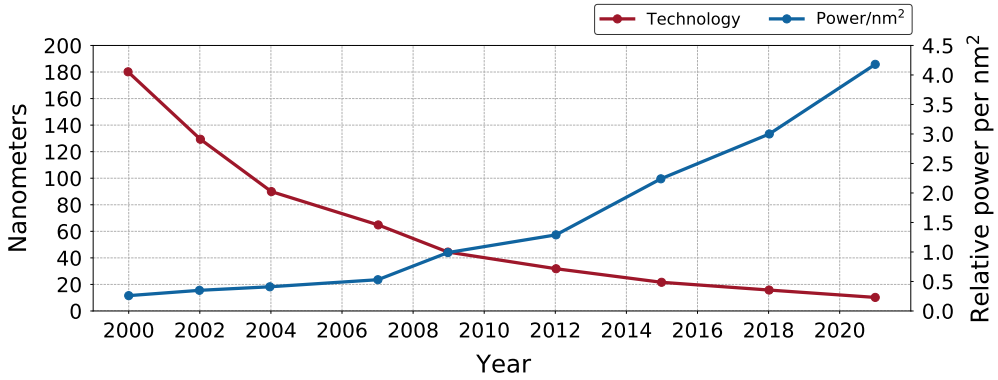


Figure 2.2: Breakdown of Dennard scaling. Power consumption in the function of the technology node [1].

However, systems also face architectural limitations: as we push the fundamental architectural ideas further, they become less efficient. For example, the breakthrough concepts in instruction-level parallelism brought immense performance improvements in the 90s but reached their limits in 2004, effectively ending the uniprocessor era. Multicore systems followed, but they also reached their limits because of Amhdal's law.

As general-purpose processor performance improvement reaches the end of the line, today's systems are becoming increasingly heterogeneous, and rely on domain-specific architectures to accelerate various workloads. Instead of relying on a general-purpose processor, heteroge-

neous systems include specialized circuits that perform specific tasks incredibly well, such as GPUs for graphic processing, tensor processing units (TPUs) for machine learning, FPGAs for hardware acceleration, or programmable network switches for networking. These specialized cores make more effective use of parallelism and memory bandwidth. The shift to heterogeneous systems can be observed in both the cloud, where vendors include various instances with different accelerators (GPUs, TPUs, FPGAs), and in embedded systems, where SoCs—including big-little CPU configurations, GPUs, FPGAs, and AI accelerators— have become the norm.

2.2 FPGAs in Heterogeneous Systems

FPGAs are highly reprogrammable integrated circuits that allow designers to create application-specific digital circuits. A key advantage of FPGAs is their flexibility: unlike ASICs, they are reprogrammable after fabrication and can implement arbitrary digital circuits. This flexibility makes FPGAs ideal for applications in which the specifications are subject to change over time and require fast time to market. In addition, their RTL programming model and hardware-level parallelism also allow low-level optimizations, resulting in low latency and better power efficiency than general-purpose CPUs.

FPGAs are often used in heterogeneous computing systems, ranging from cyber-physical systems and, more recently, to the cloud [2–4,37,38]. They are programmed to perform specific tasks, allowing them to be customized to meet the system requirements. As a result, FPGAs can accelerate specific computation, offloading work from other processors and improving overall system performance. Therefore, FPGAs are commonly used in heterogeneous systems to accelerate tasks that require real-time data processing, such as signal processing, image processing, cryptography, networking, and machine learning.

2.2.1 FPGAs in the Cloud

Microsoft pioneered the use of FPGAs in cloud computing. Their Catapult project pilot of 1,632 FPGA-enabled datacenter servers demonstrated a dramatic improvement in Bing search latency, launching the era of FPGA-accelerated cloud computing [39]. Other cloud service providers soon followed. Today, Amazon AWS, Microsoft Azure, Alibaba, Baidu, and Tencent offer their customers remote access to datacenter FPGAs, to develop, test, and deploy their custom hardware accelerators [3, 4, 37, 38].

FPGAs are typically exposed to remote users through a host CPU virtual machine interface and a *shell-role* use model [31]. The cloud service providers deploy the shell, which shares the FPGA logic with the users. The shell implements platform-specific management tasks: PCIe, direct memory access engine, DRAM controller, and debugging interfaces. The FPGA region reserved for each user is called a role, and users deploy their accelerators within their role. The shell-role separation helps faster accelerator deployment and ensures different privilege levels between the cloud service providers and the external users.

In both academia and industry, increased efforts are being made to extend multitenancy and resource virtualization from CPUs to FPGAs, to enable better management and use of available datacenter resources [19–31]. Multitenancy can be achieved through spatial and temporal multiplexing. Temporal multiplexing separates users in time, ensuring that each tenant gets their own, exclusive instance. In spatial multiplexing, FPGA roles are occupied by potentially different tenants, and consequently, the cloud service providers need to ensure security and privacy to all of them [29, 40, 41].

2.2.2 Security Concerns

Unlike other processing units in heterogeneous systems, FPGAs give users control over low-level hardware, such as look-up tables (LUTs), flip-flops (FFs), memory controllers, and even PCIe interfaces. Current FPGA compilation tools have virtually no constraints over how the logic can be used and connected, provided valid connections exist. This low-level access to

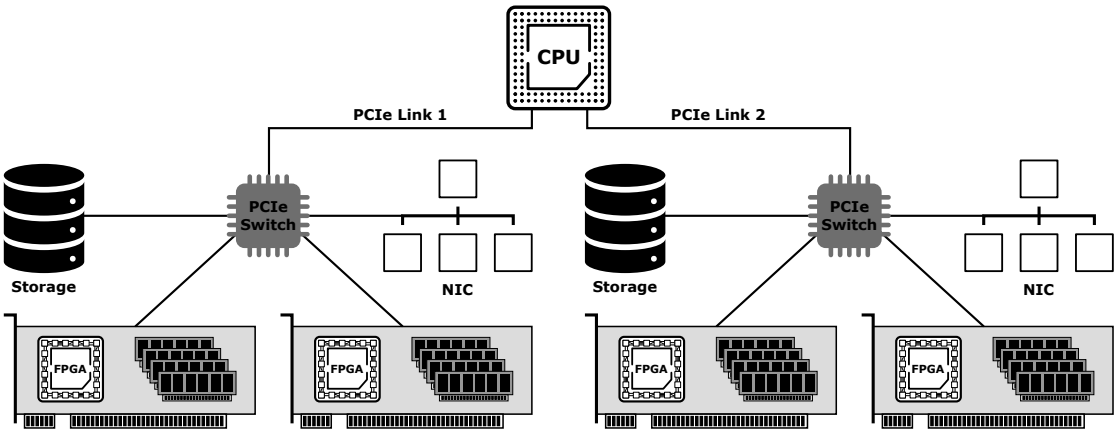


Figure 2.3: Architecture of cloud FPGA servers.

hardware resources brings many concerns, especially in security.

On the one hand, access to higher-level interface modules has serious system security ramifications, as seen in Fig. 2.3. For example, unrestricted access to the PCIe module allows attackers to access physical memory, bypassing security measures introduced by virtual memory abstractions in operating systems. Consequently, the shell-role model in cloud FPGAs allows only access to AXI interfaces— the PCIe modules are contained in the privileged shell. Nevertheless, in current cloud FPGA servers, multiple FPGAs can share the same CPU socket, and the PCIe interconnect. Even with a privileged shell, incorporating FPGAs in larger heterogeneous systems can negatively impact security. For example, previous work has shown that users can oversaturate the PCIe bus bandwidth and significantly slow down other tenants sharing the same PCIe interconnect [42].

On the other hand, access to low-level logic resources implies many electrical-level security concerns. For example, in valid FPGA designs, users can build specialized circuits such as ROs (commonly used as fast clock sources) and TDCs (commonly used to measure the time between two signals in the nanosecond range) leveraging LUTs and FFs. However, as these logic elements are closely coupled with the PDN voltage, potentially malicious parties can misuse these valid structures to exploit electrical-level effects and compromise security. Two examples of such malicious circuits are power-wasting circuits, used for fault injection attacks, and voltage-drop sensors, used for power analysis attacks.

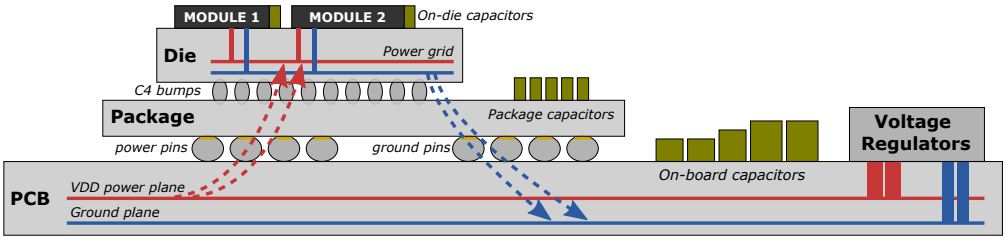


Figure 2.4: Power delivery network coupling across the board, package, and the FPGA die.

In this thesis, we focus on electrical-level attacks on remote FPGAs, in particular remote power side-channel attacks, and evaluate how these low-level FPGA structures can enhance or compromise security.

2.3 Electrical-Level Attacks on FPGA Systems

In FPGA designs, all hardware modules share the FPGA die and the common PDN illustrated in Fig. 2.4. On the printed circuit board (PCB) level, the PDN starts with the primary voltage regulator. The power is then distributed through several levels of voltage regulators, if needed, and the power and ground planes. Inside the FPGA, a PDN resembling a dense mesh supplies power to all FPGA logic and routing resources. On all levels—board, package, chip—the PDN contains resistive, capacitive, and inductive components, some of which are intended and some parasitic, which create a medium for voltage fluctuations in one hardware module to propagate to another or the external power pins.

The electrical coupling of FPGA logic opens the door to various security vulnerabilities, which can be exploited both by attackers with physical access to the system (physical attacks) and attackers with remote access to the system (remote attacks). Fig. 2.5 illustrates the threat model for remotely accessible FPGA systems. For FPGA devices deployed in the field, an attacker has physical access to the device and the external PDN, allowing them to perform physical attacks. For multitenant FPGAs in the cloud, an attacker has remote access to a part of the device and the internal PDN through the low-level FPGA logic, allowing them to perform remote attacks.

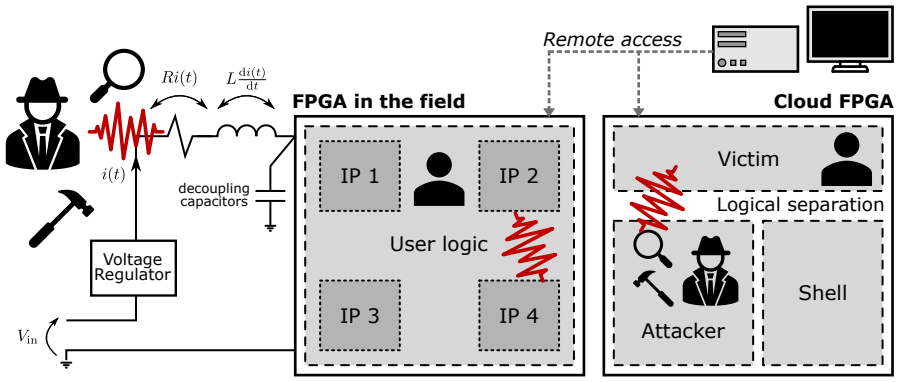


Figure 2.5: Threat model for remote FPGAs. For FPGAs deployed in the field, an attacker has physical access to the device and can perform physical attacks such as power analysis, fault injection, or tampering. For multitenant FPGAs in the cloud, an attacker has remote access and can misuse the FPGA logic to perform remote power analysis and fault-injection attacks.

2.3.1 Physical Attacks

FPGAs, like other integrated circuits deployed in the field, are vulnerable to many physical attacks. These attacks can be classified as *passive* or *active*, depending on the influence on the target. In passive attacks, the device remains working within its specifications, while the attacker observes specific physical properties of the device (power consumption, electromagnetic (EM) emanation, execution time). Side-channel attacks are the most commonly used passive attacks. They involve recording the device's power consumption, electromagnetic emissions, or other physical characteristics under normal operating conditions. Then, by analyzing the recorded signals, attackers can mount statistical attacks to extract sensitive information, such as cryptographic keys or CPU instructions and code.

In active attacks, attackers manipulate the device and operate it outside its specifications to extract secret information. The most commonly used active attacks are fault attacks. These attacks intentionally introduce faults into the device's operation to manipulate its behavior. For example, an attacker may introduce a fault in a specific location in memory to modify a critical value, or lower the power supply voltage to cause glitches within the device and bypass security measures such as key validation. Attackers can even combine passive and active attacks with more invasive methods that damage the device. For example, they can remove

capacitors to increase the power side-channel leakage and mount the attacks more easily, or use power analysis to determine the best moment to inject faults for fault-injection attacks.

2.3.2 Remote Attacks

Attackers with physical access control many system variables, allowing them to mount various attacks. Attackers with remote access are restricted to the pre-defined interfaces of the device and cannot connect measuring equipment to the target. However, unlike physical attacks, which mostly aim to compromise the confidentiality of the devices, remote attacks aim to impact the system's availability, integrity, and confidentiality. Denial of service attacks on the system's availability aim to incapacitate the device completely. Attacks on integrity, notably fault injection attacks, aim to introduce faults into the victim device and compromise the integrity of the results. Finally, attacks on confidentiality, including fault injection attacks and power analysis attacks, aim at extracting secret information that is otherwise not available.

The low-level programmability of FPGAs allows attackers in multitenant scenarios to exploit low-level electrical phenomena to build primitives such as sensors or power wasters, opening the door to various security vulnerabilities previously considered to require physical access [32]. Two main categories of remote electrical-level exploits exist: SCA attacks and fault-injection attacks. Similar to physical side-channel attacks, remote side-channel analysis is based on observing unintended information leakage from a victim design. The malicious party can use a sensor to measure power variations on the chip (power side channel) or to deduce the value carried by a neighboring wire through the EM coupling effects (crosstalk side channel). A similar class of attacks are covert communication attacks; they use the same mechanisms, but require both a sender to send a message and a receiver with a sensor to read it. For remote fault-injection attacks, the adversary leverages power viruses to affect the PDN shared among tenants. Done aggressively enough, this can cause a reset of the board, i.e., a denial-of-service (DoS) attack. If done more precisely, this can cause a computational fault (i.e., a fault attack). These findings temporarily put on hold the FPGA multitenancy in the cloud and pushed many researchers to investigate new attack surfaces, threat models, and

countermeasures [32, 33].

Power Analysis Attacks

The sensor is the essential component for a successful power side-channel exploit. For remote attacks, ROs, whose frequency of oscillation varies with voltage, are used for power SCAs [13] and for crosstalk SCAs [43–45]. They have also been leveraged for covert communication, where the sender is a CPU, GPU, or FPGA, and the receiver is an FPGA sharing the same power supply unit in a datacenter setting [46]. For a faster reaction time than ROs, delay-line sensors, similar to the one proposed by Zick et al. [11], are employed to sense fast voltage fluctuations [12, 47]. Delay-line sensors have even been used to demonstrate power SCAs on Amazon AWS F1 instances [14] and to recover the inputs to a neural network deployed on the same instances [16]. They have also been used to mount attacks against other integrated circuits on the same board [47] and against a CPU sharing the same system-on-chip [48].

Fault-Injection Attacks

Combinational ROs are not only used for sensing on-chip delay changes, but also as power viruses, which, when used in a large enough grid and with specific activation patterns, have been shown to cause board reset [49] and to inject faults [50–52]. Therefore, CSPs such as Amazon [3] have disallowed combinational loops in designs deployed on their FPGAs. However, recent work has shown that other primitives, without combinational loops, can be used for the same purposes. Examples of these primitives include sequential ROs, shift registers, dual-port RAM instances, glitch generators, and even benign-looking circuits such as AES rounds [53, 54].

3 Background

In this chapter, we provide the background required for the remainder of the thesis. We first describe the high-level architecture of FPGAs. Then, we present the most common sensors used to measure on-chip voltage in FPGAs. Further, we introduce power analysis attacks and metrics used for evaluating the attack success, and finally, we define the t-test metric for evaluating the first-order leakage.

3.1 FPGA Architecture

To allow reprogrammability, FPGAs are structured as 2-D arrays of logic resources, connected through vertical and horizontal routing channels, as shown in Fig. 3.1. The reprogrammable logic resources, organized in configurable logic blocks (CLBs), represent the building blocks of the FPGA. CLBs usually consist of multiple *slices*, each containing a number of LUTs and FFs. The LUTs can implement arbitrary combinational logic functions of up to six inputs, while the FFs implement sequential logic functions [55, 56]. In some cases, the FPGA slices contain additional resources, such as fast adders in AMD FPGAs. Depending on the device family, the adders, i.e. CARRY primitives, come in four- or eight-bit variants and can have ripple-carry or carry lookahead implementations [56]. With the growing workload complexity, FPGA architecture is becoming more heterogeneous. In addition to CLBs, modern FPGAs

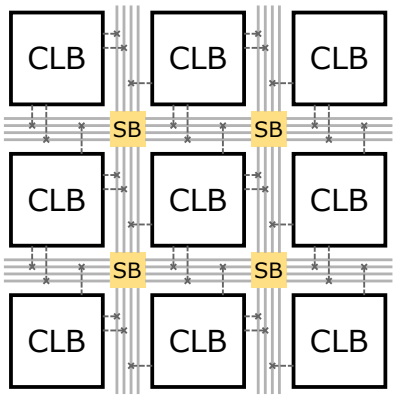


Figure 3.1: FPGA architecture consisting of a matrix of configurable logic blocks (CLBs) and routing resources. The vertical and horizontal routing resources are connected with cross-bars called switch boxes (SB).

contain more specialized blocks such as digital signal processing (DSP) units for fixed point addition and multiplication, SRAM memory in the form of block RAM (BRAM) memory units, and most recently, even specialized AI engines for machine learning acceleration [56, 57]. Finally, FPGAs offer mixed-mode clock manager (MMCM) modules, allowing users to derive several clocks from the base input clock.

Each CLB is connected to the routing resources using a local crossbar, while vertical and horizontal routing lanes are connected through switch boxes (SBs). In special cases, CLBs are connected through dedicated direct routes. For example, to support chains of fast adders required for large additions, AMD FPGAs contain direct routing paths between carry input and output bits of adders in subsequent vertical CLBs. When designing an RTL design, FPGA compilation tools map the digital circuit to a combination of LUTs, FFs, and other resources while ensuring the routing between them is correctly configured, and timing and area constraints are satisfied. After compilation, the tools generate a *bitstream*, used to program the newly compiled RTL design onto the FPGA.

3.2 FPGA-Based On-Chip Voltage Sensors

In recent years, it was shown that it is possible to measure internal voltage fluctuations on the PDN of an FPGA by using sensors implemented using the reconfigurable FPGA fabric.

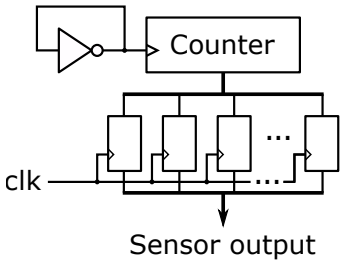


Figure 3.2: RO-based voltage sensor.

After their introduction, these sensors found a wide variety of applications: detection of voltage undershoots caused by malicious attackers [11], measurement of power side-channel leakage to mount remote power analysis attacks [12–14], protection against power analysis attacks [58, 59], and detection of voltage drops caused by the insertion of shunt resistors in tampering [60].

The two most commonly used sensors to measure voltage fluctuations on FPGA are TDC and RO sensors. Any change in the PDN voltage behavior directly influences the delays of the CMOS logic gates, which these sensors exploit to estimate the power consumption of the neighboring logic. RO-based sensors, shown in Fig. 3.2, count the number of RO oscillations in a fixed time period. In contrast, the TDC-based sensors, shown in Fig 3.3, count the number of buffers in a delay line through which a clock signal has propagated in one clock period. Both sensors produce an output in the function of their circuit delay, which is approximately inversely proportional to the core voltage fluctuations.

RO-based sensors use combinational loops and LUTs to measure the delay. As such, RO-based sensors need considerably longer to produce a value than TDC sensors. In many scenarios, such as on the AWS EC2 F1 instances studied in this thesis, combinational loops are not supported [61]. This limitation can be bypassed by designing RO-based sensors containing registers, as shown by Giechaskiel et al. [62]. Despite this workaround, the sampling frequency of RO-based sensors is much lower than that of the TDC sensors, making the former unsuitable for recording nanosecond-scale voltage fluctuations needed to perform power analysis attacks.

However, RO-based sensors have a smaller footprint and need not be calibrated, unlike TDCs.

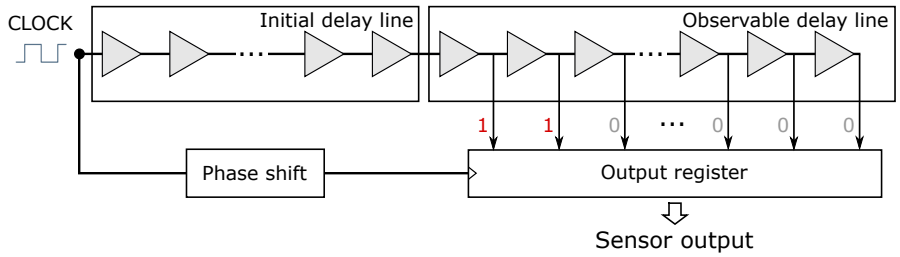


Figure 3.3: Time-to-digital converter (TDC) sensor architecture.

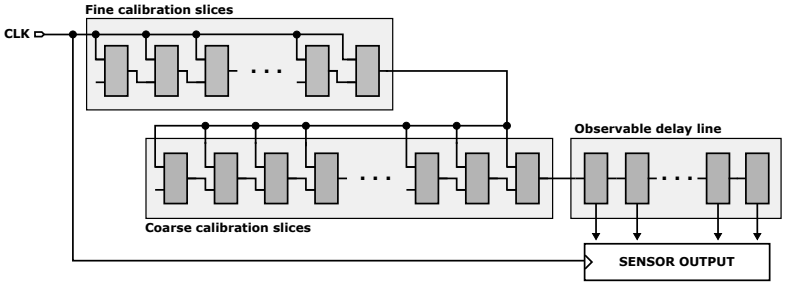


Figure 3.4: TDC sensor architecture with a tunable phase shift between the clock that enters the observable delay line and the clock that samples the output (i.e., takes a snapshot of the observable delay line). The exact number of slices in our implementation is in Table 6.2.

Good use cases for RO-based sensors are FPGA undervolting-based attacks [63, 64] and covert communication [62]. For power side-channel leakage analysis, given the importance of a high sampling rate, TDCs are the preferred solution [12, 14, 15]; they can record voltage fluctuations with sampling periods in the nanosecond range [11]. Therefore, in our work, we use TDC voltage-drop sensors, described in the following paragraphs.

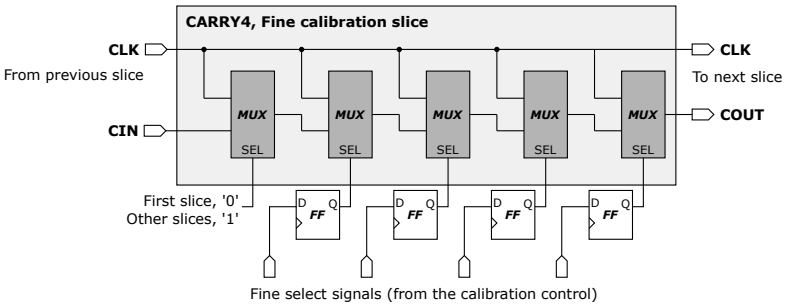
The most straightforward TDC architecture consists of a line of buffers driven by a high-frequency *input clock* signal, as seen in Fig. 3.3. The propagation depth of the clock signal is proportional to the overall delay of the buffer line. The first part of the delay line, the *initial delay*, uses elements with a higher delay (such as LUTs or latches) to introduce a more significant initial delay to the clock signal with a smaller area overhead. The second part of the delay line, the *observable delay*, consists of buffers whose outputs are *tapped*, i.e., connected to registers that record the state of the line. The sensor registers are clocked with the *sampling* clock—usually a shifted input clock—and therefore record how far the clock propagated through the delay line when the sampling clock reaches the registers. The observable delay

line is usually implemented using carry-chain primitives that provide a very fine resolution per bit.

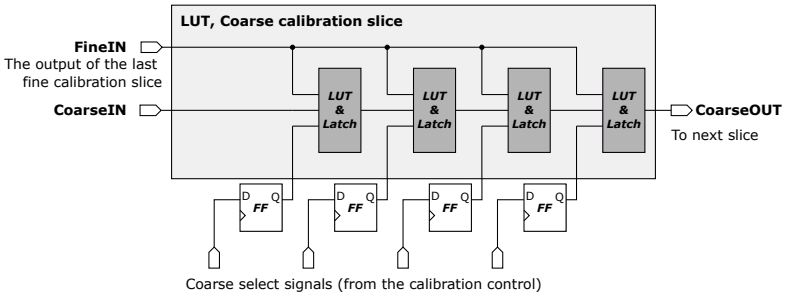
The phase shift between the input and the sampling clock determines the sensor calibration. For a correctly calibrated sensor, the rising edge of the input clock signal is positioned within the observable delay line when its state is saved to the registers. The correct calibration can be achieved by changing the length of the initial delay (at compile time or at run time), tuning the phase of the sampling clock, or both.

Since the calibration is a lengthy process of trial and error, it is usually automated and performed at run time. One possibility is to use a tunable digital clock manager (DCM) to control the phase delay between the input and the sample clock. Another possibility, introduced by Gnad et al. [65] and used throughout this thesis, is implementing a tunable initial delay line.

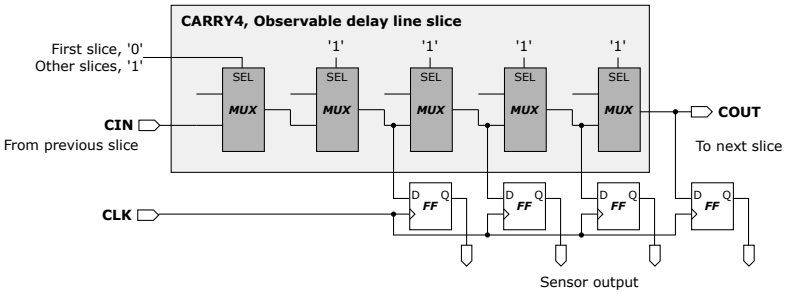
Fig. 3.4 shows the high-level architecture of the TDC sensor with a tunable initial delay line. It consists of fine calibration slices, coarse calibration slices, and the observable delay line. The input and the sample clocks are the same. Therefore, to control the phase shift between the input and the sample clock, *fine* and *coarse* calibration slices are inserted on the input clock path. The fine calibration slices are implemented using CARRY logic, as shown in Fig. 3.5a, while the calibration inputs control the number of carry chain multiplexers on the clock path. The fine calibration slices are then connected to the coarse calibration slices (Fig. 3.5b), where the calibration inputs control the number of coarser delay elements on the clock path. In the TDC design used throughout this thesis, coarse delay elements are implemented as LUTs followed by latches, to achieve coarser delay increments. The third and last stage is the *observable delay line* (Fig. 3.5c), implemented using CARRY logic connected to slice registers. During the calibration process, the length of the initial delay line is incrementally increased until the rising edge of the input clock signal is positioned within the observable delay line when its state is saved to the registers.



(a) Fine calibration slice.



(b) Coarse calibration slice.



(c) Observable delay line slice.

Figure 3.5: The implementation of each slice in the TDC sensor in Fig. 3.4, including the calibration and sensor output registers. For space reasons, CARRY4 chain is shown horizontally; in the FPGA design layout, it spans vertically.

3.3 Power Analysis Attacks

Cryptographic algorithm designers are primarily concerned with the mathematical security of the algorithms, usually without regard for their future implementation. However, if the engineers responsible for implementing the algorithm in hardware or software lack knowledge of cryptography, the implementations of mathematically secure cryptographic ciphers can unintentionally leak information through so-called *side channels*. In such cases, adversaries mount side-channel attacks that try to recover otherwise secure information through the side channels. These attacks are usually *passive* and *non-invasive*, operating the device within its specifications without any permanent alterations. Examples include timing-based attacks (targeting the execution time of secret-dependent operations), cache side-channel attacks (leveraging the cash access latency to determine the data recently accessed by a potential victim), and power analysis attacks (targeting secret-dependent power consumption).

This thesis primarily focuses on power side-channel attacks. First introduced by Kocher et al. [8], power side-channel attacks exploit the fact that the data-dependent switching activity of the logic cells directly impacts the power consumption of an integrated circuit. Changes in the circuit inputs lead to data propagation through the combinational logic, causing glitches and transitions in the logic gate outputs that consume power. Consequently, the power consumption is correlated to the changes in the circuit inputs. In the case of a cryptographic device, an attacker can infer the secret key by exploiting this correlation. The attacker can use simple, visual analysis of the power measurements, or more complex statistical analysis such as the differential power analysis (DPA) attack [8], or the CPA attack [9]. Statistical attacks represent a set of very powerful attacks, and can extract the key of unprotected cryptographic implementations using a small number of measurements [10]. Although extremely powerful, power side-channel attacks have always been considered a danger only for embedded devices since the adversary was required to have physical access to the device to collect the power measurements needed to mount the attack. In their simplest form, power side-channel attacks are noninvasive attacks, as performing a successful attack does not require physical changes

to the victim device.

In power side-channel attacks, a power trace $P[i][:]$ is the measured power consumption during one execution, i.e., execution i, of the victim circuit. A power trace consists of a finite set of points, each one being a measured *sample* of the power consumption $P[i][j]$ at time j. The number of samples per trace depends on the sampling frequency of the measurement circuit and the operating frequency of the victim circuit. The number of traces for an attack depends on various factors, such as the victim circuit, the quality of the measurement setup, and the threat model.

The measured power consumption traces contain the exploitable power consumption, made by the attacked circuit, and the noise originating both from the power consumption of the rest of the circuit and other electrical effects. The success of the attack largely depends on the SNR of the collected traces, i.e., the ratio of the variance of the exploitable signal and the noise. Therefore, to increase the chances of a successful power analysis attack, the attacker needs to record power consumption traces with as little noise as possible [10]. Depending on the threat scenario, attackers can resort to different noninvasive methods. One of the standard methods is averaging, where for each different input of the cryptographic circuit the attacker collects a number of traces and performs a sample-wise average. Another noise reduction technique is filtering, where the traces are passed through a low-pass filter to remove electronic noise at high frequencies.

In case of protected designs, in addition to first-order attacks, attackers resort to more advanced methods such as higher-order attacks. These include preprocessing the traces to obtain mean-free squared traces [66], or attacking multiple samples at the same time [10, 67]. Higher-order attacks are out of scope for this work, as our threat model targets only first-order attacks on both protected and unprotected designs.

3.3.1 Correlation Power Analysis Attack

The CPA attack, shown in Fig. 3.6, is one of the most widely used power analysis attacks today. It assumes the adversary has physical access to the device and can record the power while it is encrypting data. Moreover, it assumes the attacker has access to the encrypted data being sent over a public channel. The attack relies on a divide-and-conquer approach to extract the secret key: instead of attacking the whole 128-bit key (2^{128} space), it exploits the fact that AES, specifically the S-box, operates on key bytes (sub-keys), which allows attacking the bytes independently and reduces the key space to 16×2^8.

To start an attack, the attacker records N power traces and their corresponding ciphertexts, as seen in step (1) in Fig. 3.6. For each key byte, the attacker considers all possible 256 values (guesses). Each guess is then used to calculate the estimated power consumption for all the observed ciphertexts (step (2) in Fig. 3.6). Then, for each sample in the recorded power traces, the estimated power is correlated with the measured power using the Pearson coefficient (step (3) in Fig. 3.6). The Pearson coefficient equals to:

$$r[k][j] = \frac{cov(P[:][j], H[:][k])}{\sigma_{P[:][j]}\sigma_{H[:][k]}} = \frac{\sum_{i=0}^{N-1}\left[\left(H[i][k]-\overline{H[k]}\right)\left(P[i][j]-\overline{P[j]}\right)\right]}{\sqrt{\sum_{i=0}^{N-1}\left(H[i][k]-\overline{H[k]}\right)^2\sum_{i=0}^{N-1}\left(P[i][j]-\overline{P[j]}\right)^2}}, \tag{3.1}$$

where N is the number of power traces $P[i][:]$, where each of the traces has K samples. The matrix notation $P[i][j]$ refers to the power sample j in trace i, where $0 \le i \le N-1, 0 \le j < K-1$. If there are I values a sub-key can take, $H[i][k]$ represents the power estimate in trace i for the sub-key guess k, where $0 \le i \le N-1, 0 \le k < I-1$. With this information, the Pearson coefficient shows how well the modeled power at sample j matches with the measured power for each sub-key guess k. The correlation factor is calculated for all the samples j and all the possible values for the sub-key k, resulting in a $K \times I$ matrix of correlation factors $r[k][j]$. For each sub-key guess $\boldsymbol{k}$, the value j for which $|r[\boldsymbol{k}][j]|, 0 \le j \le K-1$ is the highest, is the power trace sample which correlates best with the model using the guess $\boldsymbol{k}$. The sub-key guess that has the highest correlation $|r[k][j]|, 0 \le j \le K-1, 0 \le j < K-1$ is assumed to be the correct

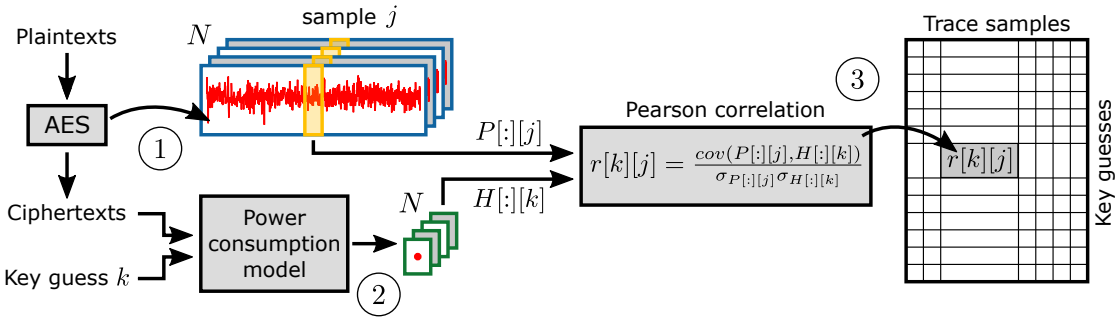


Figure 3.6: Correlation power analysis attack

sub-key. After the calculation is repeated for all the sub-keys, the attacker can reconstruct the key.

To get a power consumption estimation, attackers use various different power models. Since in most cases, the power consumption of synchronous CMOS devices depends on the toggling bits in the registers, the most commonly used power model is the Hamming distance (HD) of a register output. The advantage of the HD model is that it is implementation-agnostic. More advanced power analysis attacks require the use of more complex models, which rely on more implementation details of the attacked device [68].

3.3.2 Attack Success Metrics

Most power side-channel attacks use a divide-and-conquer strategy [8,9], and they partition the full cipher key into small parts that are easy to attack. For example, instead of directly recovering the full 16-byte key of AES-128, the CPA attack in Section 3.3 divides it into 16 parts and tries to recover every one of the bytes individually. For every 8-bit key part, the attack will produce *scores* for every key candidate. In CPA, the score is the correlation of the key candidate k. To assess the success of the attack, attackers (and evaluators) deploy different ranking metrics based on the scores. These metrics are similar for both attackers and evaluators: the only difference is the attackers do not know the key, while the evaluators do.

The **key rank** metric, performed on each byte, is obtained when the key candidates are sorted from best to worst based on the score. From an attacker's point of view, the key guess with the

key rank of 1 is the best key candidate, and is assumed to be the key byte. In the case of an evaluator, the key rank of the actual key can be used to measure the success of an attack. If the rank of the key byte is 1, it was successfully broken. Otherwise, the rank can potentially indicate how far was the attack from successfully breaking the key byte. This process of scoring and ranking is typically repeated for all bytes of the key; that is, in an example of an AES-128, it will be performed 16 times (partitioning the full key to 16 key byte-sized parts).

As previously stated, power side-channel attacks are usually performed in a divide-and-conquer fashion, where each part of the key is attacked independently from all others. The key rank metric described thus far focuses on the security of these key parts and does not generalize directly to the security of the full key. For example, in the case of AES-128, if all 16 correct key bytes are ranked first among their candidate set, the full key is trivially recovered. However, when this is not the case, the adversary has to verify different full keys through trial and error using a process called key enumeration. Similarly, the evaluator must also quantify the remaining effort for full key recovery using rank estimation. The goal of the enumeration and estimation processes is to convert the security metrics for the 16 8-bit parts of the key to a security metric for the full key.

Since key enumeration is more computationally intensive, and researchers usually know the key of the cryptographic device under attack, this thesis performs known-key power side-channel assessment with the **key rank estimation** metric. Notably, rank estimation can also work in the unknown key setting, since heuristics have been proposed to approximate the rank of an unknown secret key without performing key enumeration [69]. In the known-key case, the key rank estimation metric uses the correlation computed with the CPA—or some other scoring metric—to quickly estimate the remaining brute force effort to recover the full key, without necessarily listing all possible keys [70]. For example, if an attacker has no side-channel information, then the key rank equals to the entire key space, i.e., 2^{128} in the case of AES-128. Alternatively, when the entire key is broken, the key rank drops to zero. While there are several ways of computing the key rank, in this thesis, we use the histogram-convolution-based algorithm of Glowacz et al. [71], where the key rank is upper and lower

bounded.

3.4 First-Order Side-Channel Leakage Evaluation

Welch's t-test, most often used to determine if two datasets statistically differ from each other, is commonly used for the evaluation of the first-order side-channel leakage in the context of the test vector leakage assessment (TVLA) methodology [72, 73]. One of the commonly used versions of the t-test, the *nonspecific* t-test, does not target a specific intermediate value in the cryptographic device. In this case, to evaluate the observable leakage, two sets of power traces need to be collected: one containing traces obtained encrypting random plaintexts (Q_R), and the other containing traces obtained encrypting a constant, fixed, plaintext (Q_F). Moreover, to avoid biased initial conditions, the encryptions of fixed and random plaintexts should be interleaved in a nondeterministic or random fashion. The t-test statistic, following the Student's t-distribution, can be then computed as

$$t = \frac{\mu_F - \mu_R}{\sqrt{\frac{s_F^2}{n_F} + \frac{s_R^2}{n_R}}}, \tag{3.2}$$

The t-test is widely utilized due to its simplicity and ability to be used without the knowledge of the underlying cryptographic implementation. However, the t-test returns a binary result, classifying a trace sample as *leaky*, i.e., containing first-order leakage, and *not leaky*, i.e., not containing first-order leakage. Therefore, as a leakage evaluation metric, the t-test cannot quantify the leakage in the trace samples and cannot be used to compare two cryptographic implementations. The t-test does not provide information on whether a (leaky) sample can be successfully attacked. Therefore, the t-test is used to evaluate the *observable* leakage, but not the *exploitable* leakage. In the leakage detection framework of the t-test false positives correspond to the detection of leaky samples that contain no exploitable leakage, but for which exists *some* data dependency in the power consumption, for instance, corresponding to plaintext variations or intermediate values that are difficult to attack.

Part I

Evaluating the Security of Deployed Cyber-Physical Devices

4 Power Side-Channel Leakage Evaluation on Remote FPGAs

The pervasive diffusion of embedded and cyber-physical systems in automotive, industry 4.0, healthcare, power grid, and all other aspects of our lives imposes new challenges for system developers. The criticality of the applications and the sensitivity of data require the use of appropriate security primitives everywhere. Deploying these devices in hostile environments, often accessible to adversaries, calls for side-channel attack countermeasures and the ability to detect unwanted tampering. Finally, the service life of these devices—typically longer than that of consumer electronic devices—requires the ability to deal with device performance degradation, affecting its resilience to side-channel attacks.

To detect tampering, some devices have shields and sensors that detect changes in light, temperature, or even attempts to remove mechanical protections. However, these methods are limited and cannot detect other tampering attempts or changes due to device malfunctioning or maintenance [74]. Therefore, to ensure that the security of their devices is not compromised, users need to be able to monitor the device during its entire life span. For instance, one can monitor the statistical properties of random number generators [75] or samplers in lattice-based constructions [76]. If they exhibit irregular behavior, it can indicate malfunctioning or a tampering attempt. Other examples include monitoring the PDN for signs of voltage

This chapter is based on the work of a paper published at the 2020 International Symposium On Field-Programmable Gate Arrays [18].

drops caused by the insertion of shunt resistors [60, 77], or detecting impedance changes caused by other measuring equipment such as EM probes [78]. However, the latter type of monitoring focuses on detecting changes in the device's electrical properties and does not provide information on how these changes impact (power side-channel) security.

Despite these initial and successful attempts, developing on-chip sensors for real-time monitoring of security primitives is still a largely unexplored area of research. A security problem that would benefit from using real-time on-device sensing is that of power side-channel attacks, which have proven very effective on CPUs and FPGAs [8, 13]. Moreover, successful tampering attempts on protected and unprotected devices have shown that attackers can increase the information leakage to deploy these power side-channel attacks more easily. Tampering with the PDN, e.g., removing or adding components to the PCB, is a common method to reduce the attack effort [12, 17, 79, 80]. For example, Schellenberg et al. demonstrated that removing decoupling capacitors, which ensure the filtering of local voltage variations at the board level, results in 100× fewer measurements needed to break the secret key [47].

Our work targets embedded/cyber-physical systems often accessible to adversaries, which may be subject to aging, malfunctioning, or tampering with the intent to make the device leak a higher-than-desired amount of information via the power side channel. A large body of research is devoted to countermeasures against power side-channel attacks. However, their effectiveness can only be verified before the device deployment and almost never reevaluated, making it impossible to detect an increase in power side-channel leakage due to malfunctioning or tampering.

In this chapter, we describe and design a standalone test that performs on-chip voltage measurements and evaluates the susceptibility to power analysis attacks of an FPGA implementation of a cryptographic core. We assess the test accuracy using the AES-128 algorithm as a case study and compare the results with those obtained using standard in-lab testing procedures and equipment. The *de facto* standard method for estimating power side-channel leakage is Welch's t-test [72]; it requires measuring the power-supply voltage during many

encryptions and computing the t-test statistic over the acquired data. We employ sensors directly implemented in the FPGA fabric to measure the FPGA power-supply voltage as frequently as possible. Our test hardware computes the first-order t-test statistic and is calibrated on the target cryptographic core. Although verified using AES, our proposed methodology is general and can be applied to any cryptographic core. Moreover, our built-in test can be triggered remotely, periodically, or on power-up, thus continuously monitoring the device's susceptibility to the first-order correlation and differential power analysis attacks during its entire service life.

In the remainder of this chapter, we first introduce the threat model. Then, in Section 4.2, we outline common tampering techniques. In Section 4.3, we introduce the t-test leakage estimation methodology and describe how we adapt it for computing the t-test metric in hardware. Section 4.4 describes the architecture and FPGA implementation of our system. Section 4.5 presents and discusses the experimental results, while Section 4.6 concludes the chapter.

4.1 Threat Model

Our threat model targets FPGA-based embedded and cyber-physical devices accessible to attackers, and vulnerable to PDN tampering attacks that make the devices leak higher-than-desired information through the power side channel. This adversary model includes devices vulnerable to first-order power side-channel attacks, both protected and unprotected, and connected or unconnected to the network. The security of the devices is typically assessed only before deployment in the field, making it impossible to detect an increase in power side-channel leakage during the device service life. Our threat model is focused on cyber-physical systems where the security primitives are implemented on FPGA. In such a system, voltage traces can be recorded using FPGA-based sensors such as TDCs, while FPGA DSP blocks can be used for computing the leakage estimates.

4.2 Tampering with the PDN

As the power delivery network on the PCB is designed to minimize and filter the voltage fluctuations on the chip's power pins, a noninvasive attacker obtains filtered traces, requiring a significant number of measurements for a successful attack. Consequently, to improve the quality of the acquired traces, attackers resort to more invasive methods. On the one hand, attackers can insert shunt resistors and/or amplifiers to easily record the fluctuating current consumed by the cryptographic device [10, 17]. On the other hand, the attackers can remove decoupling capacitors, ensuring that local voltage variations are not filtered at the board level. Khan et al. showed that removing capacitors can lead to a significant increase in the detectable leakage [79], while Mazur et al. showed that removing capacitors leads to a significant reduction in the number of traces needed to extract the key [80]. Schellenberg et al. demonstrated that removing decoupling capacitors results in a 100× fewer measurements needed to break the secret key [12].

4.3 Online Computation of the *t*-test Statistic

In cryptography, the nonspecific t-test is commonly used to evaluate first-order side-channel leakage. To compute it, two sets of power traces need to be collected: one while a constant plaintext is encrypted (Q_F) and the other while randomly chosen plaintexts are encrypted (Q_R). Additionally, the decision whether to encrypt the fixed or random plaintext must be made in a nondeterministic or a randomly-interleaved fashion, to avoid predictable initial conditions. The t-test statistic can then computed as

$$t = \frac{\mu_F - \mu_R}{\sqrt{\frac{s_F^2}{n_F} + \frac{s_R^2}{n_R}}}, \tag{4.1}$$

where μ_F (resp. μ_R) is the sample mean and s_F^2 (resp. s_R^2) the sample variance of the set Q_F (resp. Q_R), and n_F (resp. n_R) is the cardinality of the set Q_F (resp. Q_R). If $|t|$ exceeds the threshold of 4.5, the test has detected a leakage with a confidence of at least 0.99999 [73]. If

the power traces contain multiple samples, the test needs to be repeated for every sample.

Therefore, calculating the t-test requires efficient and correct computation of the mean and variance for each power trace sample. With the cardinality of the power trace sets in the tens of thousands, a straightforward computation of these statistic parameters would result in excessive memory overhead, significant latency, and low throughput. Hence, the statistic must be computed online. However, algorithms for incremental computation are typically limited to recomputing the mean and variance with every new trace [81, 82]. We demonstrate that this frequent computation is inefficient and not needed to achieve correct t-test results. Instead, we propose a method that performs minimal computation after each new trace acquisition and recomputes the statistic parameters only after a batch of k power consumption traces have been processed. The parameter k is defined by the user, according to the needs of the target application.

For a set of traces $\mathbf{X}$ of cardinality N, where each trace has M samples, the mean and the variance of a trace sample x over all N traces are defined as follows:

$$\mu = \mu_N = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{4.2}$$

$$s_N^2 = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu_N)^2, \tag{4.3}$$

where i is the index of the trace in the set $\mathbf{X}$. The above equations are called *standard two-pass algorithm* [83], as they require traversing the data twice: once to compute the mean and a second time to compute the variance. From a hardware acceleration point of view, the computation is impractical, as a large cardinality of $\mathbf{X}$ can lead to an unacceptable amount of memory and computational overhead to store and access all the data. A possible solution is to use the *one-pass algorithm*, where (4.3) can be transformed to:

$$s_N^2 = \sum_{i=1}^{N} x_i^2 - \frac{1}{N}\left(\sum_{i=1}^{N} x_i\right)^2. \tag{4.4}$$

However, this form is numerically unstable when implemented in floating-point arithmetic. In

fact, it is susceptible to large cancellation errors or even a negative estimated variance. The instability can be remedied by using an *online* method [83], for instance Welford's algorithm [82], in which the statistics are recomputed with every new sample that arrives:

$$\mu_{n+1} = \mu_n + \frac{1}{n+1}(x_{n+1} - \mu_n), \tag{4.5}$$

$$s^2_{n+1} = \frac{1}{n+1}\left(s^2_n + \left(x_{n+1} - \mu_{n+1}\right)\left(x_{n+1} - \mu_n\right)\right). \tag{4.6}$$

The disadvantage of (4.6) is that the new mean needs to be ready before the update of the variance can take place. Schneider et al. address this by first introducing the term central sum

$$CS_{d,n} = \sum_{i=1}^{n} (x_i - \mu_n)^d \tag{4.7}$$

and then deriving the formula for updating it incrementally [73]:

$$CS_{d,n+1} = CS_{d,n} + \sum_{m=1}^{d-2} \binom{d}{m} CS_{d-m,n} \left(\frac{\mu_n - x_{n+1}}{n}\right)^m + \left(\frac{n-1}{n}\left(\mu_n - x_{n+1}\right)\right)^d \left[1 - \left(\frac{-1}{n-1}\right)^{d-1}\right]. \tag{4.8}$$

Finally, the expression for the variance—as it is also used by Sonar et al. [81]—is only one scaling factor away:

$$s^2_{n+1} = \frac{1}{n+1} CS_{d,n+1} = CS_{2,n} + \frac{n}{n+1}(x_{n+1} - \mu_n)^2. \tag{4.9}$$

To avoid updating the mean and variance with every new power trace, we introduce incremental updates of size k. We introduce the term *partial central sum*: $CS_d(n,k)$. It corresponds to the central sum computed over k samples in the range $(n+1, ..., n+k)$:

$$CS_d(n,k) = \sum_{i=n+1}^{n+k} (x_i - \mu_n)^d = \sum_{i=1}^{k} (x_{n+i} - \mu_n)^d. \tag{4.10}$$

The mean of $n+k$ elements can then be represented as:

$$\mu_{n+k} = \frac{1}{n+k}\left(\sum_{i=1}^{n} x_i + \sum_{i=n}^{n+k} x_i\right) = \frac{1}{n+k}\left(n\mu_n + \sum_{i=n}^{n+k} x_i\right) \tag{4.11}$$

The numerator in (4.11) can be further written as:

$$n\mu_n + \sum_{i=n}^{n+k} x_i = (n+k)\mu_n - \sum_{i=1}^{k} \mu_n + \sum_{i=1}^{k} x_{n+i} \tag{4.12}$$

Finally, merging (4.11) and (4.12) results in an incremental expression for the mean:

$$\mu_{n+k} = \mu_n + \frac{1}{n+k} CS_1(n,k). \tag{4.13}$$

Similarly, the variance of $n+k$ elements can be rewritten in terms of the variance of n elements, $CS_1(n,k)$, and $CS_2(n,k)$:

$$s^2_{n+k} = \frac{1}{n+k} CS_{2,n+k}. \tag{4.14}$$

Moreover, $CS_{2,n+k}$ can be rewritten using (4.7), (4.10), and (4.13):

$$\begin{aligned} CS_{2,n+k} &= \sum_{i=1}^{n+k} (x_i - \mu_{n+k})^2 = \\ &= \sum_{i=1}^{n+k} \left(x_i - \mu_n - \frac{CS_1(n,k)}{n+k}\right)^2 = \\ &= \sum_{i=1}^{n+k} \left((x_i - \mu_n)^2 - 2(x_i - \mu_n)\frac{CS_1(n,k)}{n+k} + \frac{CS_1(n,k)^2}{(n+k)^2}\right). \end{aligned} \tag{4.15}$$

The sum in (4.15) can be decomposed in three sums:

$$sum_1 = \sum_{i=1}^{n+k} (x_i - \mu_n)^2, \tag{4.16}$$

$$sum_2 = \sum_{i=1}^{n+k} 2(x_i - \mu_n)\frac{CS_1(n,k)}{n+k}, \tag{4.17}$$

and

$$sum_3 = \sum_{i=1}^{n+k} \frac{CS_1(n,k)^2}{(n+k)^2}. \tag{4.18}$$

The expression in (4.16) can be rewritten as:

$$sum_1 = \sum_{i=1}^{n} (x_i - \mu_n) + \sum_{i=n}^{n+k} (x_i - \mu_n)^2 = CS_{2,n} + CS_2(n,k), \tag{4.19}$$

while the expression in (4.17) can be rewritten as:

$$\begin{aligned} sum_2 &= 2\frac{CS_1(n,k)}{n+k} \sum_{i=1}^{n+k} (x_i - \mu_n) = \\ &= 2\frac{CS_1(n,k)}{n+k} \left(\sum_{i=1}^{n} (x_i - \mu_n) + \sum_{i=n}^{n+k} (x_i - \mu_n) \right) = \\ &= 2\frac{CS_1(n,k)}{n+k} \left((n\mu_n - n\mu_n) + CS_1(n,k) \right) = \\ &= 2\frac{CS_1(n,k)^2}{n+k}. \end{aligned} \tag{4.20}$$

Finally, the expression in (4.18) can be rewritten as:

$$sum_3 = (n+k)\frac{CS_1(n,k)^2}{(n+k)^2} = \frac{CS_1(n,k)^2}{n+k}. \tag{4.21}$$

Combining (4.19), (4.20), and (4.21) in (4.15) results in the incremental equation for $CS_{2,n+k}$:

$$CS_{2,n+k} = CS_{2,n} + CS_2(n,k) - \frac{CS_1(n,k)^2}{n+k}. \tag{4.22}$$

Dividing (4.22) with $n+k$ leads to the final equation for the variance with an incremental step k:

$$\begin{aligned} s^2_{n+k} &= \frac{CS_{2,n}}{n+k} + \frac{CS_2(n,k)}{n+k} - \left(\frac{CS_1(n,k)}{n+k} \right)^2 = \\ &= \frac{n}{n+k}\frac{CS_{2,n}}{n} + \frac{CS_2(n,k)}{n+k} - \left(\frac{CS_1(n,k)}{n+k} \right)^2 = \\ &= \frac{n}{n+k} s^2_n + \frac{CS_2(n,k)}{n+k} - \left(\frac{CS_1(n,k)}{n+k} \right)^2. \end{aligned} \tag{4.23}$$

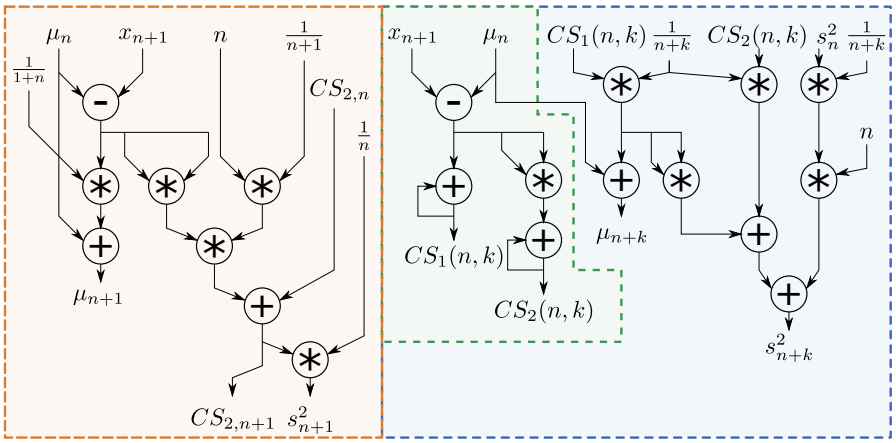


Figure 4.1: Dataflow graph for operations when calculating the mean and variance incrementally with step 1.

To eliminate the second central sum in the first element of the sum in (4.23), we can multiply and divide it by n, resulting in the final equation for the variance of $n+k$ elements in (4.14) written in terms of the variance of n elements, $CS_1(n,k)$, and $CS_2(n,k)$:

$$s^2_{n+k} = \frac{n}{n+k} s^2_n + \frac{1}{n+k} CS_2(n,k) - \left(\frac{1}{n+k} CS_1(n,k)\right)^2. \tag{4.24}$$

The new equations for the mean and variance in (4.13) and (4.24) remove the constraint of recomputing the statistics after every new trace, which is the case in the work by Sonar et al. [81]. Fig. 4.1 highlights the differences in calculating the statistics with each new trace and k new traces. On the left, the dataflow graph of the operations performed in (4.5) and (4.9) shows that computing the mean and variance takes eight operations with every new trace. In contrast, on the right, the dataflow graphs of the operations in (4.13) and (4.24) show that computing the partial central sums takes four operations with every new trace, and additional eight operations after k traces. Starting from a step size of $k = 3$, it takes fewer operations to compute the mean and the variance than with a single incremental step.

To avoid hardware divisions, updating the mean and variance with each new trace requires storing values $\frac{1}{n}$ and $\frac{1}{n+1}$ in local memory; for $1 \leq n \leq N$. The memory overhead amounts to N words, where N can be in the tens of thousands. When the step size is k, the memory

overhead is $\frac{N}{k}$ words, which is negligible when k is sufficiently large (e.g., 1024). Finally, (4.9) requires storing the second central sum—a sum of squares—of all trace samples. A large N can have large numerical values, requiring many integer bits for a fixed-point representation. Conversely, $CS_d(n,k)$ is a sum of squares of size k, requiring fewer integer bits for the hardware representation.

In our approach, the t-statistic of every trace sample is updated with the same frequency as the mean and the variance: every k power consumption traces. To avoid computing the square root and division, similar to Sonar et al. [81], we use the squared t-test statistic, and an equal number N of fixed- and random-plaintext traces:

$$t^2 = \frac{\left(\mu_R - \mu_F\right)^2}{\frac{1}{N}\left(s_R^2 + s_F^2\right)}. \tag{4.25}$$

As the t-test cannot be used to quantify leakage, and leakage is detected when the t value crosses the predefined threshold $|t| > 4.5$, we implement the following comparison:

$$N\left(\mu_R - \mu_F\right)^2 > 4.5^2\left(s_R^2 + s_F^2\right). \tag{4.26}$$

Power trace samples for which this comparison returns logical 1 are considered as *leaky*.

4.4 Built-in Leakage Evaluation System

Our system for online self-evaluation of the first-order power side-channel leakage follows the steps of the function `BuiltInTest`, shown in Algorithm 1. The parameters of the `BuiltInTest` function include E_{MAX}, the desired maximum number of random/fixed encryptions for the test, k, the rate at which the mean, variance, and the t-test statistic are updated, N_S, the number of sensor samples per power trace, the cryptographic test key K, the fixed plaintext PT_F, and the initial chained plaintext PT_0. The higher the value of k, the smaller the number of operations the built-in test needs to perform and, consequently, the shorter the time for

Algorithm 1: `BuiltInTest`. Checks if the first-order power side-channel leakage is present.

Input: E_{MAX}: number of encryptions for which the t-test is performed
Input: k: mean and variance increment size
Input: N_S: number of sensor samples per power trace
Input: K: test key, for use only during built-in testing
Input: PT_{F}: fixed plain-text input
Input: PT_0: initial plain-text input
Variables: $\mathbf{M}_R, \mathbf{M}_F, \mathbf{V}_R, \mathbf{V}_F$: memories for keeping means and variances of the random (R) and fixed (F) traces $\mathbf{pCS1}_R, \mathbf{pCS1}_F, \mathbf{pCS2}_R, \mathbf{pCS2}_F$: memories for keeping the first and second partial central sums of the random (R) and fixed (F) traces
Output: **TTEST**, array with the t-test results of every trace sample for every step k

$PT_R \leftarrow PT_0$
ttest.clock.disable()
$t = 0$
while $t < E_{\mathrm{MAX}}$ **do**
  pCS1.*clear*()
  pCS2.*clear*()
  `/* Phase I */`
  for $e \leftarrow 1$ **to** k **do**
    `/* Repeat for every new trace. */`
    foreach t, $t \in \{R, F\}$ **do**
    CRYPTO.reset()
    $CT_{\mathrm{t}} \leftarrow$ *CRYPTO.encrypt*(PT_t, K)
    $trace \leftarrow$ *load*(*sensor*(), N_S)
    ttest.clock.enable()
    foreach *trace sample* i, $i \in [0..N_S - 1]$ **do**
      $x_t \leftarrow (trace(i) - \mathbf{M}_t(i)) + \mathbf{pCS1}_t(i)$
      $y_t \leftarrow (trace(i) - \mathbf{M}_t(i))^2 + \mathbf{pCS2}_t(i)$
      $\mathbf{pCS1}_t$.*store*(x_t, i)
      $\mathbf{pCS2}_t$.*store*(y_t, i)
    ttest.clock.disable()
    $PT_R \leftarrow CT_{\mathrm{R}}$
  `/* Phase II */`
  `/* Repeat after` k `traces. */`
  ttest.clock.enable()
  foreach *trace sample* i, $i \in [0..N_S - 1]$ **do**
    $m_F \leftarrow mean(\mathbf{M}_F(i), \mathbf{pCS1}_F(i), t)$
    $m_R \leftarrow mean(\mathbf{M}_R(i), \mathbf{pCS1}_R(i), t)$
    $\mathbf{M}_F$.*store*(m_F, i)
    $\mathbf{M}_R$.*store*(m_R, i)
    $s_F^2 \leftarrow var(\mathbf{V}_F(i), \mathbf{pCS1}_F(i), \mathbf{pCS2}_F(i), t)$
    $s_R^2 \leftarrow var(\mathbf{V}_R(i), \mathbf{pCS1}_R(i), \mathbf{pCS2}_R(i), t)$
    $\mathbf{V}_F$.*store*(s_F, i)
    $\mathbf{V}_R$.*store*(s_R, i)
    $ttSuccess \leftarrow ttest(m_F, m_R, s_F^2, s_R^2, t)$
    TTEST.*store*$(ttSuccsess, i, t)$
  ttest.clock.disable()
  $t \leftarrow t + k$
return `Passed`

the test to complete. N_S depends on the frequencies at which the cryptographic core and the sensor are running and N_E, the number of cycles need to perform the entire encryption. The valid range for N_S is from one sample to at most $\lfloor N_E \times T_S / T_E \rfloor$, where T_S and T_E are the clock periods of the sensor and the cryptographic core, respectively.

The algorithm has two phases. During the first phase, using the test-only key K, the system runs k encryptions of the fixed and random plaintexts in an alternating fashion. Before every encryption, the cryptographic core is reset, to ensure constant and data-independent initial conditions. During encryption, the t-test core is disabled, minimizing any effects its power consumption may have on the measurements. For simplicity, the random plaintext of the current encryption is the ciphertext from the previous encryption. For each encryption, the according first and second partial sums from (4.10) are updated, resulting in $2 \times k \times N_S$ online updates. During the second phase, triggered after the completion of the first, the system updates the mean, the variance, and the t-test statistic for every power-trace sample. For each batch of k traces, N_S t-test results are produced, which can be used for the first-order leakage detection.

The end condition (whether the device passed or failed the test) depends on the desired use case of the system. For instance, one may want to signal a problem as soon as a leaky sample is detected—a scenario illustrated in Algorithm 1. In another situation, one may want to be less conservative and allow a small, but limited, number of leaky samples during the test, and report failure only if that number is exceeded. In yet another use case, one may want to compare the number of leaky samples between different test runs, to detect if the board has undergone changes or tampering. Whatever the target deployment scenario, the change to be made to the algorithm (and, consequently, the system implementation) remains minimal.

4.4.1 System Architecture and Integration

Fig. 4.2 shows the architecture of the complete built-in leakage evaluation system integrated with a cyber-physical system. It comprises the cyber-physical system with the cryptographic

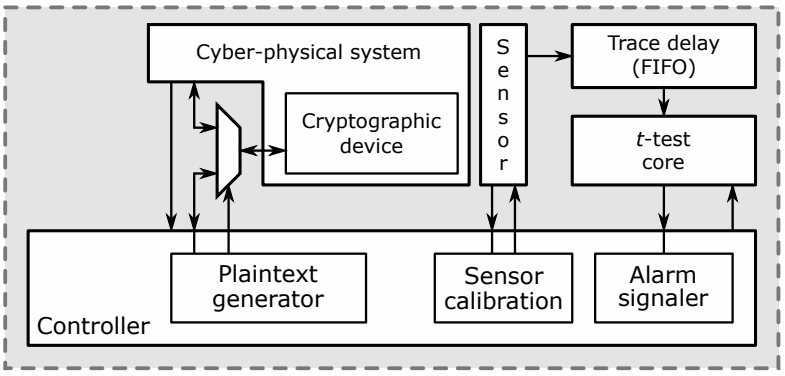


Figure 4.2: Architecture of the built-in leakage evaluation system. The leakage evaluation system is co-located with the cyber-physical system, allowing for fast leakage evaluation with minimal service interruption.

device under test, the TDC-based voltage-drop sensor, the *t*-test core, and the controller.

The TDC-based sensor records one power consumption trace per encryption, and sends it to the *t*-test system as a burst of N_S consecutive samples. The *t*-test core processes the traces and evaluates the leakage in each sample after k traces. A cross-clock domain first-in-first-out (FIFO) buffer stores a single sensor trace before the *t*-test core reads it. This intermediate storage reduces the noise from the traces by disabling the *t*-test core during the encryption process. Moreover, different read and write clocks for the FIFO allow the sensor to work at the maximum possible frequency without over-constraining the timing requirements for the compute-intensive *t*-test core. The controller is responsible for triggering the encryptions, generating the correct plaintexts, disabling the *t*-test core during encryption, correctly calibrating the sensor, and detecting if the alarm should be signaled. The controller is triggered by the cyber-physical system depending on the user requirements: at startup, periodically, or when changes in the system happen (such as impedance changes).

Two ways exist to integrate the *t*-test system into the FPGA-based cyber-physical system. On the one hand, the *t*-test system can be a separate (or partial) bitstream programmed on the FPGA whenever the test is triggered, replacing the cyber-physical system with the *t*-test system and the cryptographic device under test. This method incurs no area overhead for the *t*-test, but requires storing two bitstreams off-chip and significantly increases the unavailability of

the system. In addition to the time required for the test, the FPGA must be reprogrammed twice. On the other hand, having the t-test system co-residing with the cyber-physical system allows performing the test anytime, with minimal interruption. When triggered, the controller logic takes control of the cryptographic device until the end of the test. Colocating the t-test with the cyber-physical system incurs an area overhead. However, with the vast logic resources of the current FPGAs and the latency constraints of cyber-physical systems, some applications might find it more advantageous than reprograming the FPGA for every test. In our system, shown in Fig. 4.2, we colocate the leakage detection logic with the cyber-physical system. This configuration requires two modes of operation:

- *Regular mode*, where the t-test logic is disabled while the cyber-physical system runs in regular operating mode.
- *Test mode*, when the t-test system gains control, disabling the cyber-physical system and performing the t-test on E_{MAX} fixed and random cryptographic encryptions.

4.4.2 Power-Supply Voltage Sensor

Even though FPGAs contain embedded system monitors, these on-chip measurement circuits have a low sample rate and cannot detect nanosecond voltage fluctuations caused by logic switching at high frequencies [11]. Therefore, to capture on-chip power-supply voltage variations, we use a voltage-fluctuation sensor implemented directly on the FPGA. In this work, we use the TDC voltage-drop sensor with a phase-locked loop (PLL) to delay the capture clock, as it can capture voltage variations in time intervals as short as a few ns. We use a priority encoder to convert the thermometer code of the TDC sensor to binary encoding.

4.4.3 Hardware Implementation of the t-test

Fig. 4.3 shows the architecture of the t-test core implementing (4.26). It consists of three main computation blocks: the block for incrementing the partial central sums in (4.10) after each new trace, the block for updating the mean and the variance in (4.13) and (4.24) after

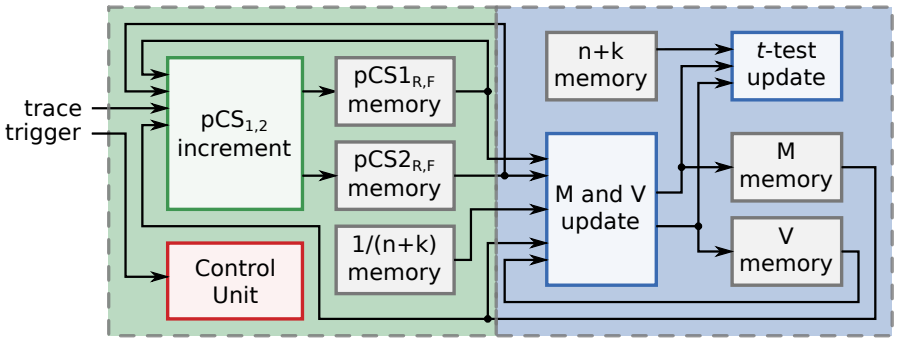


Figure 4.3: Architecture of the t-test block. The blocks in the green part are updated with every trace, while blocks in the blue part are updated after k traces.

k traces, and the block for computing the t statistic in (4.26) after k traces. The t-test core contains BRAM memories for keeping the partial central sums in (4.10), the running means and variances in (4.13) and (4.24), and t-test scores for all N_S power trace samples, for both fixed and random traces. The multiplicative factors $\frac{1}{n+k}$ and $n+k$ are computed offline and saved in read-only memories.

As its input, the t-test core processes the sensor values from the FIFO. The traces processed by the t-test can contain an arbitrarily large number of samples without increasing the datapath resource utilization, as the computation blocks are implemented in a pipelined manner. A control unit, triggered by the start of each AES encryption, ensures that the intermediate values are read and stored at the correct addresses in the corresponding memories.

The t-test computation can be performed in floating-point or fixed-point arithmetic, depending on the resources the host FPGA offers. Our test platform, Sasebo-GII [84], has a Xilinx Virtex-V LX30 FPGA. This FPGA contains fixed-point DSP blocks, which we harness for all the required arithmetic operations. Conversion from floating-point to fixed-point representation leads to precision loss. It imposes the challenge of finding the right number of bits to represent the integer and the fractional parts of each intermediate variable in the t-test computation. The DSP blocks in the Virtex-5 FPGA [85] perform addition/subtraction on two 48-bit wide operands, which enables a minimal loss in precision. However, the DSP multipliers take one 25-bit and one 18-bit wide input. These limited bit widths require careful analysis of the range of values the intermediate variables may have; this analysis is one of the steps in the t-test

calibration process described in Section 4.4.5.

4.4.4 Encryption Core

The encryption core is the circuit that needs to be monitored, because of its susceptibility to power side-channel leakage. As a case study, we use AES-128 encryption cores, but our monitor is sufficiently general to be used with any cryptographic core that leaks information through the power side channel.

4.4.5 System Calibration and Robustness

The *t*-test system requires calibration to evaluate the leakage correctly. The calibration is performed for each device instance, since process variation can significantly impact the sensor performance and the sensor output distribution [86]. The calibration is done for two reasons: first, the delay-line sensor needs calibration so that its output is within the observable range (none of the values reach min/max sensor limits). Second, the width of the integer and the fractional parts of all intermediate variables n the *t*-test system need to be set.

To calibrate the sensor, we run a sequence of encryptions, record the sensor readings, and tune the capture clock shift with a PLL until the maximum value in the steady state reaches ~130. We make the TDC sensor delay line long (160 bits) to ensure that the sensor works in the operating range even when the on-chip voltage drops or becomes excessively noisy due to tampering or changing working conditions. Later work proposed more advanced calibration techniques that maximize the side-channel leakage extraction [86], which are particularly suitable for datacenter-scale FPGAs with a low SNR and peak-to-peak ratio [14, 86]. Our *t*-test system targets low- to mid-end FPGAs used in cyber-physical devices, with a high sensor peak-to-peak ratio and SNR, allowing us to use a more straightforward and shorter calibration procedure. Using a more complex calibration procedure that maximizes the sensor quality, while not necessary, would be desirable in future work.

After the sensor calibration, we run many fixed and random plaintext encryptions and record

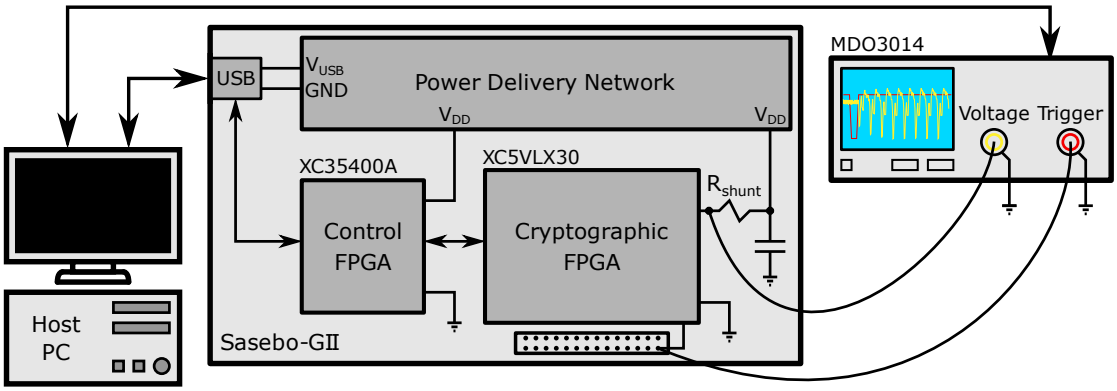


Figure 4.4: Measurement setup used for acquiring the power side-channel traces on the Sasebo-GII board, and evaluating the correctness of the *t*-test system.

the power consumption traces. Then, we compute the *t*-test statistic on the collected traces in floating-point arithmetic. Finally, using the computed *t*-test statistic as a reference, we employ standard methods for floating-point to fixed-point format conversion to find the integer/fractional-part widths of all the intermediate variables, which respect DSP-block constraints that produce accurate *t*-test results. However, if PDN tampering changes the statistical properties of the sensor traces, the *t*-test module should not start having overflows in the DSPs, as this could lead to a loss of precision. In our system, we avoid overflows by increasing the number of integer bits of each variable by two bits after calibrating the *t*-test core.

4.5 Experimental Results

Fig. 4.4 shows our experimental evaluation setup. As our evaluation platform, we choose the Sasebo-GII [84] board, equipped with two AMD FPGAs. One FPGA, commonly referred to as *control FPGA* (AMD Spartan 3A XC35400A), manages the communication with the cryptographic core. The other, *main* FPGA (AMD Virtex-5 LX30 FPGA), implements our *t*-test system. We use a Tektronix MDO3104 Oscilloscope with a 1 GHz sampling frequency and an eight-bit ADC to record oscilloscope traces. A host PC communicates with the device under test (e.g, sending plaintexts, receiving ciphertexts) and the oscilloscope (e.g., receiving traces and the *t*-test output), and checks the correctness of the *t*-test system.

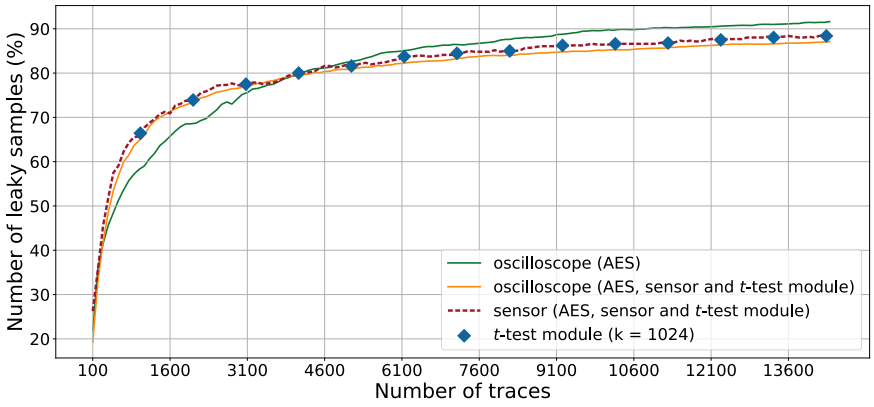


Figure 4.5: The percentage of power trace samples for which the t-test fails for the nonpipelined AES, in the function of the number of traces. Solid lines are obtained using the oscilloscope traces and a t-test software routine in floating-point precision. Dashed lines are obtained using the FPGA sensor traces and the same software routine, while the diamond markers—which almost perfectly overlap the dashed line—are the result of the FPGA t-test module.

We evaluate our system on two AES-128 cores implemented on the main FPGA; the first AES is a nonpipelined 128-bit AES [87], while the second is a 32-bit four-stage pipelined architecture [88]. Both cores operate at 24 MHz—the maximum frequency of the communication bus used for providing the plaintexts and offloading the ciphertexts from the FPGA [12]. The sensor and the t-test core run at 96 MHz, and we set the k parameter to 1024 traces.

To evaluate the system performance, we perform two experiments on each of the two AES cores. First, we record the oscilloscope traces of only the cryptographic core. Second, we record the sensor traces, the t-test core output, and the oscilloscope traces of the entire t-test system. In every experiment, we collect 28×2^{10} traces: half using a fixed plaintext and half using random plaintexts. We repeat the experiments ten times for both AES cores to obtain more statistically significant results.

Figs. 4.5 and 4.6 show the percentage of the *leaky* trace samples in the function of the number of collected traces, averaged over all experiment runs. The full (resp. dashed) curves correspond to the results obtained using the oscilloscope (resp. sensor) traces and a t-test software routine in floating-point precision. The diamonds correspond to the hardware t-test core results. The almost perfect alignment of the diamonds and the dashed curve indicates a very good

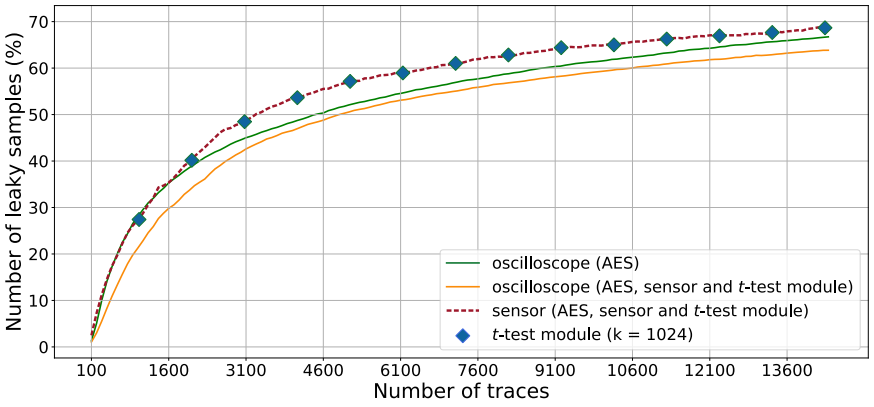


Figure 4.6: The percentage of power trace samples for which the t-test fails for the pipelined AES, in the function of the number of traces. Solid lines are obtained using the oscilloscope traces and a t-test software routine in floating-point precision. Dashed lines are obtained using the FPGA sensor traces and the same software routine, while the diamond markers—which almost perfectly overlap the dashed line—are the result of the FPGA t-test module.

accuracy of our t-test computation in hardware. A common characteristic among all these curves is that the average leakage increases with the number of power traces. Additionally, due to the noise introduced by the components of the built-in self-evaluation system, the standalone AES core (in green) has slightly more leaky samples than the complete system (in blue).

In the case of the nonpipelined AES in Fig. 4.5, the results obtained using the sensor traces closely match those obtained using the oscilloscope traces. However, sensor traces for the pipelined AES in Fig. 4.6 seem to contain more leaky samples than the oscilloscope traces. To understand why, we look at the change in the numerators (subtraction of means) and the denominators (sum of variances) in (4.1) and compare them when using the sensor and oscilloscope traces. The results show that, due to the lower precision of the TDC sensor compared to the oscilloscope ADC, sensor traces for both AES cores have, on average, ~60% smaller sum of variances than the oscilloscope traces. At the same time, for the nonpipelined AES, the subtraction of means has ~70% higher value than for the pipelined AES (because the former occupies more resources, consumes more power, and thus creates higher voltage fluctuations). The values of the numerators in (4.1)—for both the sensor and the oscilloscope traces of the nonpipelined AES—are similar and large enough to compensate for the difference

Table 4.1: FPGA resource utilization breakdown. The number of available resources shown first. In the left columns of the two AES groups, the resources occupied by the encryption core and related modules. In the right columns, the resources used by the leakage-estimation parts of the system.

Resource type	FPGA total	AES		Pipelined AES	
		Encryption core	Leakage estimation	Encryption core	Leakage estimation
LUTs	19200	11.90%	8.23%	5.16%	9.16%
Registers	19200	4.55%	7.31%	6.30%	7.47%
DSP48E	32	-	43.75%	-	40.63%
RAMB18	64	3.13%	6.25%	3.13%	7.81%
RAMB36	64	-	9.38%	-	7.81%

between the corresponding denominators. In the case of the pipelined AES, the denominators in (4.1) have a larger influence on the t-test, resulting in a slightly increased number of leaky samples in the sensor traces.

In the next experiment, we compare the t-test values obtained using the oscilloscope traces and the floating-point computation in software with the values computed by the hardware t-test module. After aligning and downsampling the oscilloscope traces, we count the false positives (samples erroneously declared as leaky) and false negatives (samples erroneously declared as not being leaky). In the case of the nonpipelined AES, we counted only 3.88% false negatives and 5.74% false positives, on average. In the case of the pipelined AES, we counted 5.63% and 13.51%, respectively. These results match the offsets between the corresponding dashed and solid lines in Figs. 4.5 and 4.6.

Table 4.1 summarizes the FPGA resource utilization. The leakage estimation part of the design (the sensor, the control logic, the DSP, and the RAM blocks) uses up to ~10% of the LUTs and registers, up to ~10% of the on-chip memory, and up to ~44% of the DSP blocks available. The minor mismatch between the systems with pipelined and nonpipelined AES is due to the change in the sensor calibration and the width of the AES core datapath.

Compared to the design by Sonar et al. [81], our t-test core requires 3.8× fewer LUTs, 1.7× fewer slice registers, 2.6× fewer BRAMs, and 18.3× less DSP blocks, for the same number of trace samples ($N_S = 64$). Unlike their solution, which requires replicating the t-test module for every new trace sample, we reuse the datapath resources in a pipelined fashion and only

increase the storage requirements for keeping the partial sums, the means, and the variances of the newly added samples only. Finally, Sonar et al. report the maximum relative error of 20% between the floating-point and the fixed-point leakage estimation, whereas we do not calculate the t-test value itself—we calculate its binary equivalent (1 if $|t| > 4.5$, 0 otherwise). Comparing the binary t-test value obtained using floating-point processing of sensor traces with the output of the FPGA t-test core shows the maximum of 0.8% of incorrectly classified trace samples.

4.6 Chapter Summary

In this chapter we have presented and validated a built-in test for self-evaluation of power side-channel leakage, suitable for FPGAs. The system consists of a digital sensor to measure the on-chip voltage fluctuations and an engine to calculate the t-test statistic on-the-fly and thus verify the presence of information leakage. Our design is validated using two AES accelerators implemented on FPGA. When evaluating the t-test statistic, our built-in test achieves results comparable to those obtained using state-of-the-art lab equipment. The system proposed in this chapter allows, for the first time, a real-time assessment of power side-channel leakage during the device operation in the field. This work will open new frontiers for ensuring security of safe and robust cyber-physical systems, which is currently verified only before the deployment.

Part II

Multitenant FPGAs:

Attacks and Defenses

5 Remote Statistical Power Analysis Attacks on Cloud FPGAs

Today, the largest cloud providers, including Amazon, Microsoft, Alibaba, and Baidu, offer FPGA instances in their large-scale datacenters [2–4, 38]. However, the cloud-scale integration and increasing research efforts to support multitenant FPGAs bring several challenges and raise concerns related, in particular, to security. For example, in multitenant systems, it is necessary to guarantee the correct insulation between different users and ensure that one user cannot interfere with or delay the computation of another user. Additionally, preventing information leakage through a side channel is essential for guaranteeing security. In the past, this problem was limited to timing and microarchitectural side channels since the adversary would have needed physical access to the device to mount all other types of physical attacks successfully.

As described in Chapter 3, remote access to low-level logic removes the need for physical access to perform some attacks [12, 13]. Power traces, now captured using FPGA-based voltage sensors instead of classical oscilloscopes, allow attackers to mount power analysis attacks on remote FPGAs. However, the first examples of these attacks were carried out in controlled lab environments with dedicated setups. For example, previous work has shown successful remote power analysis attacks on low- to mid-end FPGAs such as AMD Virtex-V [12] or AMD

This chapter is based on the work of a paper published at the 2020 Design, Automation & Test in Europe Conference & Exhibition [14].

Zynq-7000 [13] FPGAs, containing 19.2k and 53.2k LUTs respectively. In contrast, most modern cloud FPGAs are from the AMD Ultrascale+ FPGA family [2,3,37,38] and contain approximately 1–2 million LUTs. These FPGAs are 10–20× bigger than those used in previous work, have new architectures of FPGA resources, and have higher quality PDNs.

While giving the intuition that the threat of remote power analysis attacks could also be exploitable in a real cloud environment and high-end cloud FPGAs, previous work did not provide any clear evidence of this fact. The work in this chapter bridges this gap. For the first time, we show a successful remote key recovery attack on a 128-bit AES accelerator on a real cloud system: Amazon EC2 F1 instances. To collect the power traces, we carefully port the TDC sensor and adapt it to be tolerant to carry look-ahead chains available in high-end Xilinx Virtex Ultrascale+ FPGAs, which, unlike ripple-carry carry chains on low-end FPGAs, do not have monotonic delay increase at their outputs. Our results demonstrate that high-end cloud FPGAs have a lower SNR than small, low-end FPGAs, requiring significantly more traces to break the key successfully. Nevertheless, our work shows that power side-channel vulnerabilities exist even in cloud FPGAs and that cryptographic circuits should not be unprotected in a future multitenant scenario.

In the remainder of this chapter, we first introduce the threat model. Then, in Section 5.2, we discuss the issues with porting TDC sensors to state-of-the-art FPGAs used in the cloud. Section 5.3 describes the architecture and FPGA implementation of our system. Section 5.4 presents and discusses the experimental results, while Section 5.5 concludes the chapter.

5.1 Threat Model

In a co-tenancy or multitenancy scenario, multiple users share the same reconfigurable logic and deploy their hardware tasks on it. Every deployed task and its computation is a relevant intellectual property of a user and, as such, must be protected. This is achieved with logical and physical separation of the tenants [89]. However, unwanted interaction between the tenants can still happen through the PDN. This can introduce several security risks, including

the leakage of secret information via the power side channel.

In this work, we assume that a victim uses a subset of the FPGA resources to encrypt data with a secret key, while the other tenant is malicious, with the aim of attacking the *confidentiality* of the victim. Logical and physical separation of the tenants ensure that the adversary is not able to connect to any of the signals in the victim logic—he has only the indirect access to the shared PDN. Furthermore, we assume that, after performing the encryption, the victim sends the ciphertexts over a public channel that can be observed by the adversary.

5.2 Porting the TDC Sensor on Ultrascale+ FPGAs

While older FPGA generations contain ripple-carry adders, CARRY8 blocks in state-of-the-art, high-end UltraScale+ FPGAs are implemented as carry look-ahead adders [56]. Therefore, the main challenge of implementing TDC sensors on Amazon EC2 F1 instances is that the delays of the CARRY8 outputs do not increase monotonically. In other words, the delay from the input of the carry chain *CIN* to the individual carry outputs $CO_i, 0 \leq i \leq 7$ does not increase monotonically when all the other inputs are constant and set so that *CIN* can propagate. In practice, a carry look-ahead architecture implies that CO_7 output of the last reached CARRY8 in the observable delay line can be high while the other carry outputs are low, producing unexpected sensor readings.

Since the sensor output is usually encoded using a priority encoder which expects a monotonically increasing thermometer code at its input, unexpected sensor readings can be incorrectly encoded, reducing the quality of the sensor. There are two commonly used priority encoder implementations for TDC sensors. In the first, the encoded value represents the position of the last bit equal to one, regardless of the value of the previous bits. For example, values `11110000` and `1001000` would both be encoded as a decimal `4`. Since metastability can cause an unstable transition from one to zero in the sensor output register even when using ripple-carry adders [90], a second, more robust encoder returns the position of the last bit equal to one followed by more than one zero. The robust priority encoder effectively ignores values

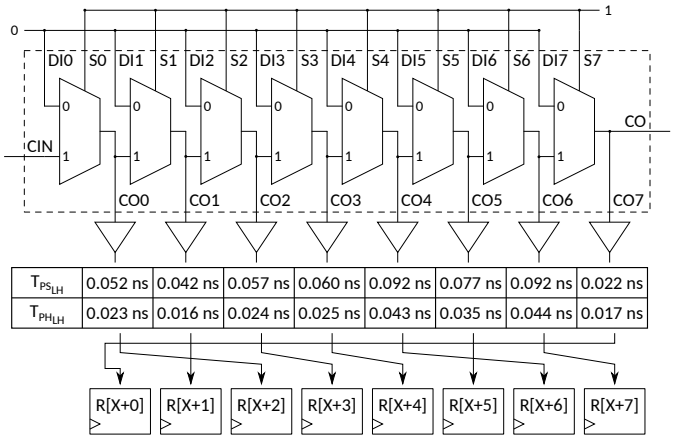


Figure 5.1: CARRY8 internal architecture and nonmonotonic delays from input *CIN* to the outputs CO_i, $0 \leq i \leq 7$. To increase the likelihood of monotonically increasing delays with the increase in the CARRY8 output index, it suffices to permute the CARRY8 outputs before they reach the sensor register. The permutation is devised from a detailed timing analysis of all paths within one CARRY8 element.

after large bubbles in the transition from one to zero. For example, values `11110000` and `11010000` would be encoded as `4`, while `10010000` is encoded as `1`. However, in the case of carry look-ahead adders, both of the aforementioned encoders are incompatible with the nonmonotonic sensor output, as the former can encode a large number of zeros as one, and the latter ignores bits set to one after any bubble larger than two zeros (which often happen with fast carry outputs of the carry look-ahead adders).

Fig. 5.1 shows the delays of the CARRY8 outputs on the UltraScale+ FPGA on Amazon EC2 F1, extracted using setup and hold timing analysis in Vivado. We conclude that to achieve a monotonic increase in the output delays needed for the priority encoders, CARRY8 outputs should be arranged in the following order: CO_7, CO_1, CO_0, CO_2, CO_3, CO_5, CO_4, CO_6. To implement the reordering, it suffices to permute the order of signals between the sensor register and the encoder, as illustrated in Fig. 5.1. Alternatively, calculating the Hamming weight of the sensor output—counting the number of output bits set to one—avoids using priority encoders entirely, as it implicitly orders the output bits in monotonic order. Hamming weight represents an implementation-agnostic way of encoding the sensor output, as it does not depend on the underlying CARRY architecture.

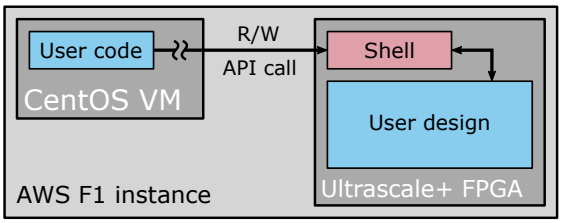


Figure 5.2: Amazon EC2 F1 Instance architecture.

In this work, we record the state of the sensor's entire observable delay line, allowing us to process and encode the sensor output in software and directly compare different encoding methods. We refer to the traces permuted in software and encoded with the robust encoder as *permuted*. Conversely, we refer to the traces obtained using the delay line values encoded without permutations as *nonpermuted*. Finally, traces obtained by calculating the Hamming weight are referred to as *HW*.

5.3 System Architecture and Experimental Setup

The target system of the attack that we present in this chapter is an AES cryptographic core running on an Amazon EC2 F1 instance. Fig. 5.2 shows the simplified architecture of the FPGA instance. It consists of a virtual machine (VM) with the CENTOS operating system, with all the drivers and APIs needed to communicate with the FPGA. This VM can run any user code in C, including a correlation power analysis attack. The instance also contains a Xilinx Virtex Ultrascale+ FPGA that has a static *privileged shell*, used for controlling the communication between the user design and the C code in the VM.

Fig. 5.3 illustrates our system architecture, composed of several modules: First, an open-source 128-bit AES core [87] with an AXI-Lite wrapper. Then, a 160-bit voltage drop sensor to measure the power side-channel leakage, a robust priority encoder to encode the propagation depth of the clock to a corresponding binary number, and a true dual-port BRAM memory to store the sensor output. The remaining modules enable communication between the AES, the BRAM, and the shell. Our system has four clock domains. The shell provides the main clock (125 MHz), which drives the interconnect, the BRAM controller, and the read port of the BRAM.

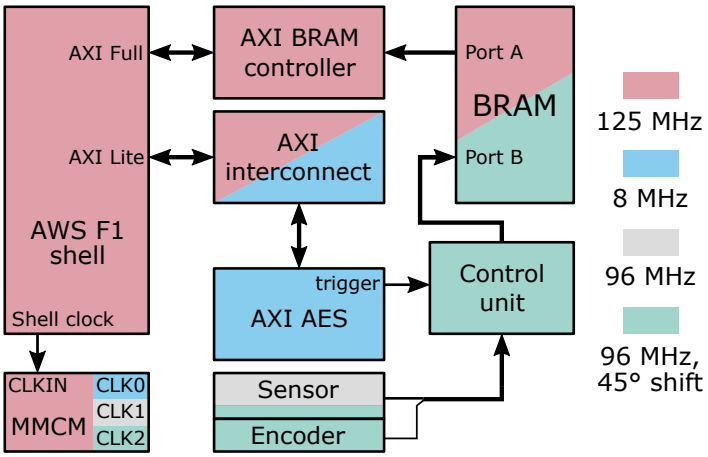


Figure 5.3: System architecture.

We derive all the remaining design clocks from the main clock. A 96 MHz clock drives the sensor delay line, while a phase-shifted 96 MHz clock is used for capturing the sensor output, the encoder, the control unit, and the BRAM write port. The phase-shifted clock is used for sensor calibration purposes, to delay the capture clock of the TDC sensor and ensure the rising edge transition falls in the sensor observable delay line. The phase shift of 45° was chosen experimentally during the sensor calibration process.

Our system simulates a potential victim by sending plaintexts to the AES core from the VM. Upon the start of each encryption, the AES core asserts a trigger signal, and a fixed number of consecutive sensor readings (encoded and raw) are stored in the BRAM. After the encryption, the ciphertext and the power consumption trace are communicated to the VM in two read requests. These steps are repeated for every new plaintext.

The implementation of our system satisfies the principle of physical separation, as a column of unused DSP blocks separates the AES core and the sensor. Even though we use the AES trigger signal to synchronize the power traces, it is done for practical reasons and can be avoided using trace alignment techniques on the sensor traces [12], in which case, the system would also satisfy the principle of logical separation.

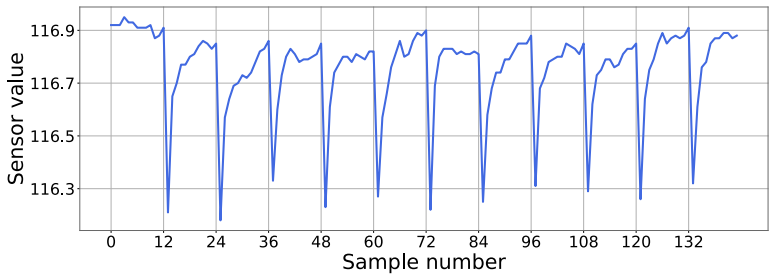


Figure 5.4: Waveform obtained by averaging a hundred power-consumption traces. Plaintext loading and all ten AES encryption rounds can be clearly identified.

5.4 Experimental Results

As our first experiment, we analyzed the permuted power traces of the AES cryptographic core. Across all the samples in 1M collected traces, only five distinct values appeared: 113, 115, 116, 117, and 121, whereas the sensor operating range was 1–159. Fig. 5.4 shows the waveform obtained after averaging a hundred power consumption traces collected from the sensor, where each trace corresponds to one encryption. The plaintext loading (the first dip) and the subsequent ten rounds of AES encryption are all clearly visible in the power trace. Hence, the traces, albeit represented with only five distinct values, contain information that an attacker could potentially exploit.

For the second experiment, we perform the CPA attack on the last round of the AES encryption, using the permuted traces. We mount the attack using 10^6 traces, with each byte attack leading to the successful recovery of the secret key byte. Fig. 5.5 illustrates the results of the attack on the seventh byte of the key. On the left side, the correlation for the correct key guess (in black) reaches the maximum at sample 133 of the power trace. On the right side, we show how the rank of the correct key guess evolves with the increase in the number of traces used in the attack. We can observe that after a short instability, the rank of the correct key guess reaches the value of one, and the key is successfully broken.

In our third experiment, we repeat the attack but this time on nonpermuted traces. Fig. 5.6 shows that the attack is still successful, although, the key rank drops to one after a much higher

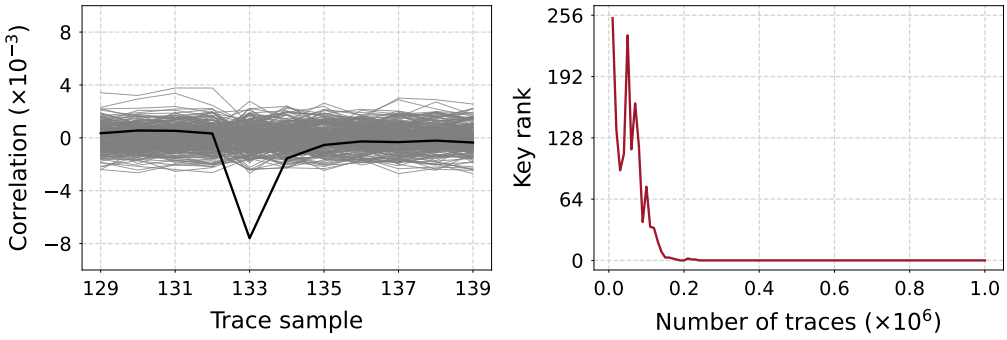


Figure 5.5: CPA attack on the seventh byte of the AES encryption, using 10^6 permuted traces. On the left, the correlation for all key guesses, with the correct key candidate in black. On the right, key rank evolution of the correct key candidate.

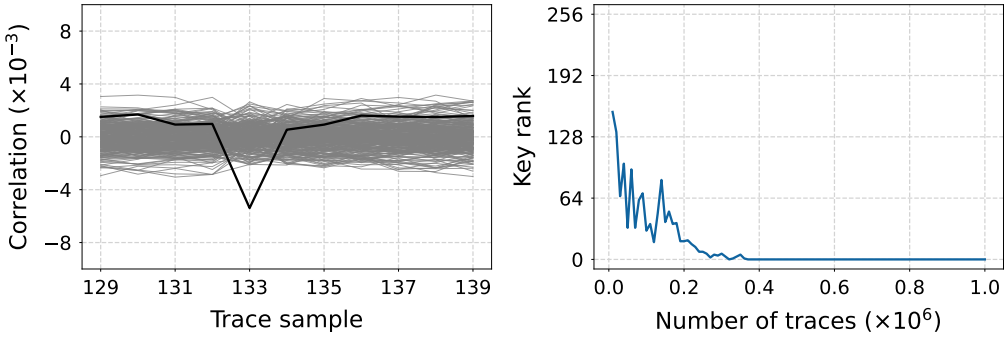


Figure 5.6: CPA attack on the seventh byte of the AES encryption, using 10^6 nonpermuted traces. On the left, the correlation for all key guesses, with the correct key candidate in black. On the right, key rank evolution of the correct key candidate.

number of traces.

To evaluate the security of the entire key, we use the CPA results from the previous experiments and calculate the key rank estimation metric outlined in Chapter 3.3.2. Fig. 5.7 shows the results on the nonpermuted and permuted traces. In the case of permuted traces, 0.65×10^6 traces were sufficient to break the entire key. Without permutation, 10^6 traces were required for a successful attack. Our results demonstrate that permuting the sensor output before encoding results in fewer traces needed to break the entire key. As discussed in Chapter 5.2, permuting the sensor output bits prevents information loss during encoding, as it prevents large bubbles caused by the carry look-ahead logic in newer FPGA families.

To obtain more statistically relevant results, we repeat the previous experiments 30 times.

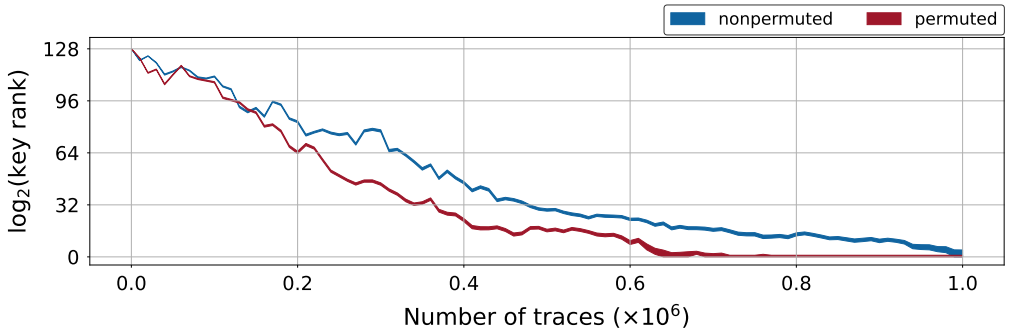


Figure 5.7: Key rank estimation when attacking the full 128-bit key of the AES encryption for the nonpermuted and permuted traces.

Before every experiment, we shut down and restart the Amazon EC2 F1 instance, potentially having the resource allocator assign a different FPGA every time. For each repeated experiment we record 10^6 traces using the same bitstream, sensor calibration, and AES inputs. Out of all attempts to attack the individual key bytes using 10^6 traces, 48% were successful. In 73% of the successful key-byte attacks, the CPA on the permuted traces succeeded with the smaller or equal number of traces than the CPA on the nonpermuted ones. Moreover, the CPA on the permuted traces required up to 88% (on average 20%) less measurements than the CPA on the nonpermuted traces, to retrieve one byte of the key.

To analyze why some key-byte attacks were not successful, we calculate the key rank estimations when attacking the entire key. Fig. 5.8 shows the results for all 30 runs with permuted traces. We can observe that in some cases, the attack using 10^6 results in virtually no broken key bits, while some attacks result in the entire key being revealed with as little as 0.6×10^6 traces. While the sensor was correctly calibrated in all 30 experiment runs, using the same calibration for all experiments does not guarantee the same measurement quality on different Amazon EC2 F1 FPGA instances. As discussed in Chapter 4.4.5, variations in sensor performance are expected across device instances because device timing characteristics, and thus sensor measurements, are known to be affected by process variations, temperature, and aging [91]. For example, Drewes et al. [86] later showed that in the case of datacenter-scale FPGAs the sensor quality depends on where the clock edge lands in the delay line at the moment it is sampled. Landing between two carry elements in the sensor carry chain results

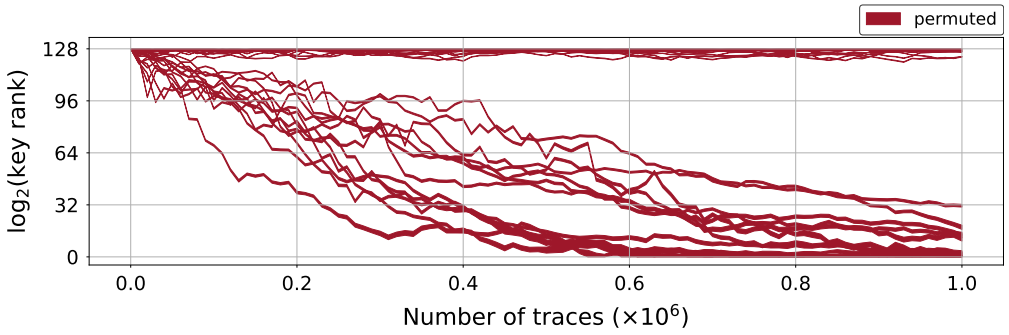


Figure 5.8: Key rank estimation for all 30 different experiment runs on the Amazon EC2 F1 instances, using permuted traces.

in poorer measurements than landing within the bits of a carry element. Since Fig. 5.4 shows the peak-to-peak ratio of the sensor is already low—less than one sensor bit when averaging 100 traces—any variation in the delay can cause changes in the quality of the sensor measurements. Drewes et al. [86] later demonstrated a calibration method for maximizing the side-channel information measured by the sensor, by carefully tuning the phase between the input and sampling clock of the sensor. The results in Fig. 5.8, along with the results from Drewes et al. [86], demonstrate the need for careful sensor calibration, especially on large datacenter-scale FPGAs with a small SNR.

As our last experiment, we compare the power side-channel attack results for the three different sensor encodings. Fig. 5.9 shows the averaged results over the 30 runs for nonpermuted, permuted, and HW traces. Since the key rank estimation has an upper and lower bound for each run, we calculate and plot the area between the average lower and average upper bound over the 30 runs. From Fig. 5.9, we can observe that, on average, permuted and HW traces outperform the nonpermuted traces. As the nonpermuted traces have neither an explicit nor implicit monotonic ordering before the encoding, the fast propagating carry bits of the carry look-ahead adders are ignored in the transition from zero to one, resulting in a less successful attack. Permuted and HW traces demonstrate similar trends in Fig. 5.9 and break the same number of key bits since both encodings preserve the notion of monotonicity. Indeed, Drewes et al. [86] later showed that the Hamming weight encoding maximizes the recovered leaked information in case of workload classification on shared FPGAs. Since Hamming weight is

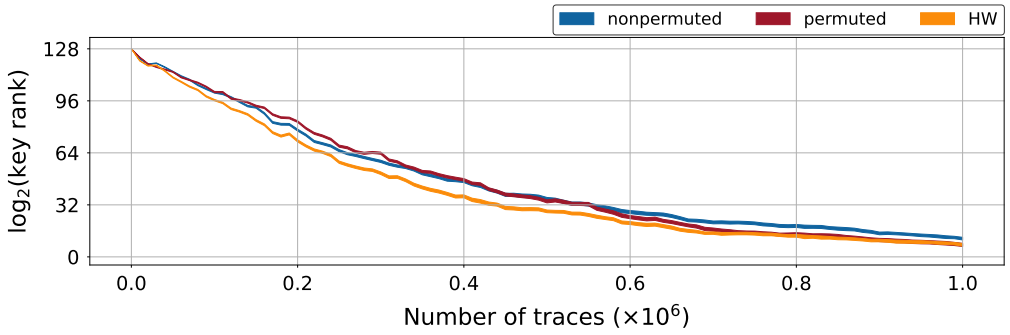


Figure 5.9: Key rank estimation for three different sensor output encodings on the Amazon EC2 F1 instances, averaged over 30 experiments.

easier to implement and port across different FPGA families, we use it as the encoding of choice in the remainder of this thesis.

5.5 Chapter Summary

When deployed, multitenant FPGAs will offer new opportunities but will also expose new security threats. Power analysis attacks are potentially among them since recent work showed the possibility of remotely extracting the secret key of a cryptographic algorithm by mounting an attack on the power traces obtained from sensors implemented using the FPGA fabric. These experiments were, however, mostly carried out on FPGA development boards in a controlled environment and not on a real system deployed on the cloud. In this Chapter, we demonstrate, for the first time, a successful key recovery attack on a cryptographic accelerator running on an Amazon EC2 F1 instance. As a case study, we used the AES-128 algorithm. However, our attack is applicable to any cryptographic core susceptible to power analysis attacks. Our results demonstrate that the security concerns raised by multitenant FPGAs are indeed valid and that countermeasures should be put in place to mitigate them.

6 Instruction-Level Power Side-Channel Leakage Evaluation of Soft-Core CPUs

In the FPGA-accelerated cloud, highly-parallel tasks are accelerated on FPGAs. At the same time, developers rely on host CPUs for general computation, particularly noniterative and user event-dependent control algorithms, which are significantly easier to implement and maintain in software than in hardware. However, the FPGA-CPU communication incurs high latency, especially for short data transfers [92]. If such delays are of no concern, then the control software can be deployed on a cloud CPU instance; yet, only a limited range of FPGA applications—usually data movement ones—can afford the resulting communication latency. Therefore, in the case of latency-critical control algorithms, system designers resort to using soft-core CPUs as real-time co-processors (e.g., Microblaze [93], Nios [94], PicoRV [95]), which allow tight and customizable integration with FPGA accelerators, and short communication latencies.

As we demonstrated in Chapter 5, FPGA multitenancy introduces security threats that cannot be mediated by physical or logical isolation between tenants. Including our own work, several remote power-analysis attacks have already been demonstrated: a simple power analysis (SPA) attack on Rivest-Shamir-Adleman (RSA) exponentiation [13], CPA attacks against AES (requiring a large number of victim power traces) [12, 14, 48], and reverse engineering attacks

This chapter is based on the work currently under review for the Springer Journal of Hardware and Systems Security.

on neural network accelerators (which occupy a significant portion of the FPGA resources) [15, 96–98].

For an FPGA user, secret information is not limited to their bitstream, the cryptographic key, or neural network accelerator parameters and architecture. If their design contains a soft-core CPU, the code being executed can be proprietary or contain secrets. If an attacker, by observing power side-channel traces during CPU code execution, can determine which instructions are being executed, the confidentiality of the code will be compromised. In embedded applications and smart cards, where adversaries have physical access to the target device to measure power and electromagnetic side-channel leakage, attacks that aim at code recovery are termed side-channel disassembly attacks [99, 100]. Unlike statistical-based power analysis attacks such as CPA, side-channel disassembly attacks are profiling attacks and assume the attacker can record a limited number of victim execution traces.

Our subsequent work takes an evaluator's point of view: we explore to which extent soft-core CPUs leak instruction-level information through the remote power side channel, in cases when an evaluator (or a potential attacker) has no physical access to the device but can deploy on-chip voltage-drop sensors. Unlike traditional side-channel disassembly attacks—where the CPU runs at frequencies orders of magnitude lower than the sampling rate of the oscilloscope—sensors used in remote power analysis attacks have sampling frequencies in the same operating range as soft-core CPUs. Our work analyzes if and under which conditions soft-core CPU instructions contain power side-channel leakage and incentivizes the use of protection methods in multitenant FPGAs. As our leakage evaluation targets, we choose two RISC-V soft-core CPUs using the 32-bit RISC-V base integer instruction set architecture (RV32I), most suitable for lightweight real-time co-processors [101].

To start, we record the side-channel traces corresponding to the execution of every CPU instruction. Then, to investigate whether the traces leak secrets, we train diverse ML classifiers used in previous work and also explore the use of novel deep learning (DL) classifiers to improve the extraction of the power side-channel leakage. The results reveal that, despite the

limited accuracy and sampling rate of on-chip sensors compared to oscilloscopes used in disassembly attacks with physical access, the limited leakage compared to previous remote reverse-engineering attacks, and the limited number of victim trace acquisitions compared to statistical-based attacks, instruction-level leakages still exist: we can determine the executed instructions with average accuracy higher than 80%. These results call for proper mitigations to limit power side-channel leakage of soft-core CPUs in shared FPGAs.

We make the following contributions:

- To the best of our knowledge, we present the first analysis of instruction-level leakage of soft-core CPUs in a shared FPGA setting.
- While power side-channel traces recorded by an on-chip FPGA sensor during the execution of one RISC-V soft-core CPU instruction contain limited visually observable leakage, we demonstrate that, in certain conditions, advanced ML techniques can extract sufficient information to identify the opcode of the executed instructions. The maximum average instruction accuracy we achieve on the RV32I instruction set architecture (ISA) is 86.46%.
- Besides evaluating previous side-channel disassembly approaches, we explore new, DL-based instruction classifiers, and experimentally find that they are superior at extracting leakage compared to common ML techniques deployed in previous work, and should be used for future side-channel security evaluations.
- We perform an extensive experimental analysis that compares how different leakage evaluation scenarios, such as the number and placement of sensors, number of templates, and type of templates, affect the instruction-level leakage. We also demonstrate our results on two soft CPU cores and two different FPGA families. In addition to the leakage analysis of the RISC-Y [102] soft-core CPU running at 80 MHz on the Sakura-X board [103], we show results on a cloud-scale, AMD Alveo U200 datacenter accelerator card, using the compact PicoRV [95] soft-core CPU, running at 320 MHz. With our on-chip sensors running at 320 MHz, the side-channel traces have only four sensor

samples per CPU clock cycle on Sakura-X, and only one sensor sample per CPU clock cycle on Alveo U200; significantly lower than in traditional side-channel disassembly attacks.

- We provide a detailed discussion of our experimental results and their impact on soft-core CPU leakage evaluation, which we use to motivate appropriate mitigation techniques.

Our work aims to provide a leakage evaluation methodology for soft-core CPUs in remotely accessible scenarios and to benefit future power side-channel disassembly attacks by providing novel DL power trace classification techniques. Therefore, we make all our FPGA designs, associated software, and ML code openly available for the reproducibility of the experiments and the results in this work [104].

The remainder of the chapter is organized as follows. In Section 6.1, we describe the threat model, while Section 6.2, we explain the experimental setup in detail. Section 6.3 covers dataset generation and the DL classification models used in our work. Experimental results are presented in Sections 6.4 and 6.5. Section 6.6 discusses the results, while Section 6.7 discusses potential countermeasures. Finally, Section 6.9 concludes the chapter.

6.1 Threat Model

Research on the security of multitenant FPGAs follows a well-established threat model of the fault and side-channel attacks on remote shared FPGAs [13–15, 32, 51, 52, 63, 105]. The primary assumption is that at least two users can remotely deploy their designs on the same FPGA instance simultaneously. For security reasons, these remote users are given control over dedicated partial reconfiguration regions, which are logically and physically isolated; thus, the attacker has no direct access or control over the victim or the victim's deployment. The adversary can deploy voltage fluctuation sensors to record power side-channel traces and send them over the network for remote analysis.

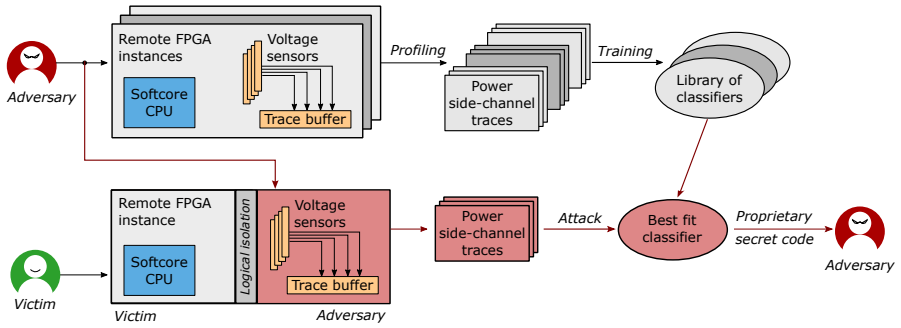


Figure 6.1: Threat model. The top half illustrates the profiling phase, which results in a library of side-channel instruction classifiers, for a number of FPGA instances and CPU and sensor placements. The bottom half shows the attack.

In this work, we assume an evaluator's point of view: we evaluate the security of a victim using a soft processor core in their shared FPGA platform, for example, to configure and control the operation of an accelerator. This work analyzes instruction-level leakage to assess if and under which circumstances soft-core CPUs leak instruction information through the power side channel in shared FPGAs, with the goal of motivating the use of countermeasures.

When evaluating the side-channel security of a device, it is a common practice to consider the worst-case estimates (even if not practically achievable by an attacker), as they quantify the limits of the leakage. For example, in the context of cyber-physical devices, white-box power side-channel leakage evaluation methods leverage proprietary architectural information (unavailable to attackers) to build better power models for power analysis attacks [68]. Removing the plastic cover of a chip to record near-field EM emanations is another example of a common practice in leakage evaluations, even though attackers might not always be able to remove the casing. Consequently, our experiments assume and evaluate various scenarios: from worst-case (a breach of physical and logical separation, no additional noise sources, and averaging of traces) to more realistic scenarios, including physical separation, no averaging, and noise from surrounding instructions and the shell.

In reality, a hypothetical attacker mounting a profiling attack on soft processor cores would have to perform a procedure similar to the one shown in Fig. 6.1. To prepare for the attack, an adversary would start by renting an FPGA instance as its only tenant. On this FPGA

instance, the adversary would need to calibrate the voltage fluctuation sensors and use them to profile the execution of the CPU instructions for various operating frequencies and several CPU placements. Then, the attacker could train side-channel instruction classifiers. This step would have to be repeated for many FPGA instances, each uniquely identified (e.g., by fingerprinting cloud FPGAs as suggested by Tian et al. [106]).

To perform an exploit using the library of trained classifiers, the attacker would need to rent a shared FPGA instance. Using fingerprinting to identify the shared FPGA instance, the attacker can focus on the subset of the classifiers in the library trained on that particular FPGA instance. Once side-channel traces are obtained, the adversary would need to identify that the co-located user is using a soft-core CPU (and repeat until a victim with a soft-core CPU is identified), using workload classification techniques [105]. Then the attacker could further prune the subset of trained classifiers using the same workload classification techniques—which can distinguish between different soft-core implementations in shared FPGAs—and run the inference. Finally, in addition to the attack procedure, the attacker would need to train models robust to noise from the shell or any other accelerator the victim might be using alongside their soft processor core.

Our aim is to evaluate how and under which circumstances soft-core CPUs leak instruction information in shared FPGAs, we therefore focus on assessing instruction leakage. We refer to related work for FPGA identification and workload classification.

6.2 Experimental Setup

The Sakura-X (Sasebo-GIII) board [103] and the Alveo U200 datacenter accelerator card serve as our target evaluation platforms. Sakura-X is an evaluation board designed for power side-channel analysis and, hence, commonly used in both cryptologic research [107, 108] and research on side-channel attacks on shared FPGAs [12, 47, 98]. Sakura-X has one AMD Kintex-7 FPGA and one AMD Spartan-6 FPGA. The former FPGA is the larger of the two, often referred to as *main* or *target* FPGA, as it hosts the adversary and the victim as two logically isolated FPGA

tenants. The second FPGA, often referred to as auxiliary or control FPGA, reduces unwanted noise by implementing the communication protocol between the target FPGA and the host machine [103]. For our evaluation, the Sakura-X architecture increases the already low SNR of soft-core CPUs and helps isolate the instruction-level power side-channel leakage. To evaluate the leakages in a more realistic, cloud-scale FPGA scenario, we use the Alveo U200 datacenter accelerator card. This card contains an AMD UltraScale+ XCU200-2FSGD2104E FPGA, and is commonly used in publicly available cloud FPGA instances [2]. Unlike Sakura-X, Alveo U200 contains a single FPGA consisting of three super-logic regions (SLRs). The shell, containing resources necessary for communicating with the DRAM and host CPU, is instantiated in the middle SLR and physically separated from both the attacker and the victim. The placement of the sensor and the victim CPU varies across experiments, however, in most cases, we physically separate the sensors and the victim soft CPU core to conform with the standard shared FPGA threat scenario described in Section 6.1.

Fig. 6.2 gives an overview of the experimental setup for both boards. The target FPGA design contains the victim and the hypothetical attacker logic and has four main components: a soft-core RISC-V processor, the on-chip voltage-drop sensors, the control finite state machine (FSM), and the shell. As discussed at the beginning of Chapter 6, the primary purpose of using soft-core CPUs is to implement latency-critical control algorithms, especially ones subject to change over time. Therefore, our study assumes the victim uses small soft-core CPUs, common in embedded bare-metal applications [109]. These soft-core CPUs are usually lightweight, with no advanced microarchitectural features such as cashing or speculative execution. They have a low area overhead and can run at high clock frequencies. Their microarchitectural simplicity allows easy and tight integration with FPGA hardware, facilitating low-latency communication. Integrating larger soft-core CPUs would reduce the operating frequency (e.g., Rocketchip can run on a couple of tens of MHz only [110]), increase the area overhead (reducing the available resources for hardware accelerators), and adversely affect the communication latency (as communication would take place through memory mapped interfaces or an operating system).

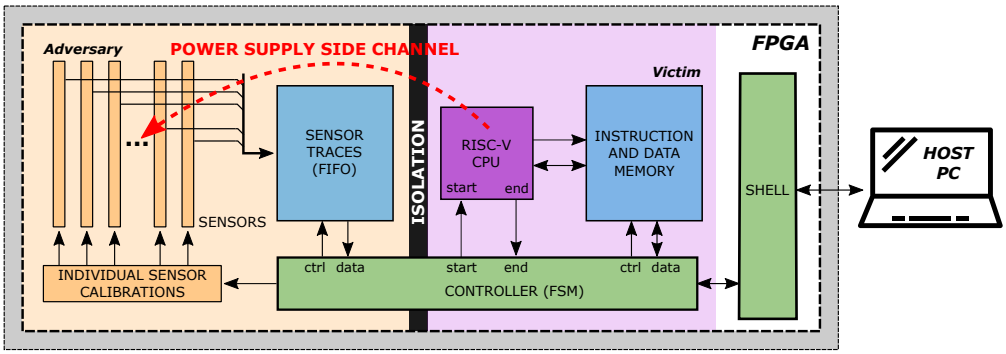


Figure 6.2: Overview of the experimental setup.

Table 6.1: Resource utilization of the soft-core CPUs.

CPU	FPGA	LUT	FF	BRAM36	DSP
RISCY [102]	Kintex-7 XC7K160T-1FBGC	2544	1944	40	0
PicoRV32 [95]	Virtex Ultrascale+ XCU200-FSGD2104-2-E	1442	1473	8	0
Rocket Chip [110]	Virtex Ultrascale+ XCU200-FSGD2104-2-E	25785	12654	12	15

For the RISC-V soft-core designs, we chose RISCY and PicoRV32, both openly available [95, 102]. Table 6.1 summarizes the FPGA resource overhead. As a reference, we also show the resource usage of Rocket Chip [110], a larger, more complex soft-core RISC-V implementation. RISCY, used on the Sakura-X board, implements a classic five-stage pipeline and supports the complete RV32I ISA at the cost of a lower operating clock frequency. On Sakura-X, the maximum operating frequency of the RISCY CPU is 100 MHz; however, our system runs it at 80 MHz, to have an integer number of sensor samples per one CPU clock cycle. PicoRV32, used on the Alveo U200 board, has a multicycle CPU microarchitecture designed to minimize resources and maximize the CPU operating frequency. Our system runs PicoRV32 at the maximum operating clock frequency of 320 MHz.

6.2.1 FPGA Voltage-Drop Sensors

In this work, we use TDC sensors with the reconfigurable initial delay line described in Chapter 3. Through experimentation, we found that a 16-bit observable delay line is sufficient

Table 6.2: Coarse calibration, fine calibration, and observable delay line slices per sensor.

FPGA	Fine calibration	Coarse calibration	Observable line
Kintex-7 XC7K160T-1FBGC	24 slices (= 96 stages)	8 slices (= 32 LUTs and Latches)	4 slices (= 16 FFs)
Virtex Ultrascale+ XCU200-FSGD2104-2-E	12 slices (= 96 stages)	4 slices (= 32 LUTs and Latches)	2 slices (= 16 FFs)

to capture the supply voltage variations caused by the CPU operation on both FPGA boards. Table 1 lists the FPGA resources used for our TDC implementation on both boards. The sensor clock frequency was set to 320 MHz on both boards, the highest operating frequency that satisfied timing constraints. Consequently, the sensor captures four samples per one clock cycle of the RISCY CPU running at 80 MHz, and one sample per clock cycle of the PicoRV32 CPU running at 320 MHz.

Previous work has shown that the side-channel information captured by voltage-drop sensors varies with both the absolute location of the sensors as well as their relative position to the victim [111]. It is, therefore, to be expected that an attacker may instantiate more than one power side-channel sensor. The exact number is usually limited by the linearly scaling on-chip memory resources and the data transfer word size. For example, to improve the success of their attack, Gravellier et al. [48] deployed eight sensors on an AMD Artix-7 FPGA. In our experimental setup, we instantiate five TDCs on Sakura-X, and 29 TDCs on Alveo U200, the highest number that fits in a communication message exchanged between the FPGA [103] and the host PC. In Sections 6.4 and 6.5, we will show to what extent having multiple sensors affects the attack efficiency.

6.2.2 Controller

The controller coordinates the experiments by executing and replying to the commands from the host machine through the shell. It is in charge of initializing the CPU instruction memory with the code to be executed, triggering the execution of the code, and saving the corresponding sensor traces to the on-chip memory. Once the CPU code execution is completed, the controller receives a trigger from the CPU, which initiates the transfer of sensor traces to the

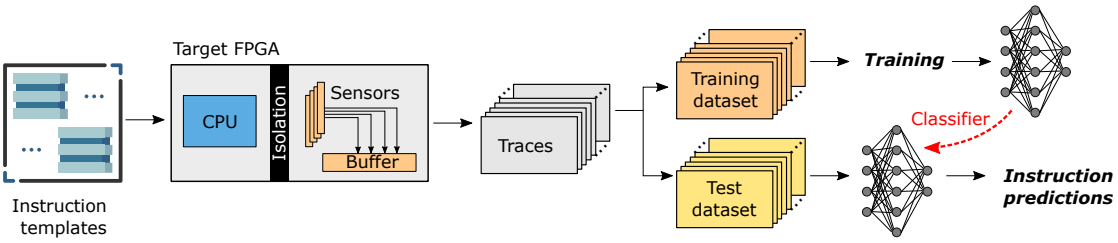


Figure 6.3: Side-channel instruction leakage evaluation.

host machine. In each message sent from the FPGA to the host, the controller inserts five (Sakura-X) or 29 (Alveo U200) simultaneous sensor readings and the 32-bit word of the corresponding CPU instruction. We replace the default read-only instruction memory of both CPUs with a dual-port block RAM, connecting one memory port to the CPU while exposing the other port to the controller. This temporary change permits the controller to write arbitrary code in the CPU instruction memory before triggering its execution and recording the side-channel traces.

Prior to starting the experiments, the controller calibrates every sensor. The calibration is performed iteratively. First, a test code sequence is loaded to the instruction memory, and the number of elements in the sensor's initial delay line is set to zero. The code execution is triggered, and the obtained sensor trace is inspected. If no clock transition is observed or the transition is located too close to the two extremes of the observable delay line, the fine and coarse calibration slices are adjusted. This process is repeated until the sensor is calibrated. The calibration settings are then communicated to the host machine for record keeping.

6.3 Instruction Classification

Like all hardware circuits, soft-core CPUs leak information through the power side channel. Various ALU operations, memory accesses, and control-flow changes all impact power consumption differently. In addition, as a combination of fetch, ALU, memory, and program counter operations, instructions also leak information: in the form of unique patterns spread across the time and amplitude domain of the recorded power traces. For example, on the one hand, memory instructions might have high power consumption both in the ALU stage, when

the address is computed, and in the later stages of instruction execution, i.e., when the data is read/written to the memory. On the other hand, arithmetic instructions might only have a power consumption peak during the ALU stage.

To analyze the instruction-level power side-channel leakage of soft-core CPUs, we employ an ML-inspired method illustrated in Fig. 6.3. The key idea behind this approach is that leakage patterns are discovered during ML model training, while the leakage is assessed using the prediction accuracy achieved on templates unseen during training. For this purpose, we first build a large set of template assembly codes for all the target instructions: we generate a set of 10,000 templates for every instruction. Once the templates database is ready, we run the experiments to collect the corresponding power side-channel traces. As leakage evaluators, we reduce the background noise and improve the signal-to-noise ratio by executing each template multiple times and averaging the side-channel traces: 100 times for Sakura-X and 1000 times for Alveo U200. Even though our work represents an instruction-level leakage analysis, averaging is still a commonly used noise reduction approach even in real attack scenarios: for an attack, the victim code is often executed frequently, allowing averaging, while during training, the attacker can execute templates an arbitrary amount of times [10, 112–114]. Finally, to spread out the impact of environmental noise equally across all instruction classes, we record traces in an interleaved fashion: we record a single trace of each class, in a round-robin order, before continuing the acquisition of the next power trace. Subsequently, we prepare the acquired side-channel traces for the training and inference steps. Similar to previous work [112, 115, 116], we partition the final dataset into a training set (for training the instruction classifier) and a test set, for evaluating the instruction classification accuracy and the leakage learned by the models. The following subsections explain the template generation and the training of the side-channel instruction classifiers in greater detail.

6.3.1 Instruction Template Generation

For our leakage analysis, we create two templating configurations. In the first, denominated as **N**, the target instruction is surrounded by NOP instructions. We use this set of templates to

Table 6.3: RV32I base integer instructions for template generation.

Category	Instructions
Arithmetic	ADD, ADDI, SUB, LUI, AUIPC
Logical	XOR, XORI, OR, ORI, AND, ANDI
Compare	SLT, SLTI, SLTU, SLTIU
Shifts	SLL, SLLI, SRL, SRLI, SRA, SRAI
Loads	LB, LH, LW, LBU, LHU
Stores	SB, SH, SW
Branches	BEQ, BNE, BLT, BGE, BLTU, BGEU
Jump & Link	JAL, JALR

analyze the instruction-level leakage without additional noise from the surrounding instructions. In the second configuration, denominated by **R**, we surround the target instruction with a random instruction before and after. We use the R templating configuration to analyze instruction-level leakage in the presence of other instructions, which represents a more realistic leakage scenario: in practice, the target instruction will be surrounded by a pair of random instructions instead of NOPs.

Algorithm 2: Instruction template generation. N_{NOP} stands for the number of NOPs.

Input : Target instruction, Seed
Output: Target instruction template

```
/* Randomize registers                                          */
```
foreach *All registers x, except x0* **do**
  Initialize with a randomly chosen 32-bit value;

```
/* Insert NOPs                                                  */
```
for $i = 1..N_{\text{NOP}}$ **do**
  `ADDI x0,x0,0`

```
/* Insert random instruction                                    */
```
if *templating* == *R* **then**
  random instruction;

```
/* Insert target instruction                                    */
```
target instruction;

```
/* Insert random instruction                                    */
```
if *templating* == *R* **then**
  random instruction;

```
/* Insert NOPs                                                  */
```
for $i = 1..N_{\text{NOP}}$ **do**
  `ADDI x0,x0,0`

```
/* Insert instruction with an invalid opcode                    */
```
exit;

For both templating configurations, we generate 10,000 templates for every instruction from the RV32I ISA, which are listed in Table 6.3. The process of template generation is detailed in Algorithm 2. The first step is the initialization of x registers with random values. Then, if needed, we insert additional preparation instructions (e.g., to initialize the contents of a memory location for the load instruction). The central and key part of the template contains the target instruction itself: in the case of N templating, similarly to previous work [115, 116], we surround the target instruction with a few NOPs to separate it from the setup phase, while in the case of R templating, we insert a random instruction before and after, making sure the control flow is not altered. Finally, at the end of the template code, we insert an instruction with an invalid opcode, to trigger a signal to the controller that the code execution is completed (see Fig. 6.2).

6.3.2 Instruction Classification Models

Most power side-channel disassemblers in previous work used traditional ML methods and common classification algorithms, e.g., quadratic discriminant analysis (QDA), k-nearest neighbors (k-NN), support vector machine (SVM), Gaussian diffusion model (GDM) [112, 114–117]. However, the accuracy of these algorithm-driven ML classifiers dramatically depends on the preprocessing for dimensionality reduction and feature extraction. Without suitable preprocessing, the noise in the dataset can significantly affect the classification results. For these reasons, previous research relied on the high sampling rate of the oscilloscope to achieve reasonable accuracy. In this work, considering the limited sampling frequency of the on-chip sensors with respect to the soft-core CPU operating frequency, besides testing how well the ML methods proposed in previous work perform in this scenario, we explore leakage analysis using DL-based classifiers.

First, we treat the side-channel instruction classification as a time-series classification problem, as different instructions have unique patterns spread across the time and amplitude domain. Since we use multiple sensors for classification, we represent the trace of each sensor as a separate input channel. Fig. 6.4 shows the classification process.

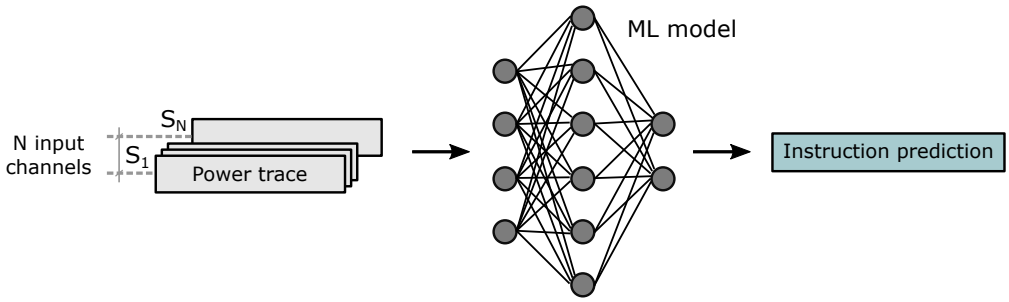


Figure 6.4: Classification process. The power trace of each sensor (S_1 to S_N) is used as one of N input channels. The input is then forwarded to the model, and the instruction prediction is collected for accuracy evaluation.

A class of networks naturally suited to processing sequential data is recurrent neural network (RNNs), specifically long short-term memory (LSTM) models [118]. They have an internal state that can represent context information, and they keep information about past inputs for an amount of time that is not fixed but depends on the weights and the input data. As LSTMs do not perform well when directly extracting features from raw data, they are commonly paired with more complex networks for feature extraction [119, 120], such as convolutional neural networks (CNNs). In practice, feature extraction with CNNs can be applied before or after the LSTM model. Moreover, recent work showed that 1D-CNNs consisting of single-dimensional convolutional layers achieved good results in time-series classification [121]. Finally, CNNs structured as residual networks (ResNets) have shown to be very performant in time-series classification, achieving high accuracy across a range of datasets [121]. Therefore, we train and compare the following models: LSTM, a small 1D-CNN, a large 1D-CNN, and the combination of LSTM and 1D-CNN (LSTM followed by 1D-CNN and LSTM preceded by 1D-CNN), a multi-layer perceptron (MLP), and a time-series ResNet [121].

6.4 Evaluation on Sakura-X

In this section, we provide a detailed instruction-level leakage analysis on Sakura-X. The first step in experimental evaluation is deciding the hypothetical attacker and victim's placement. Given the power delivery network imperfections and knowing that side-channel leakage picked up by the sensors varies with both the absolute and the relative positions of the victim

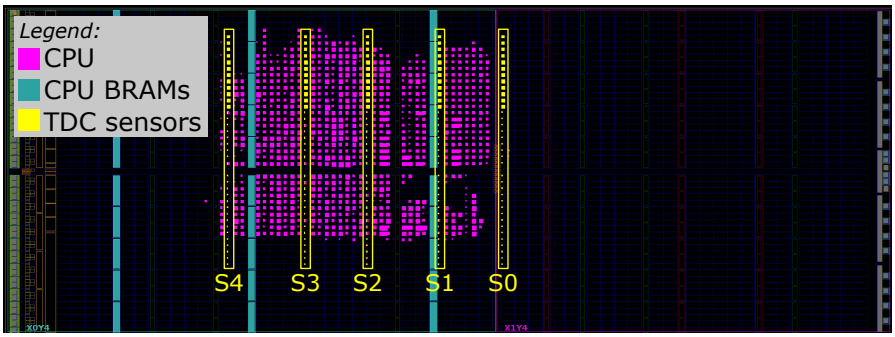


Figure 6.5: Sensor delay lines (in yellow) and CPU (in purple) in Exp-IN.

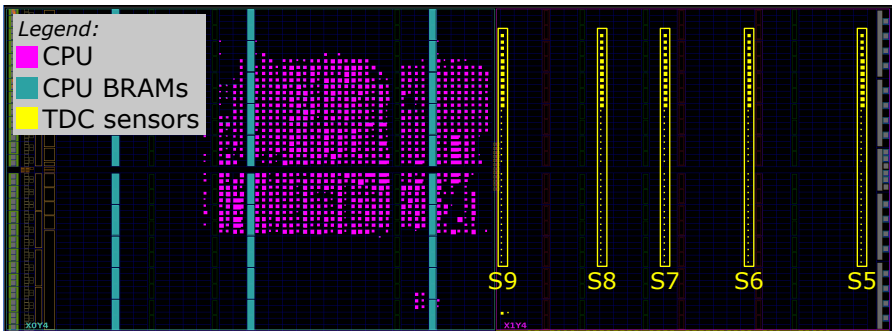


Figure 6.6: Sensor delay lines (in yellow) and CPU (in purple) in Exp-OUT1.

and the attacker [111], we opt to assign the victim to an arbitrary FPGA region and vary the sensor placement.

Figs. 6.5, 6.6, and 6.7 zoom in on the FPGA floorplan containing three different placements of the target CPU and the sensors. In the floorplan in Fig. 6.5, we place the sensors inside the region occupied by the target CPU, in the top-left clock region of the Kintex-7 FPGA (X0Y4). Even though this floorplan does not conform to the standard shared FPGA threat model—where the FPGA regions assigned to the tenants do not overlap—we use it as a worst-case leakage scenario for the evaluator (best-case scenario for the attacker). In the floorplan in Fig. 6.6, we place our five sensors to the right of the target—in the top-right clock region (X1Y4)—in the space between the CPU and the edge of the FPGA, simulating an attacker that spreads out the available sensors across their entire FPGA region. In the floorplan of Fig. 6.7, we move the target CPU one clock region down (X0Y3), further away from the sensors. In the remainder of this section, we will refer to the described floorplans as Exp-IN, EXP-OUT1, and Exp-OUT2, respectively.

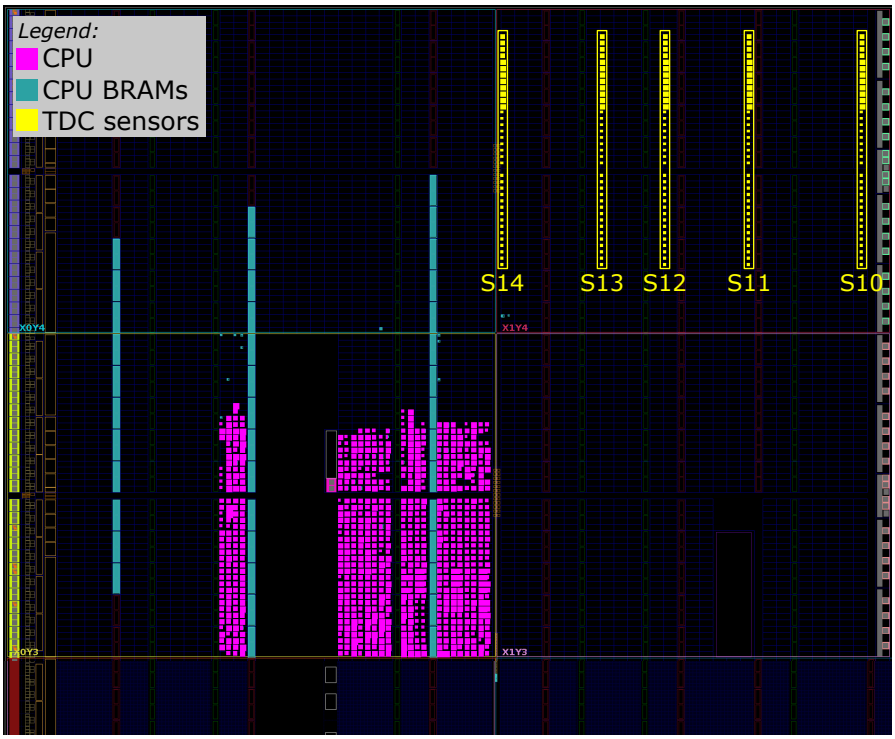


Figure 6.7: Sensor delay lines (in yellow) and CPU (in purple) in Exp-OUT2.

Using the leakage evaluation setup and following the instruction classification method described in Sections 6.2 and 6.3, we create 10,000 templates per instruction (for both N and R template types) and collect the corresponding power side-channel traces, creating four datasets: Exp-IN-N, Exp-OUT1-N, Exp-OUT1-R, and Exp-OUT2-N. We use the Exp-IN-N dataset to evaluate the worst-case leakage (i.e., with physical separation between the victim and the adversary violated and no noise of surrounding instructions). Exp-OUT1-N and Exp-OUT2-N are collected in addition to Exp-IN-N to evaluate the impact of CPU and sensor placement on the instruction-level leakage and model accuracy. Finally, we use Exp-OUT1-R to evaluate the most realistic scenario, where the templates contain the noise of the surrounding instructions. With these four datasets, we cover the three main goals of our instruction-level leakage evaluation: worst-case for the evaluator (Exp-IN-N), the impact of CPU and sensor placement on the accuracy (Exp-IN-N, Exp-OUT1-N, Exp-OUT2-N), and a realistic case for the attacker (Exp-OUT1-R).

We set the sensor trace length to $T = 60$ samples (i.e., 60 consecutive readings of the TDC

output register), to guarantee that all the execution cycles of the instructions in Table 6.3 are captured. To ensure we are capturing the correct instruction execution, we align the start of all instructions to the same sample in the traces (fourth sample): we center the traces around the correct instruction using the recorded CPU opcode.

In our experimental evaluation, we first visually analyze the recorded power traces for multiple sensor placements. We show that instructions of different types show limited visual leakage patterns, while the instructions of the same type do not display any differences. To determine the limits of the instruction-level leakage, we train a range of DL models on the four datasets and show how the accuracy changes depending on the placement and template type. We also use ML techniques to further evaluate the inter- and intra-type instruction leakage and show that most of the classification confusion comes from two or three instructions with similar leakage. We show that preprocessing techniques and ML approaches used in previous work are outperformed by DL techniques. Finally, we evaluate the limits of the instruction-level leakage by investigating the impact of the number of sensors, averaging, and the dataset size on the accuracy.

6.4.1 Visual Analysis of Sensor Traces

Before analyzing leakage using the DL-based classification methodology described in Section 6.3.2, we first visually analyze the recorded sensor traces. In our first experiment, we investigate how sensor placement impacts the waveforms and the leakage in the traces. Fig. 6.8 shows the average trace of all templates of arithmetic and logical instructions across all 15 sensors (five in each of the three floorplans) for the Exp-IN-N, Exp-OUT1-N, and Exp-OUT2-N datasets. We can observe that the sensor placement significantly impacts the shape of the traces, including the peak-to-peak ratio: S10, the furthest from the CPU, has a peak-to-peak ratio of less than one, while S3 has a peak-to-peak ratio of almost six. For some sensors (e.g., S4, S8, S7, S12, and S13), we observe peaks every four samples, perfectly synchronized with the CPU clock. For some other sensors (e.g., S1, S2, and S4), we observe a different pattern: slight dips every 20 sensor samples (around samples 12, 32, and 52), corresponding to five

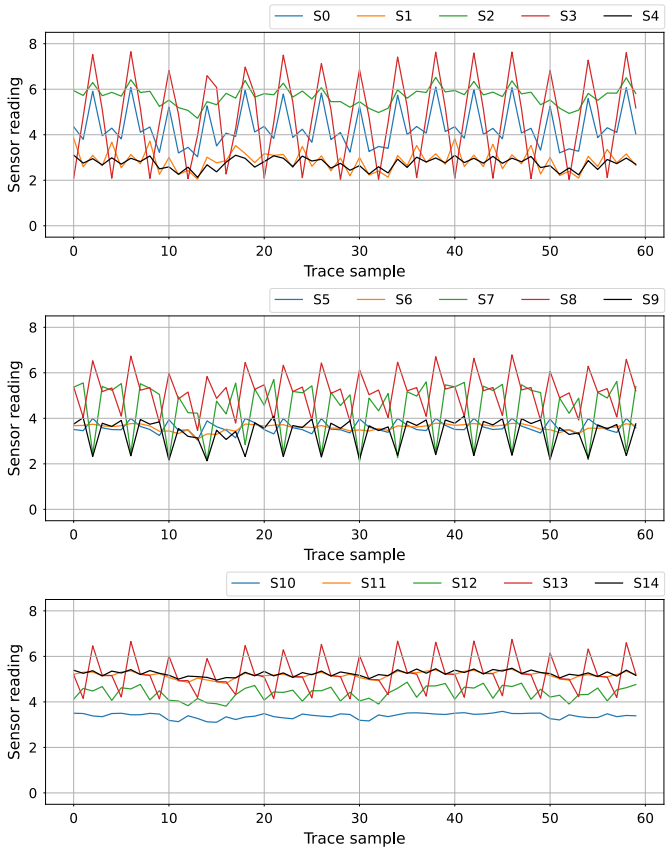


Figure 6.8: Average sensor traces for the arithmetic and logical instructions in Table 6.3.

CPU clock cycles, i.e., to the fetch of the next instruction. This experiment already shows the benefit of having multiple sensors for increasing the power side-channel leakage.

In our next experiment, we visually inspect the inter-type instruction leakage, i.e., how different instruction types impact the shape of the recorded side-channel traces. Fig. 6.9 shows the average traces of sensor S9 (Exp-OUT1-N) for the six instruction groups in Table 6.3. We chose sensor S9, as the plots in Fig. 6.9 were most visually distinguishable for this particular sensor and it represents the worst-case scenario for an evaluator. The peak in sample 48 makes the load and store instructions clearly distinguishable from other groups. Branches and jumps also contain a distinguishable peak centered around sample 28, surrounded by dips on both sides. This experiment shows that, after significant averaging, some distinct visual traits can be attributed to specific instruction groups. However, not all instruction groups can be identified visually. For example, just like loads and stores, jumps and branches have very similar power

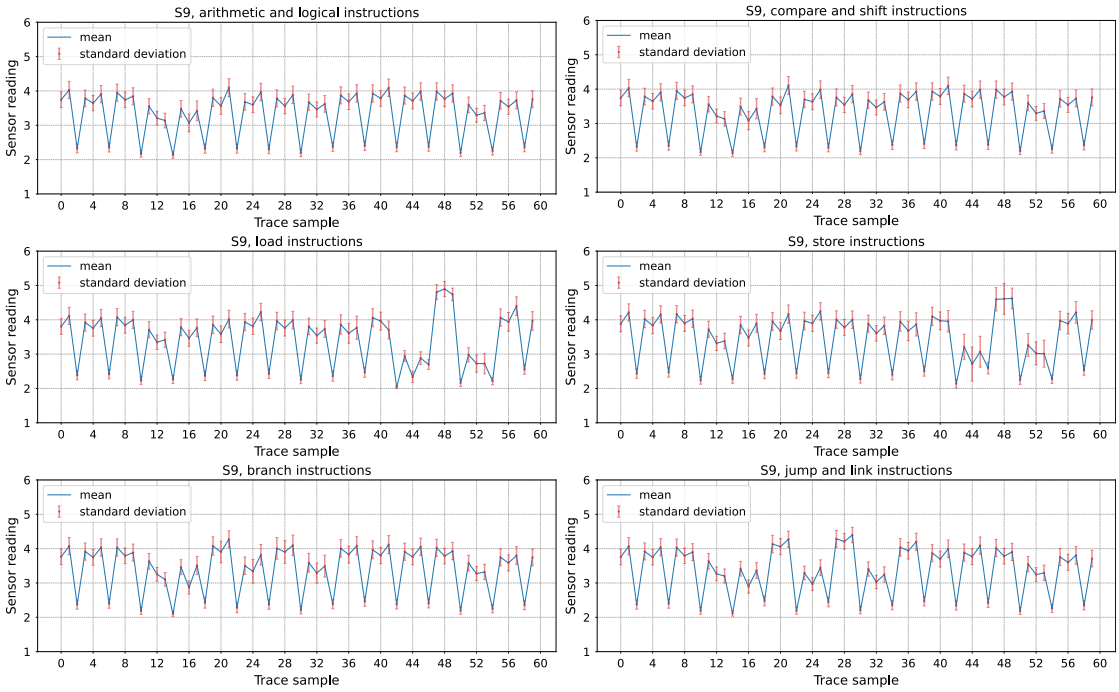


Figure 6.9: Average traces of sensor S9 for all instruction groups in Table 6.3.

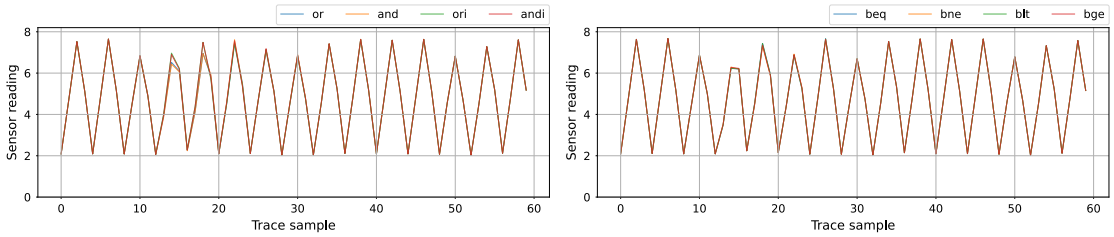


Figure 6.10: Average S3 traces for OR, AND, ORI, and ANDI (left) compared to S3 traces for BEQ, BNE, BLT, and BGE (right).

consumption traces, and it is difficult to tell the exact instruction type from visual analysis alone.

As the final visual experiment, we compare the side-channel traces of several instructions of the same instruction type. Fig. 6.10 shows average sensor S3 traces for eight instructions. We chose S3 because it is in the heart of the soft-core CPU (Fig. 6.5), and it shows, when averaged, the biggest visual differences between instructions of the same type. On the left, we overlap the average traces for OR, AND, ORI, and ANDI. The differences, located between samples 10 and 20, are difficult to notice even with averaging across all templates, as all four instructions

use the same datapath. On the right, we overlap the average traces of four branch instructions: BEQ, BNE, BLT, and BGE. We practically see no difference between these instructions and cannot distinguish them visually. Therefore, even though the visual classification of instructions is possible for some victims (e.g., sizeable ML-based accelerators [15]), soft-core CPUs require more advanced methods for instruction-level leakage analysis.

6.4.2 Deep Learning-Based Instruction Leakage Evaluation

After showing that visual analysis is insufficient to identify CPU instructions executing on remote FPGAs, we deploy advanced DL techniques. We obtain our four datasets by collecting all the sensor traces for each instruction in Table 6.3, as described in Section 6.3.2. Each data point, corresponding to one instruction template, is represented as a matrix with five rows, where each row, i.e., the input channel, is the trace of one of the five sensors. Using the newly created dataset, we first train our deep learning models from Section 6.3.2 using 10-fold validation and compare the resulting accuracy. Then, we compare the results of our DL models with the classical ML methods previously proposed for side-channel disassembly attacks, and we evaluate if frequency-based preprocessing methods, shown promising in previous work [116], have any impact on the extracted leakage. Furthermore, we evaluate how the number of sensors used in the attack impacts the final accuracy. Finally, we evaluate how the amount of averaging or a smaller dataset size can impact the leakage, i.e., the best model accuracy.

To train our deep learning models, we set the number of epochs and the batch size to 100 and 64, respectively. We use the Adam optimizer with an initial learning rate of 0.0001 and the loss to monitor and adjust the learning rate. Table 6.4 summarizes the model details. To facilitate reproducibility, we choose deep learning models with standardized parameters and openly available implementations [121, 122].

Table 6.5 lists the average test accuracy obtained with the four datasets, with the highest accuracy in bold. We can observe that overall, ResNet and 1D-CNN2, the two most complex

Table 6.4: Architecture details of the deep learning models.

Model	Architecture
MLP	Dense(X units, ReLU) | X = (250, 350, 150, 50) Dropout(0.2) Dense(100, ReLU) + Dense(37, Softmax)
1D-CNN1	Conv1D(X filters, kernel size of Y) + MaxPool(2) | (X,Y) = ((64,10), (64, 4)) Dropout(0.2) Dense(100 units, ReLU) + Dense(37, Softmax)
1D-CNN2	Conv1D(X filters, kernel size of Y) + MaxPool(2) | (X,Y) = ((32,12), (45, 10), (64,8), (128,4)) Dropout(0.2) Dense(100 units, ReLU) + Dense(37, Softmax)
LSTM	LSTM(100 units) Dropout(0.2) Dense(100 units, ReLU) + Dense(37, Softmax)
1D-CNN & LSTM	Conv1D(64 filters, kernel size of 4, ReLU) Conv1D(64 filters, kernel size of 4, leakyReLU=0.3) Dropout(0.2) MaxPool(2) LSTM(100 units) Dense(100 units, leakyReLU=0.3) + Dense(37, Softmax)
LSTM & 1D-CNN	LSTM(100 units) Conv1D(64 filters, kernel size of 2, leakyReLU=0.3) + MaxPool(2) Dropout(0.2) Dense(100 units, leakyReLU=0.3) + Dense(37, Softmax)
ResNet	Standard time-series Resnet: 3 blocks with 3×Conv1D layers and residual connections [121]

DL models, achieve the highest accuracy for all datasets. Results in Table 6.5 also show that models without convolutional layers do not manage to extract leakage well and result in low classification accuracy. Moreover, the accuracy drops as the sensors are placed further away from the target CPU. For example, for the best model (ResNet), Exp-OUT1-N has a 16.07% lower accuracy than Exp-IN-N, and Exp-OUT2-N has a 22.5% lower accuracy than Exp-OUT1-N. Therefore, an evaluator testing local CPU leakage with sensors placed inside the CPU will achieve an overestimation of the leakage—if leakage does not exist in a scenario such as Exp-IN-N, an evaluator can, with a high probability, guarantee that a potential attacker will not be able to exploit the leakage. Finally, Table 6.5 also shows the contrast in accuracy resulting from differences between isolated instructions (Exp-OUT1-N) and instructions with random instructions surrounding them (which is the case in a code sequence execution). We can observe that the drop in SNR caused by additional instructions results in a 10.69% accuracy drop for the ResNet model.

Table 6.5: Instruction classification accuracies (in %) for the deep learning methods. The highest accuracies, in bold, are obtained using the 1D-CNN2 and ResNet models.

Dataset	Average Accuracy (%)						
	MLP	1D-CNN1	1D-CNN2	LSTM	1D-CNN & LSTM	LSTM & 1D-CNN	ResNet
Exp-IN-N	79.24	82.38	84.91	77.21	81.53	84.04	**86.46**
Exp-OUT1-N	61.60	65.15	69.58	61.63	65.07	68.45	**70.39**
Exp-OUT1-R	52.89	56.79	59.10	50.58	55.59	58.44	**59.71**
Exp-OUT2-N	43.29	46.11	**48.03**	42.10	44.70	46.84	47.89

For the best model, i.e., ResNet, we explored different hyperparameters with the goal of increasing the accuracy of Exp-OUT1-R, a realistic dataset in terms of an attack. Increasing the initial learning rate and the number of epochs did not result in higher accuracy. An increase of the batch size to 128 or a reduction to 32 did not significantly change the classification accuracy. Increasing the number of ResNet blocks, increasing the number of layers per block, or adding an LSTM layer at the input did not significantly change the ResNet classification accuracy either.

To analyze further the instruction-level leakages and understand why the classification accuracy for Exp-OUT1-R does not reach a number significantly higher than 60%, we evaluate the top-K accuracy of our best model for all four datasets. Unlike the regular model accuracy, i.e., top-1 accuracy, the top-K accuracy labels a prediction as correct if the real class is among the top K predicted classes (ranked by predicted scores). If the top-K accuracy is high while the top-1 accuracy is low, this signifies that groups of classes are often confused. Fig. 6.11 shows the top-K accuracy for ResNet (using 10-fold validation), for all four datasets, and K ranging from one to six. We can observe that, for all datasets, the trend is the same, and the accuracy

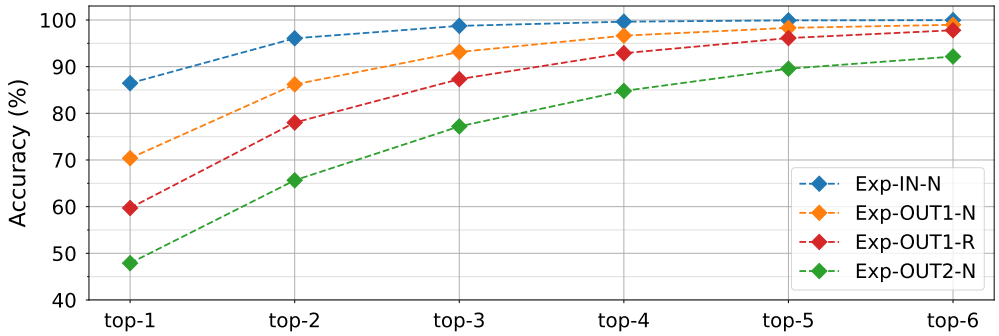


Figure 6.11: Top-K accuracy (K = 1, 2, 3, 4, 5, and 6) using ResNet, for all four datasets.

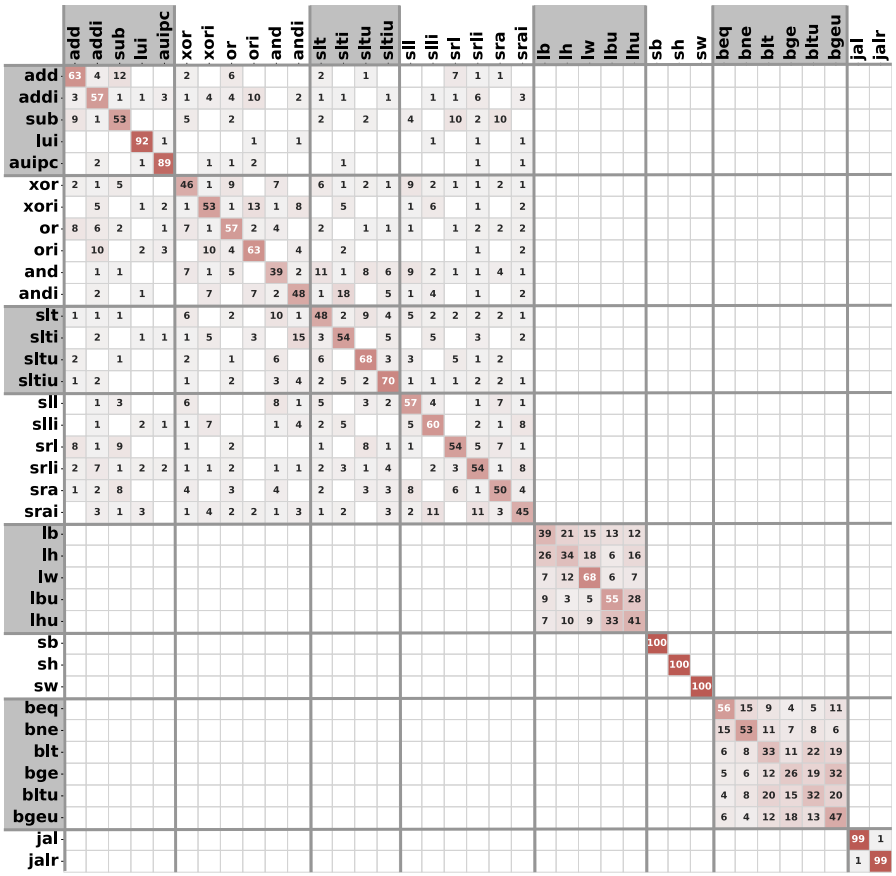


Figure 6.12: Normalized confusion matrix (in %, rounded) of the ResNet model (100 epochs), for Exp-OUT1-R.

significantly increases with K. The most significant accuracy increase is observed between top-1 and top-2 accuracy: 15% on average. The difference reduces for every subsequent K increase while the accuracy converges to almost 100% for all datasets except Exp-OUT2-N. This trend shows that the main difficulty for the classification is distinguishing between two or three similar instructions. For Exp-OUT2-N, the sensors are far away from the soft-core CPU and record a limited leakage compared to the other two placements, which is also noticeable in the weaker visual trace properties in Fig. 6.8.

To evaluate which instructions have similar leakages and lower the top-1 accuracy, we look into how well ResNet distinguishes between the instructions in Table 6.3. The corresponding normalized confusion matrix is shown in Fig. 6.12. We see that instructions of a similar type are more challenging to tell apart; for example, different branch instructions. Other

Table 6.6: Classification accuracy of the ResNet model trained for hierarchical classification on the Exp-OUT1-R dataset.

Average Hierarchical Classification Accuracy (%)								
Inter-Type Classification	**Intra-Type Classification**							
83.60	Arithmetic	Logic	Compare	Shift	Load	Store	Branch	Jump
	86.77	72.03	81.35	70.31	49.14	100	39.62	99.20

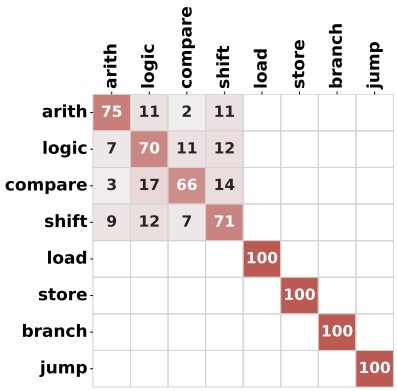


Figure 6.13: Confusion matrix in case of instruction type classification.

examples include arithmetic, shift, and logical operations. The confusion is not surprising, as many instructions share the CPU datapath and, consequently, tend to have very similar power consumption patterns, which makes the classification task harder. However, it is worth noting that the classification is highly successful for instructions of different types, allowing a potential attacker to easily distinguish between the control and data flow of the executed code sequence. This is also confirmed by the 100% top-6 accuracy in Fig. 6.11, and the visual analysis presented in Section 6.4.1.

As our final instruction leakage evaluation experiment, we implement a hierarchical approach to instruction classification. We first perform inter-type classification—training the ResNet model to classify between different instruction types—and then we perform intra-type classification by training a ResNet per each instruction type. Table 6.6 shows the results of inter- and intra-type classification using 10-fold validation on Exp-OUT1-R. We can observe that the inter-type accuracy is significantly (>20%) higher than the ResNet accuracy on the entire dataset, as the model does not need to classify between similar instructions of the same type. Furthermore, we can see that the intra-type classification accuracy heavily correlates with

the confusion shown in Fig. 6.12: instruction types with high intra-type confusion in Fig. 6.12 such as loads and branches, have very low intra-type classification accuracy in Table 6.6.

Finally, Fig. 6.13 shows the confusion matrix of the inter-type classification. Similar to Fig. 6.12, there is no confusion between loads, stores, branches, and jumps. The confusion is limited to arithmetic, logic, shift, and compare instructions, which are ALU instructions and share most of the processor datapath.

6.4.3 Impact of Preprocessing On Instruction Leakage

Previous work showed the importance of preprocessing for increasing the classification accuracy when identifying instructions [116, 117, 123, 124]. Furthermore, frequency-domain analysis, particularly the continuous wavelet transform (CWT), was shown to be well-suited for side-channel disassembly [116]. Therefore, we evaluate our deep learning models with CWT preprocessing to determine if CWT is beneficial for extracting instruction-level leakage in our setting.

From a time-series vector of M sampling points, CWT creates a matrix of $M \times D$ entries, where the D dimensions describe how D frequency components of the time series change over time. Including the original time-series vector in the CWT matrix results in a matrix of $M \times (D+1)$ entries. Knowing that every entry in our original dataset contains five sensor traces, each having T samples, we create the following two additional datasets using CWT with a scale parameter of D. We perform CWT on each sensor trace individually, resulting in five $T \times (D+1)$ matrices for each data point. For the first dataset, *CWT-H*, we concatenate these matrices horizontally in a $5T \times (D+1)$ feature matrix. For the second dataset, *CWT-V*, the matrices are concatenated vertically, resulting in a $T \times 5(D+1)$ feature matrix.

Table 6.7 shows the average 10-fold validation accuracy drop (compared to the baseline datasets in Table 6.5) using DL models with CWT. The scale parameter D for CWT is set to 49 [116]. As can be seen, most DL models do not benefit from the increase of the input space size. For models with high accuracy in Table 6.5 such as ResNet and 1D-CNN2, the overall

Table 6.7: Difference in the instruction classification accuracies when using classical ML approaches, compared to the deep learning methods.

Model	Average Accuracy Increase (%)								
	Exp-IN-N		Exp-OUT1-N		Exp-OUT1-R		Exp-OUT2-N		
	CWT-H	CWT-V	CWT-H	CWT-V	CWT-H	CWT-V	CWT-H	CWT-V	Average
MLP	3.00	3.01	3.87	4.18	1.74	1.63	2.32	2.36	2.76
1D-CNN1	-0.70	-0.27	-1.98	-0.05	-3.67	-1.93	-2.59	-1.04	-1.53
1D-CNN2	-2.40	-1.92	-4.59	-2.72	-4.95	-2.73	-4.39	-2.07	-3.22
LSTM	0.05	3.81	0.54	3.28	0.11	3.42	1.15	2.74	1.88
1D-CNN & LSTM	-13.35	-3.25	-16.05	-3.41	-10.23	-2.23	-5.23	-0.95	-6.84
LSTM & 1D-CNN	-0.56	-0.69	-2.56	-1.57	-3.43	-2.24	-2.78	-1.62	-1.93
ResNet	-3.66	-2.04	-5.19	-2.37	-4.77	-2.25	-3.04	-2.01	-3.17

accuracy drops on average by approximately 2–3%, with a maximum drop being 16.05% for 1D-CNN & LSTM with CWT-H. For models with originally low accuracy in Table 6.5 such as MLP and LSTM, preprocessing slightly increases the accuracy: for approximately 2–3%. We can therefore conclude that with well-fitted models, a DL approach does not require computationally heavy CWT preprocessing, as the models are complex enough to capture the correlation between the traces and the instructions.

6.4.4 Comparison with Classical ML Approaches

Physical power side-channel disassembly attacks relied on high-frequency oscilloscopes and classical ML techniques to achieve high profiling accuracy. However, in our work, we use TDC sensors sampling at 320 MHz, having a significantly lower sampling frequency than oscilloscopes. To evaluate how disassembly techniques used in previous work translate on TDC sensor traces, we obtain the instruction classification accuracy using common ML models (GDM, QDA, k-NN, and SVM) and preprocessing techniques—principal component analysis (PCA) and linear discriminant analysis (LDA)—used in previous work on side-channel disassembly, discussed in Section 10.3. Table 6.8 lists the average 10-fold validation accuracy of classical ML approaches. We obtain the highest accuracy using SVM with PCA and QDA with LDA: 68.74% for Exp-IN-N, 52.45% for Exp-OUT1-N, 47.06% for Exp-OUT1-R, and 37.65% for Exp-OUT2-N, which is 10–20% lower than the accuracy of our best-performing DL-based classifier. Even the deep learning models with lower accuracy (LSTM, 1D-CNN & LSTM) are

Table 6.8: Instruction classification accuracies for the classical machine-learning methods. The highest accuracies, in bold, are obtained when combining SVM with PCA and QDA with LDA.

Method	**Average Accuracy (%)**							
	PCA				LDA			
	GDM	QDA	k-NN	SVM	GDM	QDA	k-NN	SVM
Exp-IN-N	56.55	67.15	39.64	**68.74**	59.78	**67.17**	50.28	65.33
Exp-OUT1-N	45.36	48.56	26.87	**51.83**	48.24	**52.45**	36.15	50.44
Exp-OUT1-R	39.66	41.80	22.02	**47.06**	43.31	**46.47**	30.59	45.96
Exp-OUT2-N	33.74	34.23	26.63	**37.65**	34.82	**37.19**	29.02	37.11

comparable with the best results in Table 6.8. We can, therefore, conclude that advanced techniques, such as deep learning, are required in the shared-FPGA attack scenario, as it involves low resolution and a reduced sampling rate of the voltage sensors coupled with a high victim CPU frequency.

6.4.5 Impact of the Number of Sensors on Instruction Leakage

To investigate the role of the number of sensors in the attack, we analyze the impact of incrementally including additional sensors in the dataset on the classification accuracy. The analysis is performed for all four datasets and on sensor data collected in the setup where all sensors are simultaneously present. Since the sensors record power traces of the same events simultaneously, they are subject to the same experimental conditions (e.g., environmental noise or temperature), facilitating a fair comparison. Furthermore, as the power-intensive measurement logic (memory and controllers) is placed far from the sensors, only the last few elements in their 16-bit delay lines cause differences in sensors' switching activity, which is thus negligible compared to the switching activity of the CPU.

In this experiment, we choose our best-performing model: ResNet. We start by training separate models, one for each sensor, and evaluate the instruction classification accuracy. Table 6.9 summarizes the results. We can observe that the closest sensor does not necessarily have the highest classification accuracy—as sensor S7 has higher accuracy than S5—which is in line with conclusions from previous work [111]. However, we can observe that the further the sensors are from the soft-core CPU, the smaller the accuracy difference between the best

Table 6.9: Average instruction classification accuracies (in %) of ResNet, when trained on the traces of a single sensor only. In bold, the highest accuracies for each of the four datasets.

Average Accuracy (%)														
Exp-IN					Exp-OUT1-: N (top), R (bottom)					Exp-OUT2-N				
S0	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**
48.76	60.22	64.59	**71.52**	53.14	39.32	40.52	**55.50**	45.00	44.34	39.78	41.24	41.79	**42.38**	39.69
					31.17	31.54	**45.86**	36.92	38.16					

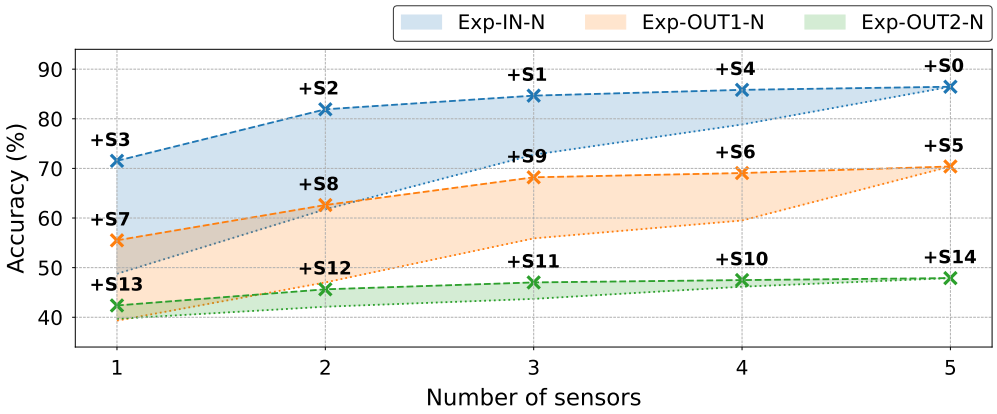


Figure 6.14: Average instruction classification accuracy in the function of the number of sensors contributing to the dataset, for all datasets, with the ResNet model. Upper, dashed lines correspond to including the next best sensor in the dataset. Lower, dotted lines correspond to including the next worst sensor in the dataset.

and the worst sensors: for Exp-IN-N and Exp-OUT1-N the range is 10–20%, while the sensors in Exp-OUT2-N have a maximum difference of 2–3%. This signifies that across multiple sensors, the overall placement does have an impact on the accuracy, also confirmed by results in Table 6.5.

Using data from Table 6.9, we sort the sensors by the obtained accuracy, once in increasing order (from “worst” to “best”) and once in decreasing order (from “best” to “worst”). Fig. 6.14 illustrates the accuracy increase in function of the number and the choice of sensors in the dataset used for training. The dashed (respectively, dotted) lines show the accuracy increase when the next best (respectively, next worst) candidate sensor is added to the dataset. For example, the highest accuracy achieved with a single sensor (71.52% in Fig. 6.14) corresponds to sensor S3 and Exp-IN-N (Table 6.9), while the accuracy obtained after adding the next best candidate (81.91% in Fig. 6.14) corresponds to sensors S3 and S2 used together. The accuracy

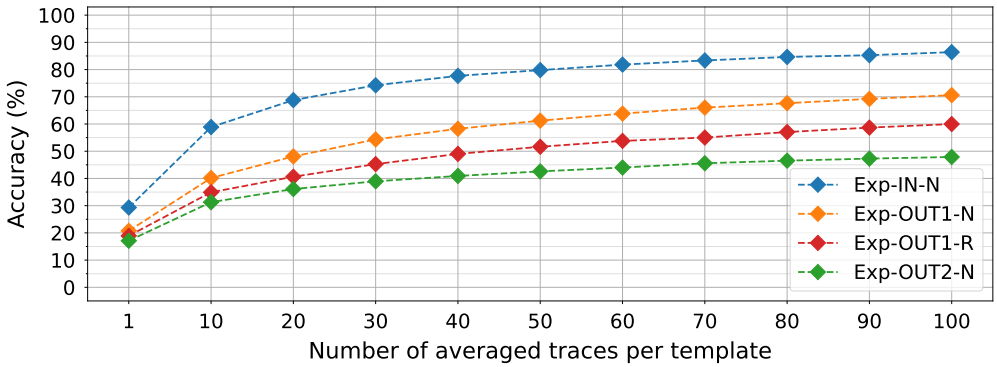


Figure 6.15: Accuracy in the function of the number of averaged traces per template. The dataset size is 10,000 templates per instruction, while the model used for training is ResNet.

increase is more pronounced on the dotted lines, as every new sensor added to the dataset has better individual accuracy than the ones already in the dataset. Comparing the trend of the shaded regions, we see that the sensors in Exp-OUT1-N and Exp-OUT2-N floorplans, being further away from the CPU, pick up less information leakage. However, the distance between the best and worst-case region borders reduces significantly when four or five sensors are used, showing the importance of using multiple sensors for better leakage extraction. Finally, we can observe that the shaded region for Exp-OUT2-N is significantly narrower than for the other two datasets in Fig. 6.14: an effect that arises because the range between the best and the worst sensor for Exp-OUT2-N is significantly smaller than for Exp-IN-N and Exp-OUT1-N.

6.4.6 Impact of Averaging on Instruction Leakage

To evaluate the impact of averaging on leakage and the ability of DL models to extract it, we use the best model, ResNet, and train it on the four datasets while changing the number of traces averaged for each template. Fig. 6.15 shows the results. We can observe that with only a single trace (no averaging), all four datasets have very low accuracy: approximately 30% for Exp-IN-N and 20% for the other three datasets. Note that, in the beginning, as we increase the number of averaged traces, the accuracy increases significantly for all four datasets, showing the benefit of averaging in eliminating noise. This experiment also shows that increasing the averaging does not indefinitely increase the SNR, as the curves in Fig. 6.15 are logarithmic and

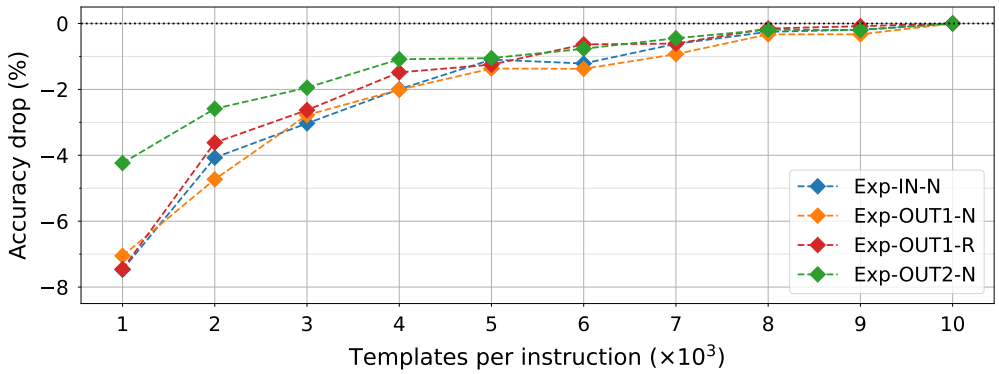


Figure 6.16: Accuracy drop in the function of the dataset size (number of templates) used for training and testing compared to the full dataset with 10,000 templates. Results are shown on the ResNet model for all four datasets.

flatten off after using roughly 80 averaged traces per template.

From a leakage evaluator point of view, we can see the usefulness of averaging for identifying and analyzing instruction-level leakages. As trace acquisition is a time-consuming process, we see that finding a good number of traces for averaging can help reduce the evaluation time. From an attacker's point of view, we can observe that recording only one trace of the victim execution might not be sufficient for extracting secret information: the attacker might have to resort to recording multiple victim executions or folding loops to accommodate averaging for a better SNR.

6.4.7 Impact of the Dataset Size on Instruction Leakage

As our final experiment on the Sakura-X board, we evaluate the impact of the number of templates per instruction, i.e., the dataset size, on the accuracy of the ResNet model. For five different input seeds, we randomly select a subset of the templates for each instruction and train the ResNet model. Fig. 6.16 shows the accuracy drop compared to the full dataset for a range of template sizes, averaged across all five seeds. We can observe that when using only 10% of the dataset size, i.e., 1000 templates per instruction, the accuracy of Exp-IN-N drops only 4%, while it drops approximately 7% for the Exp-OUT datasets. This indicates that increasing the accuracy is a very difficult problem, as the initial accuracy comes from

the inter-type classification, and the intra-type confusion cannot be significantly improved even by increasing the dataset size by 10×. Fig. 6.16 also shows that after some point, as with averaging, the accuracy does not significantly increase with a bigger dataset size. We can observe that the curves flatten for all four datasets and that using more than 8k templates does not significantly affect the accuracy.

6.5 Evaluation on Alveo U200

This section provides an instruction-level leakage analysis on the Alveo U200 board containing a cloud-scale AMD Virtex Ultrascale+ FPGA. Fig. 6.17 shows the floorplan. At the bottom, we can see the entire FPGA rotated by 90°, with three SLRs, and the shell occupying half of the middle SLR. Fig. 6.17 also shows the enlarged view of SLR2. Similar to the Exp-OUT1 placement in Fig. 6.6, we physically separate the soft-core CPU (PicoRV32 [95]), the sensor region, and the controller. As Alveo U200 contains a much larger FPGA than Sakura-X, we instantiate 29 sensors, as described in Section 6.2. However, unlike the spread-out placements in Exp-OUT1 and Exp-OUT2, we place the 29 sensors along the border of the sensor region, clustered in 6 equidistant groups of five sensors (except the last group with four sensors), simulating an attacker placing all the sensors as close to the victim as possible.

Using the leakage evaluation setup and following the instruction classification method described in Sections 6.2 and 6.3, we create 10,000 templates of N type and 20,000 templates of R type. We collect the corresponding power side-channel traces, creating three datasets: Exp-10k-N, Exp-10k-R, and Exp-20k-R. Since the sensor and the soft-core CPU both work at the same clock frequency, the sensor traces do not need as many samples as for the Sakura-X board: we set the sensor trace length to $T = 16$ samples which guarantees that the longest instruction execution is completely captured. Like in the Sakura-X experiments, we align the start of all instructions to the same sample in the traces using the recorded CPU opcode.

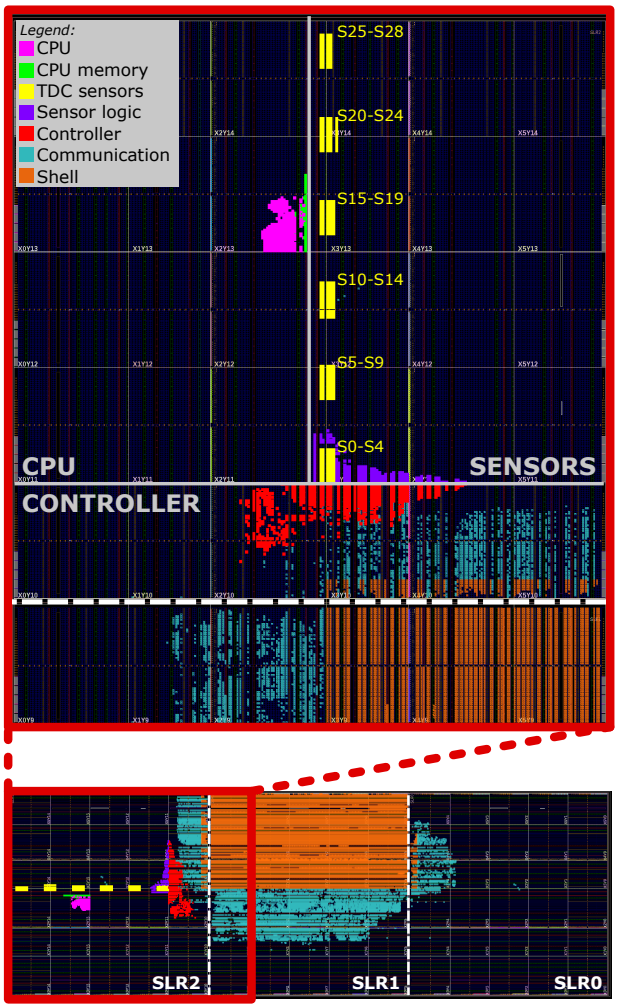


Figure 6.17: Floorplan on the Alveo U200 board. Sensor delay lines (in yellow) and CPU (in purple).

6.5.1 Instruction-Level Leakage on Cloud-Scale FPGAs

In the first experiment on Alveo U200, we evaluate the instruction-level leakage by training the ResNet model on all three datasets. Table 6.10 shows the averaged results for 10-fold validation. We can observe that on a significantly larger FPGA and a CPU running at the same high clock frequency as the sensor, the accuracy for all three datasets is approximately 40%. The Exp-OUT-20k-R dataset has a 20% lower accuracy than the Exp-OUT1-R dataset on Sakura-X, while the Exp-OUT-10k-N dataset has an approximately 30% lower accuracy than the Exp-OUT1-N dataset. Table 6.10 also shows that the difference between N and R datasets

Table 6.10: Instruction classification accuracies on Alveo U200 (in %) for the ResNet model.

Model	Average Accuracy (%)		
	Exp-OUT-10k-N	**Exp-OUT-10k-R**	**Exp-OUT-20k-R**
ResNet	42.59	37.60	39.05

is much smaller on Alveo U200 (≈5%) compared to Sakura-X (≈10%). As the target used on Alveo is a multicycle CPU (PicoRV32 [95]) and the target used on Sakura-X is a pipelined CPU (RISC-Y [102]), the impact of surrounding instructions is smaller on a multicycle CPU. Finally, Table 6.10 confirms the findings shown in Section 6.4.7, as the accuracy difference when using 10,000 templates (Exp-OUT-10k-R) and 20,000 templates (Exp-OUT-20k-R) is less than 2%: after a certain threshold, increasing the templates does not significantly impact the model's ability to extract leakage.

6.5.2 Code Sequence Classification

As our final experiment, we evaluate the leakage of code sequences instead of single instructions. To do this, we train a classifier to predict which sequence was executed from a set of known code sequences. We create eight code sequences, each comprised of multiple instructions of the same type, separated by instructions of another type. Each code sequence has a primary type from Table 6.3: load, store, branch, arithmetic, logic, compare, and shift. For example, the load code sequence consists of load instructions separated by shift instructions, and the store code sequence contains store instructions separated by logic instructions. The number of instructions is tailored so that all eight sequences have the same execution length of 40 clock cycles. This structure makes them representative of short code sequences dominated by same-type instructions, commonly found in open-source code.

The code sequence templates have the same structure as instruction templates of type N in Algorithm 2, where instead of a single instruction, the target is a fixed set of instructions for the given sequence. For each code sequence, we create 10,000 templates, each representing an execution of the code sequence on random data. For this dataset, called Exp-10k-S, the traces for each template are recorded as an average across 1,000 executions of the same code

Table 6.11: Code sequence classification accuracies (in %) for the deep learning methods. The highest accuracies, in bold, are obtained using the 1D-CNN1, 1D-CNN2, LSTM & 1D-CNN, and ResNet models.

Average Accuracy (%)						
MLP	**1D-CNN1**	**1D-CNN2**	**LSTM**	**1D-CNN & LSTM**	**LSTM & 1D-CNN**	**ResNet**
99.45	**100**	**100**	26.82	82.63	**100**	**100**

sequence and input data.

Table 6.11 shows the average accuracy across five different seeds for all the DL methods in Section 6.3. We can observe that unlike instruction-level leakages, which contain the randomness of the operands and data as noise, code sequences emit significantly higher leakage, as almost all the models achieve high accuracy: 1D-CNN1, 1D-CNN2, LSTM & 1D-CNN, and ResNet achieve the accuracy of 100%. The only model with a noticeably low accuracy is LSTM since it fails to converge for four out of five seeds, while for the remaining seed, it achieves 80% accuracy.

From an evaluator's point of view, this experiment shows that it is important to evaluate not only instruction-level leakages but also the deployed code in its entirety. Moreover, since short code sequences dominated by same-type instructions are common in open-source code, our results demonstrate that known, i.e., open-source code sequences can be profiled and more easily distinguished than single instructions. For example, in an AES algorithm, the attacker can use a load-intensive piece of code for profiling and easily differentiate it from a branch-intensive code sequence in a control-flow algorithm. To avoid potential exploits, users should deploy countermeasures or use proprietary (unknown) code. From an attacker's point of view, these results show that attacking code sequences instead of individual instructions requires less attack effort for a potentially higher benefit.

6.6 Discussion

In Sections 6.4 and 6.5, we have seen the evaluation of instruction-level leakages on two FPGA boards. Unlike large ML accelerators—which require recording long execution traces

and have significant architecture- and data-dependent power variations—the instructions of soft-core CPUs have very short execution traces: in the range of tens of microseconds. Consequently, our results show that soft-core CPUs do not have visible leakage in power traces that SPA can exploit; unless extensive averaging of a million traces is performed. Through visual analysis of averaged traces, we have observed that instructions of different types are more likely to have different leakages. In contrast, instructions of the same type have almost no differences despite averaging. These results are also confirmed by DL models, as both the confusion matrix and hierarchical classification indicate that the classification confusion is concentrated within instruction groups, not between them. Furthermore, a significantly higher top-K accuracy also demonstrates that the confusion between only a few instructions prevents the models from achieving 100% accuracy.

Our analysis demonstrates that for the evaluator's worst-case scenario, i.e., an attacker breaching the physical separation barrier, the highest achieved accuracy is 86.46% using the time-series ResNet model [121]. We show that classical ML approaches used in previous side-channel disassembly work do not transfer well to the shared FPGA scenario. As no high-end oscilloscope equipment is available, using ML and preprocessing approaches from previous work on sensor traces results in a 10–20% lower accuracy than DL approaches.

Our results indicate that the templating impacts SNR and the model accuracy, where Exp-OUT1-N has an approximately 10% higher accuracy than the Exp-OUT1-R dataset. These results suggest that the evaluator should use N templating for the worst-case estimate, while for a more realistic estimate, they should use R templating.

Throughout our experimental evaluation, we show that placement does matter: the overall distance between the sensors and the CPU impacts the SNR and, thus, the accuracy. We demonstrate that increasing the distance between the CPU and sensors (Exp-IN, Exp-OUT1, Exp-OUT2) incurs an approximately 15% accuracy drop. For one sensor placement, our results, like previous work [111], also indicate that sensors closer to the CPU do not necessarily have the highest accuracy, possibly due to the imperfections of the PDN implementation and

different sensor calibrations. Additionally, we show the benefit of using multiple sensors: the accuracy increases significantly (≈15% for Exp-IN-N) when using five sensors instead of only one.

We analyze the impact of averaging on the ResNet model accuracy. We demonstrate that no averaging, i.e., only one trace recording per template, results in very low accuracy due to noise. From the point of an evaluator, averaging increases SNR, which reflects in our results, showing that averaging 80 traces can significantly increase the accuracy; however, averaging more traces brings limited to no further benefit. This means that an evaluator can use averaging for a worst-case leakage analysis, to identify weak points of their soft-core CPU design, while also knowing an attacker would need to deploy additional techniques, such as loop folding, to be able to average a single code execution trace. We find that increasing the dataset size does not significantly impact the accuracy: having 10× more templates increases the accuracy at most 8% (for Exp-OUT datasets), showing again that inter-type instruction classification is a relatively easy classification problem, achieving a specific accuracy with only 1,000 templates per instruction. In contrast, distinguishing between instructions of the same type is a hard classification problem, where even 10× more templates are insufficient for increasing the accuracy significantly.

Our results show that cloud-scale FPGAs exhibit less leakage due to their size and PDN structure. Consequently, the accuracy on both N and R template types is approximately 40%, significantly lower than on Sakura-X. However, unlike instructions, we demonstrate that short code sequences have significant leakage and that DL models can predict them with an accuracy of 100%.

Finally, our experimental analysis shows that to ensure no exploitable leakage, the evaluator should always test the worst-case scenario: multiple sensors with no physical separation, using N-type templates on a smaller FPGA with higher SNR, and averaging. In this case, the evaluator will either ensure there is no leakage or, if there is, they will be able to analyze it more efficiently and design appropriate mitigations.

6.7 Countermeasures

Countermeasures against power side-channel analysis have been extensively studied, and they fall into two main categories: *hiding* and *masking* [10]. Hiding aims to reduce the SNR of the signal recorded by the attacker. Therefore, protections can either focus on reducing the leakage signal, e.g., by equalizing the data-dependent power consumption [125], or increasing the side channel noise. Because most attacks depend on aligned traces, hiding can also be done in the time dimension, by adding random delays or clock jitters during the hardware execution. Masking, on the other hand, requires processing algorithmically-randomized data, while maintaining the correctness of the circuit operation [88]. Both hiding and masking, however, suffer from considerable area overhead and vulnerability to higher-order attacks [10].

Mitigations for power side-channel disassembly attacks involve restructuring the code or redesigning the hardware to reduce leakage [7]. De Mulder et al. integrated defenses into the microarchitecture of a soft-core RISC-V processor and tested them on a Zynq FPGA [126]. They enhanced the side-channel security by protecting memory accesses and introducing masking in the CPU. Another example of a side-channel protected microprocessor is PARAM, developed and tested on a Sakura-X FPGA [127]. After analyzing the RTL and leakage of an open-source RISC-V processor, the authors used obfuscation to reduce datapath leakage and to conceal the addresses sent to the cache. Alternative (or complementary to) hardware changes are software defenses: random code injection, code obfuscation [128], or shuffling the instruction execution [129] are most used to protect proprietary code against side-channel disassembly attacks.

On shared FPGAs, protections against side-channel analysis commonly deploy different hiding techniques, better tenant isolation, or methods that prevent the deployment of sensor circuits. As hiding techniques, the works of Le Masle et al. [58] and Krautter et al. [59] are most relevant. Le Masle et al. designed a network of on-chip RO-based sensors to control power wasters and maintain a constant power consumption, thus reducing the SNR [58]. They used a proportional-integral-derivative (PID) controller as the control circuit, while power wasters

were implemented using long routing wires (equivalent to high capacitive load). Similarly, Krautter et al. designed an active *fence* composed of ring oscillators placed between two neighboring FPGA tenants [59]. The actuator controlling the fence in a closed-loop control system was a TDC sensor, and the fence area overhead was 100% compared to the unprotected design.

Additionally, Güneysu and Moradi proposed a set of countermeasures on FPGAs [130]. Using BRAM write collisions, short circuits, and shift register LUTs, they implemented Gaussian noise to reduce the SNR. Sasdrich et al. improve the resistance against FPGA side-channel attacks by dynamically changing the hardware implementation of a PRESENT cipher at runtime using the FPGA partial reconfiguration [131]. All these countermeasures are independent of the design under protection and can hence be used to increase the side-channel security of soft-core CPUs in a shared FPGA scenario.

The final way of preventing remote power side-channel attacks on shared FPGAs is by detecting and forbidding sensor-like structures in the RTL designs: Krautter et al. [132] and La et al. [133] developed bitstream scanners, which search for *signatures* of potentially malicious circuits. Deploying them on the cloud could prevent remote attackers from recording power traces and thus achieve power side-channel security of soft-core CPUs. However, bitstream scanners are not 100% effective in preventing malicious designs, as researchers have found ways to implement stealthy voltage sensors using benign circuits [134].

6.8 Limitations

In this work, we experiment with two lightweight soft-core CPUs commonly used for embedded bare-metal applications, which support straightforward integration with FPGA logic. Within the spectrum of embedded CPU microarchitectures, we have focused on two prevalent varieties commonly used in previous work on power disassembly attacks: a multicycle and a pipelined CPU. Therefore, the results and conclusions in Sections 6.4 and 6.5 should generalize to a good number of embedded CPU implementations and ISAs.

Our results show that inter-type instruction leakage is the strongest, while it is harder to distinguish instructions of the same type. This result implies that instructions using different hardware and datapath in the CPU, typically instructions of different types, exhibit varied leakage, thus rendering them more distinguishable by machine learning models. For instance, logic and arithmetic instructions solely utilize the ALU, while loads additionally fetch data from memory. Branches use the ALU and modify the program counter, and jumps merely alter the program counter. Conversely, instructions sharing most of the datapath exhibit similar leakage, making them difficult to distinguish. For example, for instructions where only the ALU opcode differs—such as logic instructions—the CPU controller executes the same steps, with only a different ALU operation. Regardless of the microarchitecture and ISA, these observations hold. Some microarchitectures might be single-cycle, some multicycle, and some pipelined. However, the overall impact of the microarchitecture is on the SNR, resulting in the leakage (and classification accuracy) being stronger or weaker, but not impacting our conclusions. For example, pipelined architectures might have a more significant difference between N and R datasets, while multicycle architectures might have a comparably smaller difference (as is the case of RISC-Y and PicoRV32).

Our insights on DL superseding classical ML approaches in cases with low SNR are also general and should not depend on the CPU microarchitecture. Similarly, our conclusions regarding placement, averaging, and dataset size also apply to other soft-core CPU cores. Only in cases with high SNR (e.g., in ML-based processors with more straightforward differentiation between workloads) might our conclusions change: classical and DL methods may display more comparable accuracy if faced with an easy classification problem.

As mentioned earlier, the evaluation presented in this work is limited to embedded soft-core CPUs. Considering more complex processor cores would bring a new set of challenges. Larger CPUs—superscalar, out-of-order, and speculative—entail a higher communication latency, lower operating frequency, and higher area overhead. These factors impact instruction identification accuracy in various ways. On the one hand, a larger area may make the instruction-level leakage stronger. Conversely, the hardware overhead for operating system

support or instruction-level parallelism (out-of-order and speculative execution) could increase the noise and reduce the instruction-level leakage. More complex cores might have a lower maximum operating frequency, allowing more sensor samples per CPU clock cycle (i.e., higher quality measurements), but the out-of-order execution could make synchronizing the power traces more difficult. Evaluating the impact of microarchitectural features of larger soft-core CPUs on the leakage is, therefore, an interesting avenue for future work.

Our work evaluates instruction-level leakage of soft-core CPUs in isolation. Future research could explore the leakage in the context of a complete system consisting of a soft-core CPU and an accelerator. To further justify the need for mitigations, future work could showcase an attack on longer code sequences, e.g., detecting loops in power traces (with no averaging) and then folding loop executions to obtain averaged traces, or profiling longer open-source code sequences to detect specific code execution. Turning to the countermeasures, Section 6.7 outlines a palette of mitigation techniques that could be implemented in many ways. Evaluating their performance and scalability is important and, as such, merits a study on its own.

Last but not least, the instruction-level leakage evaluation methodology presented in this chapter is general and can be used for any CPU microarchitecture to obtain implementation-specific results and conclusions.

6.9 Chapter Summary

This work analyzes the instruction-level leakages of soft-core CPUs in shared FPGAs. We show that, unlike with ML accelerators, potential attackers cannot rely on SPA alone, as even with significant averaging, the visual leakage of small soft-core CPUs is limited. Instead, to analyze the instruction-level leakages, we compute the classification accuracy using instruction profiling templates. We demonstrate that ML methods from previous power disassembly attacks are insufficient for remote leakage analysis and that evaluators should deploy DL methods: they achieve approximately 10–20% higher accuracy when classifying instructions from power templates. Using DL methods and a worst-case scenario for the evaluator—a

breach of physical and logical separation—we achieve a maximum accuracy of 86.42%.

Our analysis demonstrates that as the leakage evaluation scenarios become more realistic for a potential attack, the leakage, and thus the classification accuracy, reduces. Enforcing physical separation and placing the soft-core CPU further away from the on-chip sensors reduces the accuracy significantly, as well as using more realistic templates with the target instruction surrounded by random instructions. Furthermore, we show that most of the instruction-level leakage is constrained to instructions of different types and that the confusion comes from only a few similar instructions: using the top-4 accuracy metric already results in an accuracy above 90% for most of our datasets.

We quantify the impact of averaging on the accuracy and show that the accuracy increases, up to a certain point, as the number of averaged traces increases. We also demonstrate that increasing the number of templates does not significantly increase the accuracy. Furthermore, our analysis shows that a cloud-scale FPGA on the Alveo U200 board has significantly less leakage, as the more prominent and higher quality PDN results in a lower SNR. Finally, we demonstrate that, unlike instruction-level leakages, code sequences exhibit significantly higher leakage and can be classified with an accuracy of 100% even on cloud-scale FPGAs.

Our work can serve as a leakage evaluation methodology for remotely deployed soft-core CPUs. It can also be leveraged for building more advanced power side-channel disassembly attacks.

In conclusion, we demonstrate that even small circuits leak information on shared FPGAs, and that potential attackers can remotely extract that information with a small number of power trace acquisitions. This result highlights the need for deploying appropriate mitigations on soft-core CPUs, in multitenant cloud FPGAs.

7 Active Wire Fences Against Remote Power Analysis Attacks

As we have seen in Chapters 5 and 6, multitenancy on FPGAs presents a unique set of security challenges that cannot be fully addressed with solutions involving physical or logical isolation between tenants [135]. Our work and the related work outlined in Chapter 10, highlight the need for protecting FPGA users in a multitenant setting.

In Chapter 6.7, we discussed possible countermeasures to protect against power SCA attacks on shared FPGAs, specifically regarding attacks on soft-core CPUs. As a universal mitigation technique, we discussed hiding techniques (i.e., reducing the signal-to-noise ratio). To implement hiding on shared FPGAs, Krautter et al. designed an *active fence* [59] consisting of ROs, and placed it between the victim (in their case, an AES module) and the rest of the FPGA. The advantage of active fences is that they are independent of the victim's application and do not require modifications to the victim's design. Krautter et al. showed that the RO fence, of approximately the same size as the victim AES circuit, when activated with a pseudo-random number generator (PRNG), leads to a considerably higher attack effort: the number of traces needed to break a byte of the secret key using CPA increased by approx. 60×.

On FPGAs, noise can be generated in many ways. Here, we present a novel active fence design, offering better area efficiency and power side-channel security than the RO-based fence. We

This chapter is based on the work of a paper published at the 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems [36].

construct wire-based power wasters which are easy to replicate to build an *active wire fence* of arbitrary size. Similarly to previous work, we used a PRNG to activate the fence. To evaluate the effectiveness of the fence against a power side-channel attack, we collected millions of side-channel traces while an AES-128 cryptographic core was performing encryption. Our results show that, without any fence, the entire 128-bit key could be broken with approx. 30 thousand traces. With an RO fence, that number increased to one million traces, on average. Finally, with our novel active wire fence, more than six million traces (an improvement of 6×) were required to recover all the bits of the secret key. These results highlight the importance of developing alternative power wasters for building effective active fences.

In the remainder of this chapter, we first introduce the threat model. Then, in Section 7.2, we present the architecture of our active wire fences. Section 7.3 describes the experimental setup. Section 7.4 presents and discusses the experimental results, while Section 7.5 concludes the chapter.

7.1 Threat Model

Our work adopts the threat model commonly used in the literature on attacks in shared FPGAs [12–14, 59]. For security reasons, the users of the multitenant FPGA are physically and logically separated [89]. We assume an attacker attempting to impact the system's confidentiality by extracting secret information through the power side channel enabled by the shared PDN. In this threat model, the adversary can use a region of the FPGA to instantiate TDC sensors and measure the voltage changes caused by the victim, where the victim performs AES encryption with a secret key. The ciphertext is sent over a public channel accessible to the attacker, allowing them to execute a power analysis attack using the sensor traces and the ciphertext. Finally, the fence is implemented by the cloud service provider; therefore, the power wasters do not have to conform to security checks such as combinational loop detection.

7.2 Active Wire Fence

In this section, we present our design of a wire-based power waster and the architecture of the active wire fence.

7.2.1 Wire-Based Power Waster

Our unit wire-based power waster consists of three components: a driver, a wire, and a sink. The driver acts as a *source*: it creates a high-frequency signal to drive the wire. What we refer to as the *wire* is, in fact, a connected sequence of FPGA routing resources (i.e., local and global wires, and routing multiplexers); when driven by the toggling signal, the wire consumes power and introduces noise. Finally, the *sink* is a termination point of the wire waster needed to prevent the FPGA compilation tool from optimizing it away.

To implement the source, we opt for a high-frequency signal generator: an RO implemented as a two-input NAND gate (occupying one LUT). Other pulse generators available in the literature, such as glitch generators [136], require considerably more resources than one LUT, thus negatively impacting the area overhead of the entire fence. When enabled to generate a high-frequency signal, the sources of the wire fence contribute to the overall noise [49]. In our wire waster, the sink is implemented as a buffer (one LUT). It, too, contributes to the overall power consumption of the wire waster, because its input is driven by a toggling signal. It can be noted that we do not attempt to maximize the contribution of the source or the sink, as we opt for the simple and resource-efficient implementations of the two terminal points of our wire waster. This approach facilitates the comparison with the RO-based fence and allows us to better estimate the impact of wires on the fence performance.

To ensure the use of global routing resources (i.e., wires spanning one or more FPGA slices), we place the sources and sinks sufficiently apart, and constrain them to the same FPGA column. After placing the source and sink of a power waster, we let the FPGA compilation tool complete the routing. To minimize the number of FPGA logic resources used by the fence, we do not

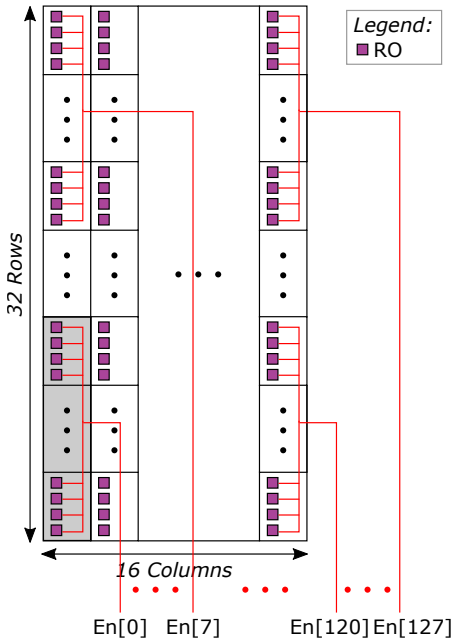


Figure 7.1: RO fence occupying 32×16 FPGA slices; four LUTs are used per slice. Four vertically neighboring slices form a bank. In gray, bank with index zero. Each bank is controlled by a dedicated enable signal.

impose any constraints on the signal routes (e.g., we do not force the routes to pass through LUTs or transparent latches at specific locations [58, 136]).

7.2.2 Building a Wire Fence

Before explaining the implementation of the wire fence, let us look at Fig. 7.1, illustrating a fence built solely from ring oscillators [59]. The ROs are organized in groups referred to as *banks*, which, in our implementation, have two configurable parameters: density (the number of ROs per FPGA slice) and size (the total number of slices occupied by the ROs). Hence, the total number of ROs in the fence equals the number of banks times the size and density. All ROs in the bank are controlled by the same enable signal, whereas each bank can be controlled independently from another. The fence is built by tiling multiple banks. As shown in Fig. 7.1, for the given area budget of the fence, we place the banks in column-major order, connecting one enable signal to each bank.

Our wire fence is shown in Fig. 7.2. The sources (ROs) are placed at the top; the sinks (buffers)

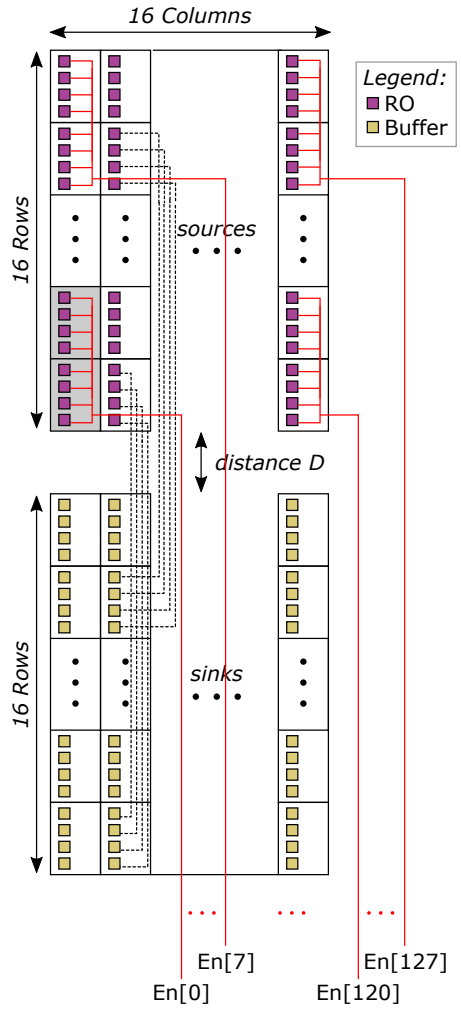


Figure 7.2: Wire fence of 32×16 FPGA slices; four LUTs are used per slice. The top region is occupied by sources (ROs). The bottom region is reserved for sinks (buffers). The distance between the two regions is adjustable. Two vertically-neighboring slices form a bank. Each bank is controlled by a dedicated enable signal.

are at the bottom. The distance parameter D corresponds to the number of unused slices between the *regions* occupied by the sources and the sinks. Similarly to the RO fence, the sources are organized in banks, characterized by their density and size, and placed in column-major order. The sources are enabled the same way as the blocks in the RO-fence (Fig. 7.1), with one enable signal per bank. To ensure the distance between a source and its corresponding sink (in terms of the number of slices vertically between them) is constant, we order the banks of sinks in the same way as their corresponding sources.

In the experimental evaluation (detailed in the next section), similar to Krautter et al. [59], we set the size of the fences to match the resource utilization of the design under protection. In our case, that value amounts to 2048 LUTs. Hence, the wire fence has, in total, 32 rows (16 for the sources and 16 for the sinks) and 16 columns of slices. Four LUTs are used per slice. Sources are grouped in 128 banks, each containing two slices (i.e., eight LUTs) and enabled independently. For a fair comparison, we implement the RO fence with the same resource utilization, resulting in twice as many ROs than in the wire fence. Fig. 7.1 illustrates the RO-fence implementation: 128 banks, each having four slices with four LUTs, organized in 32 rows and 16 columns.

7.3 Experimental Setup

For evaluating the fences, we choose Sakura-X [103], a board specially designed for power SCA evaluation and commonly used in research on attacks in both standalone and multitenant FPGAs [12, 47, 98]. It has two AMD FPGAs: Kintex-7 and Spartan-6. The former, also referred to as the target FPGA, houses the adversary and the victim. The latter, referred to as the control FPGA, aids in reducing unwanted noise by running the communication protocol between the target FPGA and the host machine. For the FPGA design compilation, we use AMD Vivado 2018.3.

Fig. 7.3 gives an overview of the system architecture, consisting of four main components. The open-source AES-128 running at 20 MHz is the victim design requiring protection [87]. The attacker's TDC sensor and FIFO, both running at 200 MHz, serve to record the side-channel traces. The attacker and the victim are physically and logically separated, according to the threat model. The fence is placed between the victim and the adversary. It is directly controlled with a 128-bit PRNG, implemented using a 7-bit Fibonacci linear-feedback shift register (LFSR) [137]. The LFSR generates a pseudorandom value from zero to 127, which is then decoded in hardware, and enables the corresponding number of fence banks. Finally, the controller is in charge of sending the plaintext and receiving the ciphertext, enabling or

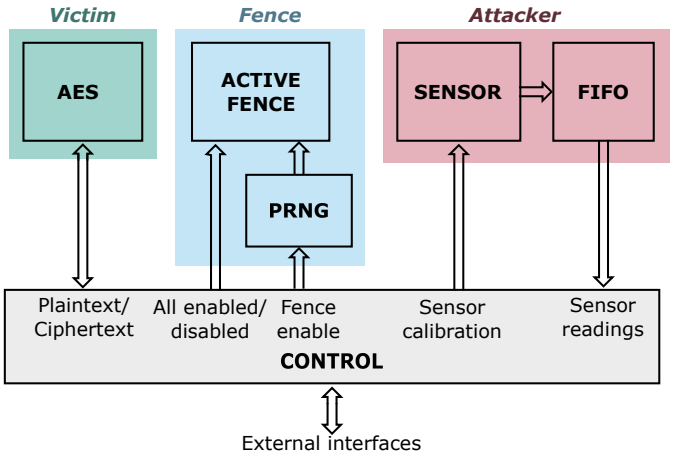


Figure 7.3: System block diagram.

disabling the fence, calibrating the sensor, triggering the trace recording, and offloading the traces to the host machine.

The design floorplan is shown in Fig. 7.4. The attacker and the victim reside in separate FPGA regions. The sensor is placed at the edge of the attacker's region (top right), close to the victim, simulating a worst-case attack scenario. The active fence is located towards the edge of the victim's region (top left). The controller is placed away from the remaining logic, to minimize the noise.

To evaluate the efficiency of our active wire fence, we record sensor traces for three designs: (1) RO fence disabled, (2) RO fence enabled, and (3) wire fence enabled in place of the RO fence (using a different bitstream). For all three designs, we collect traces with a fixed key and chained plaintext: starting with an initial plaintext, we use the resulting ciphertext as the next plaintext. With the obtained traces, we run the CPA attack on the ninth AES round in steps and compute two metrics: the key rank metric of each byte of the 128-bit secret key [70], and the *key rank estimation* metric described in Chapter 3.3 [71].

Using the key ranks obtained for all 16 bytes of the key, we compute the following additional metrics. First, we find the number of bytes broken when using all traces; this metric shows how successful an attack is in terms of broken bytes. Second, with all traces, we find the number of bytes for which the key rank is below or equal to 25, 50, and 100; this metric shows the key rank

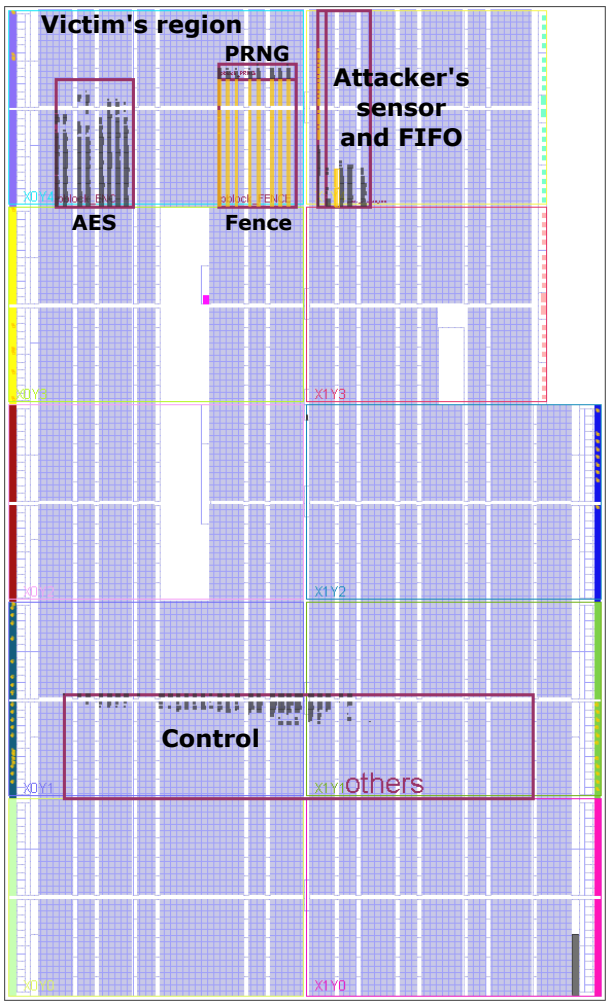


Figure 7.4: Design floorplan, as seen from AMD Vivado.

trends and how many bytes the attacker was close to or managed to break. Finally, we find the number of sensor traces for which the rank of (at least) one of the key bytes drops to zero and stays at zero for at least 10k traces (Metric-10k) and 30k traces (Metric-30k). These last two values will tell us how many traces are required to break at least one byte of the secret key.

7.4 Results and Discussion

In this section, we evaluate our active wire fences. First, we demonstrate the power wasting capabilities of the wire wasters, and then, we show their effectiveness in protecting against remote power side-channel attacks.

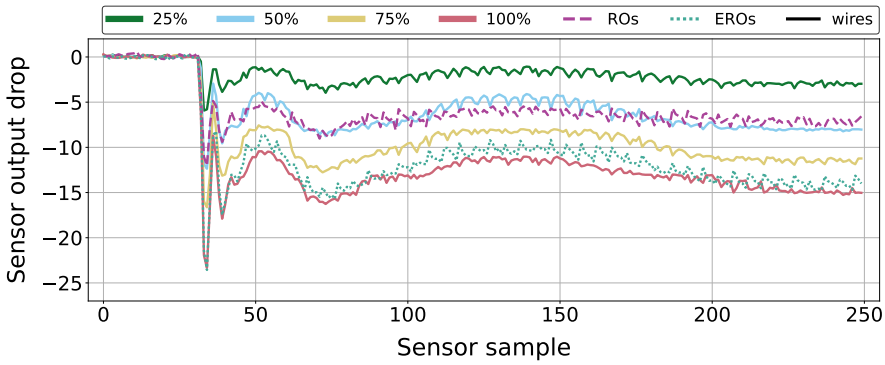


Figure 7.5: Sensor readings drop (i.e., voltage drop) caused by the activation of RO/ERO wasters (dashed lines) compared to wire-based wasters (solid lines).

7.4.1 Voltage Drop Comparison

In our first experiment, we compare the voltage drops caused by the following three types of power wasters: ROs, EROs, and the wire wasters described in Section 7.2. We build the RO fence from Fig. 7.1 (twice: once with ROs, once with EROs) and the wire fence from Fig. 7.2 (setting the distance parameter to zero). At the beginning of the trace recording, we disable the fence. Then, we enable all wasters in the RO (ERO, resp.) fence. In the case of the wire fence, we ran multiple experiments: with 25%, 50%, 75%, and 100% of the wasters enabled (by adjusting the number of active fence columns). Results, visualized in Fig. 7.5, show that, for the same amount of FPGA resources, EROs and wire wasters create significantly more pronounced voltage drops than ROs. In fact, with only 50% of the resources of the RO fence, the wire fence can create a voltage drop similar to that obtained with the fully utilized RO fence. Additionally, we observe that EROs and wire wasters create a similar voltage drop, even though the ERO fence has twice the number of ROs. From this, we can conclude that the long interconnects and the sinks used in our wire fence compensate for the smaller number of ROs.

7.4.2 Varying Distance D

To evaluate the impact of the distance D (between the source and sink regions in Fig. 7.2) on the voltage drop, we vary D from 0 to 150 and record the sensor output once the fence is enabled. In these experiments, the location of the sensor and the sources remains fixed.

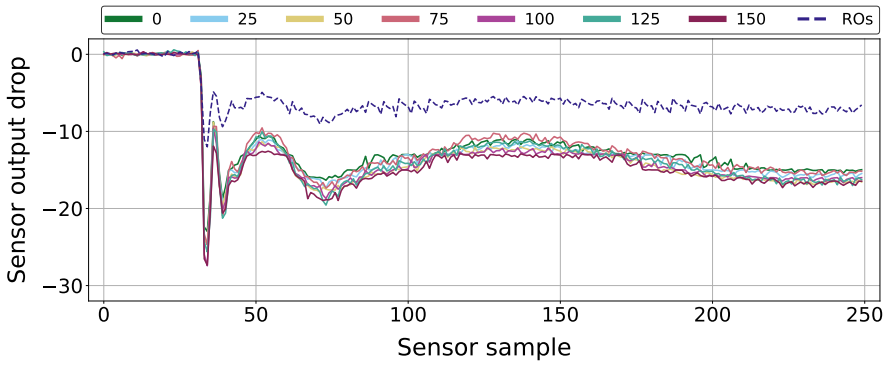


Figure 7.6: Voltage drop for different distances D (in slices) with wire-based wasters (solid lines), compared to ROs (dashed line).

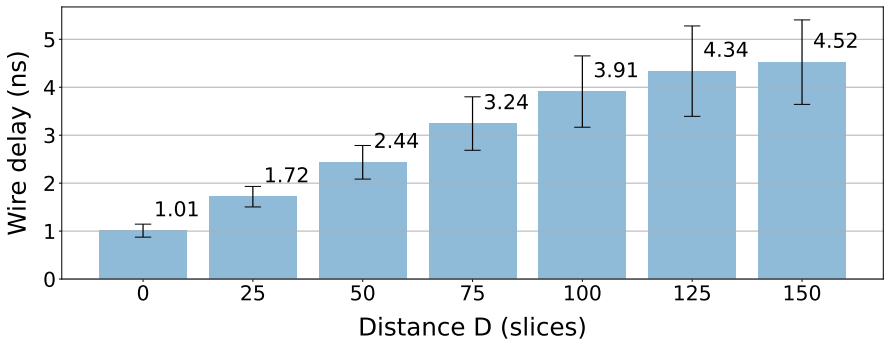


Figure 7.7: Wire delay from source to sink of a wire-based waster extracted using Vivado 2018.3 static timing analysis, in the function of distance parameter D.

Fig. 7.6 shows the results, together with the RO-fence baseline from the previous experiment.

We can observe that as the distance D increases, the overall voltage drop does change, but not as much as one would expect. This result is rather unexpected because, if the distance is significantly larger, the wires connecting the sources and the sinks should intuitively be much longer. Thus, the voltage drop should be proportionally more pronounced. To investigate why this is not the case, we extract the routing delays from the fence sources to the sinks using Vivado static timing analysis, for each considered value of D. Fig. 7.7 shows the average delays and their standard deviations. We observe that the wire length, on average, increases proportionally with the parameter D. However, for the fixed location of the attacker sensor, higher D results in a greater distance between the sensor and the sinks, reducing the effect of the noise generated by a considerable part of the fence (the sinks and long wire segments

close to them). Given the limited impact of increasing D on the voltage drop, and to minimize the area occupied by the fence, we set D to zero in the remaining experiments.

7.4.3 Power Side-Channel Attack Success

In our third and last set of experiments, we evaluate the effectiveness of the wire fence against a power side-channel attack in a multitenant FPGA setting. To that end, we collect power side-channel traces for the following scenarios: (1) RO fence disabled (three runs with 0.5 million traces), (2) RO fence controlled by the PRNG (three runs with 0.5 million and three runs with three million traces), and (3) wire fence controlled by the PRNG (three runs with 0.5 million, three runs with three million, and one run with eight million traces).

Three runs with half a million traces

The results of the CPA attack with 0.5M traces are summarized in Table 7.1. The two central columns of the table show the number of traces needed to break one byte of the secret key. Metric-10k (Metric-30k, resp.) considers the byte broken when its key rank drops to zero and stays at zero for at least 10k traces (30k traces, resp.). The four right-most columns show the total number of key bytes for which the rank is below a certain threshold (100, 50, 25, and 0) at the end of the attack.

Table 7.1: CPA attack results with 0.5M traces.

Setup	Run	Number of traces (10^3)		Bytes with key rank ≤			
		Metric-10k	Metric-30k	0	25	50	100
No fence	1	4	4	16	16	16	16
	2	3	3	16	16	16	16
	3	5	5	16	16	16	16
RO fence	1	89	115	9	14	15	15
	2	81	100	7	11	13	15
	3	96	190	6	15	15	15
Wire fence	1	N/A	N/A	0	0	1	9
	2	449	N/A	0	2	3	4
	3	N/A	N/A	0	0	1	5

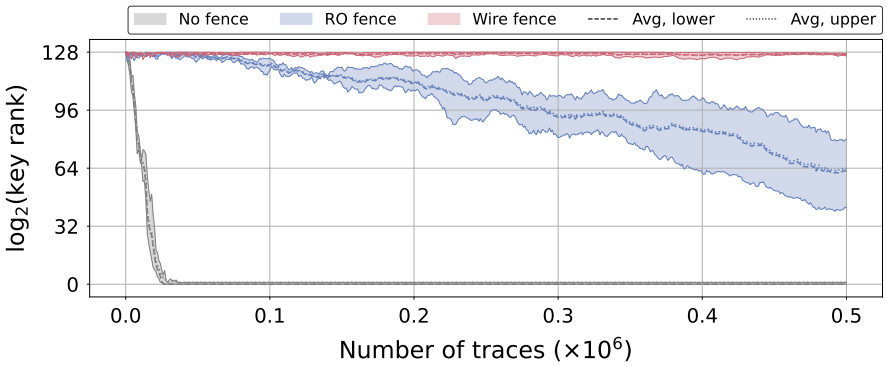


Figure 7.8: Key rank estimation with 0.5M traces.

From Table 7.1, we can observe that without any fence, it takes fewer than 10k traces to break the first byte of the key. Moreover, a CPA attack with 0.5 million traces can break all key bytes. With the RO fence, the number of traces needed to break the first byte according to Metric-10k increases by approximately 20×: 81–96k traces. According to Metric-30k, which requires the key rank to be stable for an even longer period of time, the number of traces to break a key byte increased even further (at least 100k). Between six and nine bytes were broken by the end of the attack. At the same time, all but one key byte had a rank lower than 100, showing that even though not all bytes were broken, their rank decreased considerably. Looking at the wire-fence results at the end of the attack, only four to nine key bytes had rank ≤100.

Next, we compute the *key rank estimation* metric [71]. The results for the three runs are aggregated and shown in Fig. 7.8. Because the key rank estimation is upper and lower bounded, we plot a shaded area indicating the entire range of the key rank estimation (min, max) observed across the runs. Additionally, the dashed and dotted lines correspond to the average lower and upper bounds across the runs. From Fig. 7.8, we see that, without a fence, the key is broken rather quickly: after approx. 30k traces. With the RO fence, the log key rank estimate drops to approx. 63, on average. With the wire fence, the key rank estimation remains very close to its maximum value. These results agree with those listed in Table 7.1 and show that the wire fence achieves higher power side-channel security than the RO fence when attacking with 0.5M traces.

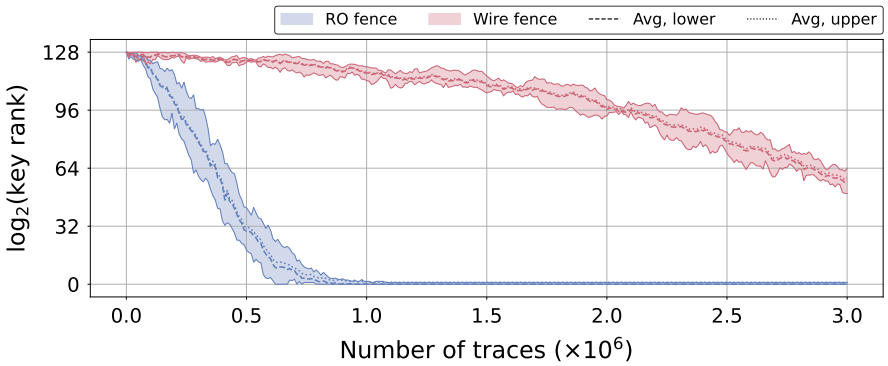


Figure 7.9: Key rank estimation with 3M traces.

Three runs with three million traces

Next, we repeat the experiments, this time to collect 3M traces. The results of the CPA attack are summarized in Table 7.2. According to Metric-30k, to break the first key byte with the wire fence, approx. 2–10× more traces are required than with the RO fence. Moreover, with 3M traces, all key bytes are broken with the RO fence, while only seven to nine are broken with the wire fence.

Fig. 7.9 visualizes the log key rank estimation metric. Analyzing the results with the wire fence, we find that the key rank estimation dropped to 58 bits after 3M traces. With the RO fence, this level was reached after approx. 0.4M traces (7.5× faster). Lastly, with the RO fence, all key bytes were broken after approx. 1M traces, showing again that the wire fence provides superior power side-channel security.

Table 7.2: CPA attack results with 3M traces.

Setup	Run	Number of traces (10^3)		Bytes with key rank ≤			
		Metric-10k	Metric-30k	0	25	50	100
RO fence	4	115	136	16	16	16	16
	5	170	170	16	16	16	16
	6	37	37	16	16	16	16
Wire fence	4	759	864	8	14	16	16
	5	318	345	7	14	15	16
	6	148	394	9	13	14	14

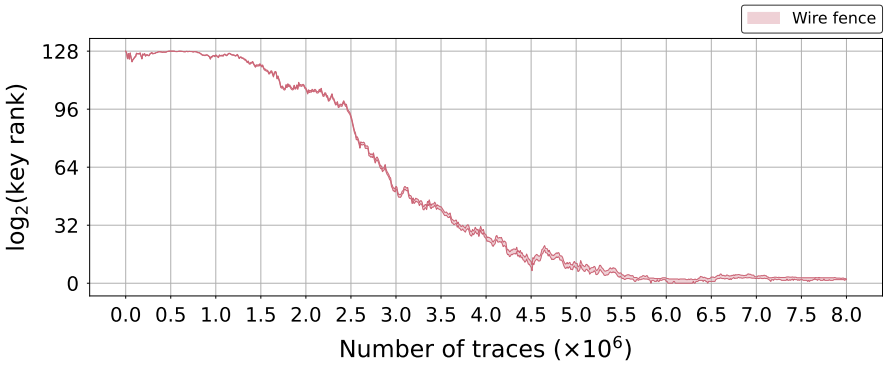


Figure 7.10: Key rank estimation for the wire fence with 8M traces.

One run with eight million traces

In our last experiment, we collect 8M traces with the wire fence and plot the key rank estimation results in Fig. 7.10. The key rank drops to a value below three at approx. 6M traces and stays within the [0, 6) bound for the subsequent 2M traces. Even though not all 128 bits of the key are broken with 8M traces, the key rank estimation stays sufficiently low for an attacker to guess the remaining key bits with a limited effort. Therefore, we can conclude that with the wire fence, at least 6M traces were required to break the key: an improvement of at least 6× compared to the RO fence.

7.5 Chapter Summary

In this work, we presented a design of an active wire fence and demonstrated its ability to provide protection against remote power side-channel attacks in multitenant FPGAs. Our wire fence uses FPGA routing resources to draw more current and generate more noise than a fence built solely with ROs. Comparing the voltage drop resulting from the activation of the wire fence and the RO fence, we found that the RO fence compares to the wire fence of approximately half the logic resources. Therefore, when the space is limited, active wire fences are a better alternative to their RO counterparts. Comparing the side-channel attack effort to break a 128-bit AES key, we found that at least 6× more traces were required in a setup with a wire fence than with an RO fence. Future work will investigate combining enhanced ROs with

the wire wasters, to further improve the efficiency and reduce the resource overhead of the active wire fence.

Part III

Advanced Voltage Sensing and Temperature Analysis

8 FPGA Routing Delay Sensors for Effective Remote Power Analysis Attacks

In Chapter 3, we have seen that on-chip sensors suitable for remote power-analysis attacks can be classified into two groups: TDCs and ring oscillators. The working principle is the same: instead of measuring the voltage directly, they sense the variations of the logic delay caused by the voltage fluctuations, which carry the side-channel information. In parts I and II of this thesis, we have seen that TDC sensors can be successfully leveraged for both leakage evaluation and for performing remote power analysis attacks.

In this chapter, we present a novel FPGA on-chip voltage sensor design, fundamentally different from both TDCs and ROs. To pick up voltage variations, our sensor uses the type of FPGA resources that is the most abundant and least constrained: FPGA wires and routing multiplexers, i.e., FPGA routing resources. The routing-delay sensor (RDS) can be implemented in various ways: with or without a tapped delay line, with or without placement constraints. Furthermore, it can be made stealthy and even more effective than TDC in the context of remote power side-channel attacks.

We begin by designing and implementing two RDS variants working on the principle of a tapped delay line; both use the routing resources only, but one is constrained vertically (VRDS), whereas the other is constrained horizontally (HRDS). Then, we remove the routing

This chapter is based on the work of a paper published in the IACR Transactions on Cryptographic Hardware and Embedded Systems, Volume 2023/2 [34].

and placement constraints to obtain the third variant, which we name simply RDS. As we will demonstrate, the third design is the most performing of the three.

We perform extensive experiments to evaluate the success of a remote power side-channel attack against an AES-128 cryptographic core. To this end, two experimental platforms are used: Alveo U200 (AMD UltraScale+) and Sakura-X side-channel evaluation board (AMD Kintex-7). Experiments are repeated multiple times, the placement of the sensor and the AES is varied, as well as the secret key. We compare the three RDS variants to the TDC implementation commonly used in literature [11, 14, 65, 138] and the recently published voltage-fluctuation sensor VITI [139] (openly available).

Our results show that the final, third variant of our RDS sensor, is more effective not only than VRDS and HRDS, but also VITI and even TDC. When attacking an AES-128 on Sakura-X, the attack with the RDS sensor requires, on average, 35% fewer power side-channel traces to break the entire secret key with the TDC, with this number going as high as 80% in the extreme case. In our experiments, RDS outperforms TDC on an Alveo U200 datacenter card as well.

The remainder of the chapter is organized as follows. In Section 8.1, we describe the threat model of a remote power side-channel analysis attack on cloud FPGAs. Then, in Section 8.2, we explain the design and operation of time-to-digital converters for on-chip voltage sensing. In Section 8.3, we present the designs of our routing delay sensors and explain how to calibrate them. Section 8.4 describes the experimental evaluation approach and the corresponding hardware and software setups. Section 8.5 presents and discusses the results. Finally, Section 8.6 concludes the chapter.

For reproducibility of the experiments and the results in this work, we make the sensor designs and the associated software openly available [140].

8.1 Threat Model

This work follows the common threat model of remote electrical-level attack on multitenant FPGAs [14, 32, 43, 49], in which an FPGA in a datacenter or the cloud is spatially shared by multiple users. The FPGA tenants are given physically separate FPGA regions to deploy their circuits. Furthermore, the assigned regions are logically isolated. To access the external interfaces, such as PCI Express, or the off-chip memory, the tenants use dedicated FPGA logic called *shell*, deployed by the datacenter or the cloud service provider. The tenants are free to implement almost any FPGA circuit (the exception being circuits containing combinational loops [3]) and set placement and routing constraints for their designs. Finally, the FPGA tenant applications share the on-chip power delivery network.

The adversary, in the assigned FPGA region, implements one or more on-chip voltage-fluctuation sensors, together with the control logic and the on-chip buffers, for saving the measurements. The adversary has the possibility of offloading the sensor traces for the off-chip analysis. The victim, on the other side, is performing encryption using a secret key and sending the ciphertexts over a public channel that can be observed by the adversary.

8.2 Sensitivity of TDC Voltage Sensors

In Fig. 8.1, we illustrate the typical implementation of a TDC sensor, as described in Chapter 3. We can identify three parts: (1) a tapped delay line, commonly implemented using fast carry propagation logic and dedicated routing, (2) an output register (with every carry output driving one flip-flop), and (3) some circuitry (e.g., look-up tables, phase-locked loops, or IDELAY adjustable input delay elements) for tuning the phase shift between the sampling clock of the output register and the clock propagating through the delay line. These two clocks have the same frequency.

Careful tuning of the phase shift and the length of the tapped delay line is critical for correct sensor calibration, i.e., ensuring that only one clock transition is captured in the output register

per sampling clock period. Additionally, the delay line must be properly formed by chaining the carry output of one FPGA slice to the carry input of the next one, where all the occupied slices are constrained to one vertical column of the FPGA; and, every carry output must drive precisely the corresponding FF residing in the same slice. These strict placement constraints are necessary for ensuring the optimal sensitivity of this TDC sensor.

In the absence of any on-chip activity, the sensor output usually is constant (modulo background noise) and determined by three parameters: the clock frequency, the initial delay (i.e., the phase shift), and the length of the delay line N. For a given clock frequency, the initial delay and the parameter N are chosen so that the output register captures a single clock transition in every clock cycle. In other words, the output register should always be filled with a sequence of ones followed by the sequence of zeros—with a possibly imperfect transition—where the location of the FF with the transition corresponds to the depth of propagation of the clock signal through the delay line. The value in the output register can be converted to the numerical value of one *sensor sample* using a thermometer code [49] or by taking the Hamming weight of the bits in the output register [14,65], as we do in this work. Once the on-chip voltage starts fluctuating due to the activity of the victim circuit, the delay of the elements in the sensor change, and consequently, so does the sensor output. We consider the sensor well calibrated (i.e., the initial delay and the length of the delay line are well chosen) if, for the entire duration of the measurement, the Hamming weight of the output register lies in the range $0 < \mathrm{HW}(O) = \mathrm{HW}(O_0, O_1, ..., O_{N-1}) < N$ and only one clock edge is captured.

Let us define d_i as the time the delayed clock signal takes to propagate from the input of the

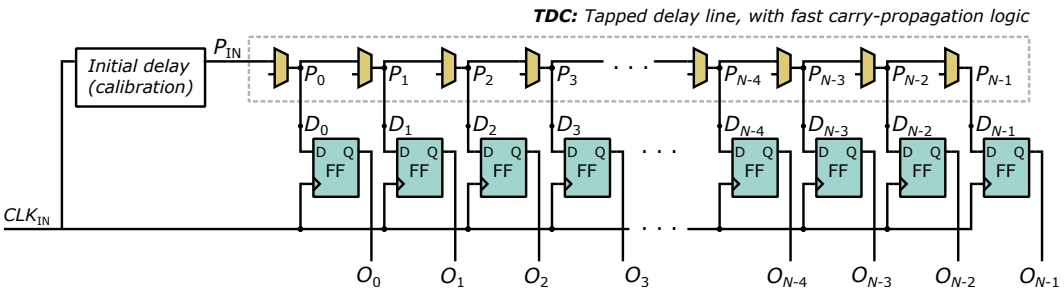


Figure 8.1: TDC sensor.

tapped delay line P_{IN} to the input D_i of the flip-flop FF_i. For the TDC illustrated n Fig. 8.1, d_i is, therefore, the following time difference:

$$d_i = t(D_i) - t(P_{IN}). \tag{8.1}$$

The sensitivity of the TDC sensor, i.e., the minimum signal change that can be detected, can be expressed as

$$S = \min(d_i - d_{i-1}), \;\; 0 < i < N. \tag{8.2}$$

Because the TDC has a tapped delay line, we can rewrite the expression for d_i as the sum of the propagation delay through the first segment, last segment, and all the intermediate segments of the delay line:

$$d_i = \underbrace{t(P_0) - t(P_{IN})}_{\text{first segment}} + \underbrace{t(D_i) - t(P_i)}_{\text{last segment}} + \underbrace{\sum_{1 \le k \le i} t(P_k) - t(P_{k-1})}_{\text{intermediate segments}}, \tag{8.3}$$

where P_i is at the output of the delay element i in Fig. 8.1. Combining the expressions in Equations (8.1), (8.2), and (8.3), the sensitivity can be reformulated as:

$$S = \min\left(\underbrace{t(D_i) - t(P_i)}_{\text{routing resources}} + \underbrace{t(P_i) - t(P_{i-1})}_{\text{delay element}} - \underbrace{(t(D_{i-1}) - t(P_{i-1}))}_{\text{routing resources}} \right). \tag{8.4}$$

Therefore, the sensitivity of the TDC improves if, first, the input signal takes a very short time to pass through a delay element; and, second, if the delays through routing resources ($t(D_i) - t(P_i)$, for $0 \le i < N$) are all approximately equal. These two requirements have a direct impact on the way TDCs are often implemented on FPGAs: the former is satisfied by using fast carry propagation logic, while the latter is achieved by ensuring that each carry output drives only the corresponding flip-flops in the *same* slice. Routing outside of the slice boundaries is, in general, avoided. The exception is when it is necessary to connect the carry output of one slice to the carry input of another. Even then, dedicated carry-specific routing resources are used to minimize the penalty, and consequently, the tapped delay line has to be placed in a

single FPGA column.

Another property of a tapped delay line is the monotonic increase of the propagation delay:

$$d_0 < d_1 < \cdots < d_{N-1}. \tag{8.5}$$

In practice, carry propagation logic in FPGAs is often implemented in the look-ahead fashion, and the delays from the carry input to the carry outputs of the same slice may not be strictly monotonically increasing [14]. Given that we apply Hamming weight instead of the thermometer code on the output register value, the mentioned lack of monotonicity is of no practical concern.

8.3 Routing Delay Sensors

As described in Sections 8.2, the principle of operation of the FPGA on-chip voltage sensors is measuring and quantifying the change of the propagation delay of the carry logic (in TDCs) or LUTs (in ring oscillators and VITI). Even though it is not explicitly mentioned in the literature on remote power-side channel attacks, the routing multiplexers in the FPGA interconnect are affected by the on-chip voltage fluctuations and they too contribute, though to a lesser extent, to the delay variations captured by the sensors. Ahmed et al. explored techniques for optimizing FPGA logic circuitry for variable voltage supplies [141]. As part of the study, the authors compared the impact of the power supply voltage on the propagation delay of LUT-dominated and routing-dominated signal paths, to find that (1) both were affected and (2) the LUT-dominated paths were affected more. Motivated by their findings, we ask ourselves the following research question: *if the delay line is built using FPGA routing resources only, how effective would such a sensor be in the remote power SCA attack setting?* To answer this question, we start by designing the following two variants of a routing delay sensor:

- VRDS, with the placement and the routing constrained *vertically*, as in TDCs, and
- HRDS, with the placement and the routing constrained *horizontally*, completely oppo-

site of the TDCs.

In the remainder of this section, we present our VRDS and HRDS implementations and explain their operation. Then, we introduce a third and improved design. Finally, we discuss the calibration procedure and the challenges of portability and detectability of the routing delay sensors.

8.3.1 VRDS and HRDS

In Fig. 8.2, in the same style as in Fig. 8.1 for TDCs, we illustrate the design of our routing delay sensor, in which the delay line is implemented with FPGA routing resources (RR) only. We use the label RR to model the global interconnect; the local interconnect is modeled with the direct connection between P_i and D_i, for $0 \leq i < N$. It is worth noting that all the expressions in Equations (8.1), (8.2), (8.3), and (8.4) hold here as well.

When placing and routing, we aim for a modular design and as uniform RR delay as possible across the entire delay line. Fig. 8.3a shows the placement of two subsequent FFs in the output register and the routing of the delay line for the VRDS. The delay element labeled RR in Fig. 8.2 is realized with a vertical wire segment of length one (the full line), whereas the segment between P_i and D_i is routed locally (dashed line). The placement of the output register is constrained to a single FPGA column. This placement and routing pattern is then repeated for the entire length of the sensor. Reusing the same connectivity pattern helps improve the sensitivity of the sensor, as the first and the third of the three terms in Eq. (8.4) become equal, thus canceling each other out.

The implementation of the HRDS is similar, except for using horizontal wires and constraining the FFs to a single FPGA row (Fig. 8.3b). However, there is one important difference between the HRDS and the VRDS. FPGA columns have a uniform architecture, whereas FPGA rows do not (e.g., in one FPGA row, there are CLBs, DSP blocks, RAM memories, etc.). As a consequence, in the case of the HRDS, the RR blocks in Fig. 8.2 no longer have a constant delay because the wires connecting two immediately neighboring CLBs are shorter than the wires between two

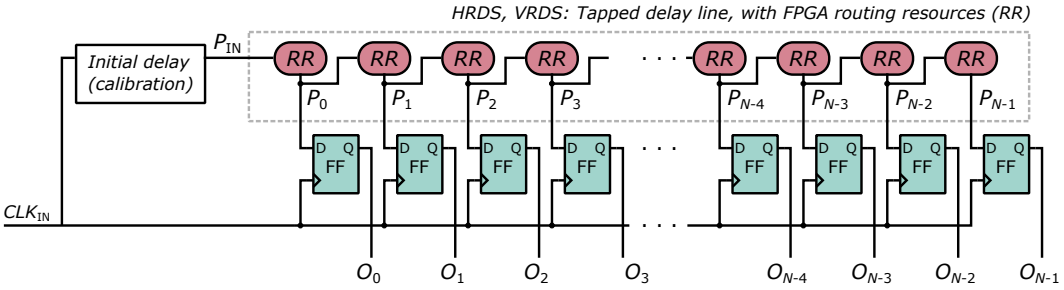
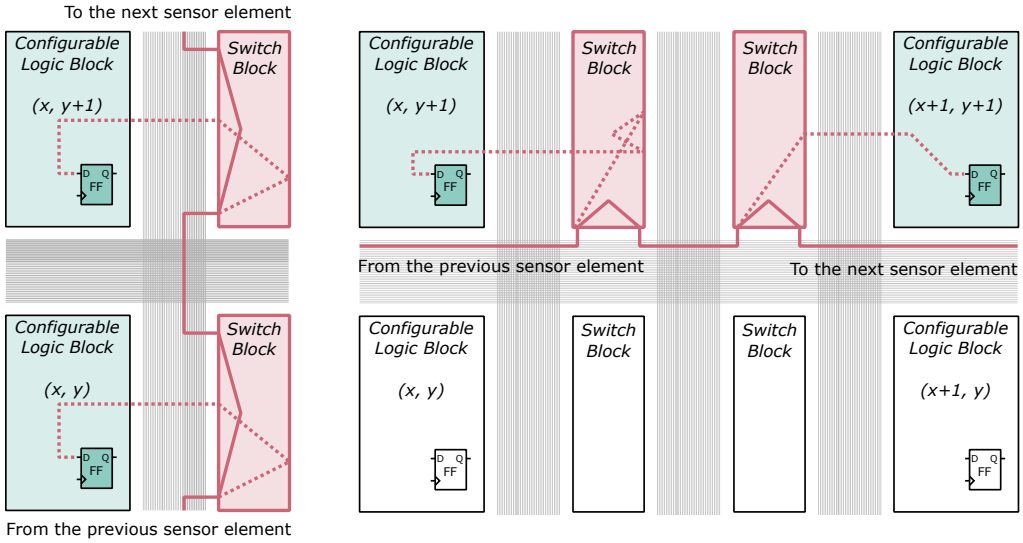


Figure 8.2: Routing delay sensor with a tapped delay line.



(a) VRDS (b) HRDS

Figure 8.3: Placement and routing of a segment of VRDS and HRDS sensors.

CLBs that are separated by a column of DSP or memory blocks.

8.3.2 RDS

As the global interconnect wires are longer than the dedicated connections in the carry propagation chains, we expect both VRDS and HRDS to have lower sensitivity than the TDC and, hence, to be less effective for remote power side-channel attacks. When it comes to VITI, it is harder to predict how it compares to VRDS and HRDS without experimenting, as the sensitivity of VITI is considerably lower than the sensitivity of the TDC, because of the LUTs in the tapped delay line.

Is it even possible to build an RDS sensor with better sensitivity than TDC? The answer is *yes*. To explain how, let us again recall the expression for the sensitivity in Eq. (8.4). The first and the last term are unavoidable, as the connections to the inputs of the FFs in the output register have to exist. Ideally, the first and the third term could cancel each other out, if the exact same last-mile routing path is taken; this is the case in our realization of the VRDS and the HRDS. The middle term in Eq. (8.4), labeled as delay element, is more challenging to optimize. In TDCs, the carry propagation logic is fast, hence the high sensitivity of TDCs. In VITI, VRDS, and HRDS sensors, the middle term has a more significant impact on the sensitivity.

It is hard to imagine that the middle term in Eq. (8.4) can be better optimized than it is already with the dedicated carry propagation logic and routing. Therefore, we turn to the alternative expression for sensitivity in Eq. (8.2) and formulate a new sensor design goal.

Goal: *All the paths between P_{IN} and D_i should be routed so that the delays d_i, where $0 \leq i < N$, are as similar as possible:*

$$d_0 \sim d_1 \sim d_2 \sim \cdots \sim d_{N-2} \sim d_{N-1}.$$

Ideally, the difference between any pair (d_i, d_j), where $0 \leq i, j < N$, should be lower than the sensitivity of a TDC. In practice, the more (d_i, d_j) pairs that satisfy the above goal, the more effective we expect the sensor to be, compared to the TDC, in an attack scenario.

The ideal sensor we formulate above, however, would not be usable in practice because the observable time window, i.e., the range of the sensor

$$W = \max|d_i - d_j|, \;\; 0 \leq i, j < N \tag{8.6}$$

would be significantly reduced compared to any delay-line-based sensor, where $W = d_{N-1} - d_0$. Such a sensor would not only be challenging to calibrate, but it would also be prone to becoming easily decalibrated, to the extent that even the calibration at run time may not be the solution. With this in mind, we rephrase our sensor design goal as follows.

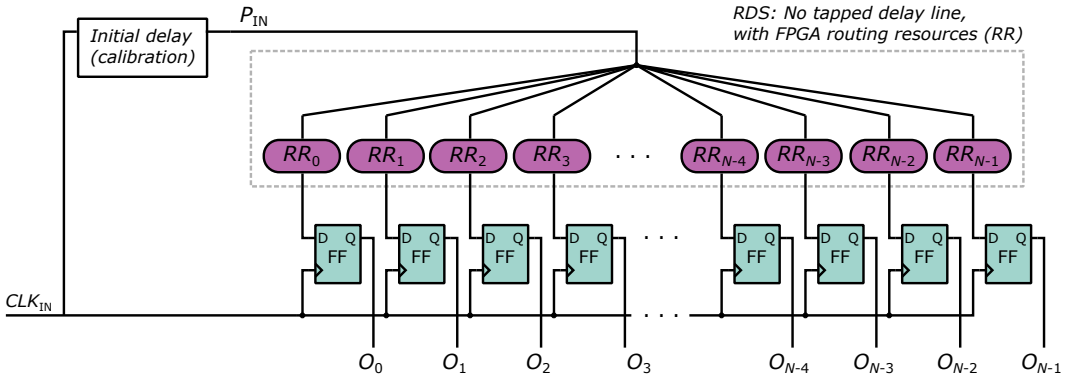


Figure 8.4: RDS sensor.

Table 8.1: Characteristics of the TDC, VITI, and our three variants of the RDS sensor.

Sensor	**TDC [11]**	**VITI [139]**	**VRDS**	**HRDS**	**RDS**
Main component	Carry chain	LUT	Routing resources (wires, multiplexers)		
Tapped delay line	Yes	Yes	Yes	Yes	No
Placement and routing	Vertically constrained	Unconstrained	Vertically constrained	Horizontally constrained	Unconstrained
Sensitivity	High	Low	Low	Low	High

Goal: *Many of the paths between P_{IN} and D_i should be routed so that the delays d_i, where $0 \leq i < N$, are as similar as possible. At the same time, the observable time window (W in Eq. (8.6)) should not be too narrow.*

The approach we take to achieve the above goal is threefold: first, not having a tapped delay line at all; second, letting the FPGA router build the connections between P_{IN} and D_i, $0 \leq i < N$; and, third, letting the FPGA placer decide on the locations of the FFs in the output register, in the interest of helping the router find suitable paths. In Fig. 8.4, we illustrate the resulting RDS design. Finally, it is worth noting that using Hamming weight to convert the binary value in the output register to the integer value of the sensor sample applies perfectly well also to the RDS sensor free of the tapped delay line.

In Table 8.1, we summarize the main characteristics of the TDC, VITI, and our three variants of the RDS sensor.

8.3.3 Calibration

The goal of the calibration is to ensure that an edge of the clock signal is within the observable delay line when its state is captured in the output register. Drewes et. al [86] later showed that in the case of TDC, there is no significant difference in attack success between capturing the rising or falling edge of the clock in the delay line, provided the sensor uses Hamming weight encoding and is calibrated to maximize the power side-channel leakage. Therefore, even in the case of RDS, one can choose either the rising or the falling edge and adjust the calibration procedure accordingly. In this work, we opt for the rising edge.

As the calibration is a lengthy process of trial and error, it is convenient to automate it. We implement a reconfigurable initial delay line approach proposed in literature [65]: implementing delay line elements as multiplexers allows a reconfigurable clock entry point in the initial delay line, thus changing the clock delay. Selecting a subset of the available coarse delay elements (in our implementation, LUTs) and fine delay elements (carry propagation logic) allows fine clock delay tuning required for voltage-drop sensors [65]. We control the calibration from software for better control and higher flexibility.

For the sensors with a tapped delay line (TDC, VITI, VRDS, and HRDS), the calibration procedure is straightforward: increasing (respectively, decreasing) the initial clock delay—an action that moves the rising edge closer to (respectively, further from) the beginning of the delay line (the point P_{IN} in Figs. 8.1 and 8.2)—until the rising edge is close to the middle of the observable delay line [12]. Considering the Hamming weight of the output register, the rising edge being close to the middle of the observable line translates to the sensor sample being equal to approximately $N/2$.

Given that the RDS does not have a tapped delay line but a tree of routing resources, it requires a somewhat different calibration approach. We can no longer tell where the edge precisely lands, but we can observe and influence the Hamming weight of the output register. We aim to calibrate the sensor so that the power side-channel trace (i.e., the time sequence of sensor samples for a given measurement duration) has a high variance. In other words, the more bits

Algorithm 3: Calibration algorithm

Input : L_{IDC}, maximum number of coarse elements
L_{IDF}, maximum number of fine elements
n, number of samples per trace
N, observable delay line length (maximum sensor value)
δ, calibration parameter
N_{traces}, number of traces recorded at each calibration step

Output: IDC, IDF, number of coarse and fine elements, respectively

```
for IDC_cnt to L_IDC do
    s_min ← N;
    IDC ← IDC_cnt; IDF ← 1;
    send_calibration(IDC, IDF);
    for trace to N_traces do
        (s_1, s_2, ..., s_n) ← record_trace();
        s_min ← min(s_min, min(s_1, s_2, ..., s_n));
    if s_min = N then
        break;
for IDC_cnt to L_IDC do
    for IDF_cnt to L_IDF do
        s_max ← N;
        IDC ← IDC_cnt; IDF ← IDF_cnt;
        send_calibration(IDC, IDF);
        for trace to N_traces do
            (s_1, s_2, ..., s_n) ← record_trace();
            s_max ← max(s_max, max(s_1, s_2, ..., s_n));
        if s_max = δ then
            return IDC, IDF;
return failure;
```

in the output register that toggle when a voltage drop happens, the better. Our RDS calibration approach is illustrated in Fig. 8.5 and described in Algorithm 3.

The algorithm has two parts that together ensure that many bits of the output register capture the changes in the propagation delay of the rising edge of the clock signal. In the first part (lines 2–13), we incrementally increase the initial delay by including more coarse elements (IDC) until the falling edge of the clock exits the observable delay line and all the output bits become '1': $HW(O) = N$. The effects of increasing the initial delay are illustrated in steps 1, 2, and 3 in Fig. 8.5. We stop including more coarse elements (line 10 of the algorithm)

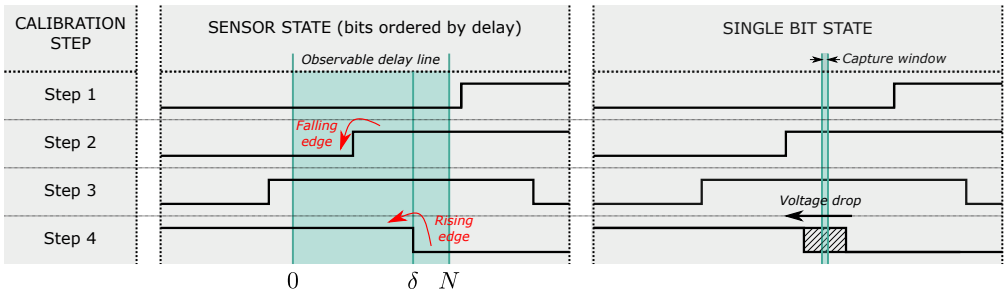


Figure 8.5: Calibration steps for the RDS sensor.

when all the samples in N_{traces} measured traces reach the maximum value, N. In the second part of the algorithm (lines 14–28), we gradually increase the number of coarse and the fine (IDF) elements, to bring the rising edge carefully into the observable delay line. For every (IDC, IDF) pair, we record N_{traces} traces and compute the maximum value of all the sensor samples. When that maximum value crosses the predefined threshold δ ($delta < N$), as shown in step 4 of Fig. 8.5, the calibration is finished. Alternatively, we choose another (IDC, IDF) pair and repeat the procedure.

The user-controlled threshold δ ($0 \leq \delta < N$) determines the number of output FFs that capture '0' instead of '1'. If δ is high, a few bits in the output register satisfy this condition: for them, the clock edge has passed to the left of the capture window (shown in the right half of Fig. 8.5). For the other bits, those equal to '1', the clock edge is to the right of the capture window. It is these bits that, when the on-chip voltage drops due to the victim activity and, consequently, the clock edge moves to the left of the capture window, may change from '1' to '0'. If it happens, this change will be reflected in the lower Hamming weight of the output register (i.e., the lower value of the sensor sample s in Algorithm 3).

If δ is low, for many bits in the output register the clock edge will be to the left of the capture window. These bits are unlikely to change their value, as lowering the on-chip voltage moves the edge further away from the capture window. Therefore, low δ results in fewer bits potentially toggling and, hence, less side channel leakage getting captured. Therefore, we set δ to a value close to N, so that for as many output FFs as possible, the clock edge is likely to enter the capture window during trace recording. If the calibration fails, then the width of the

observable window N needs to be increased, by adding more FFs in the sensor output register.

The described calibration algorithm is not limited to RDS; it can be used for VRDS, HRDS, and even TDC. For these three sensors, we expect the observable window to be wider than for RDS. Consequently, for the same supply voltage variations, fewer bits in their output registers should toggle. From the calibration perspective, fewer bits toggling means that a wider range of δ values should result in equally correct calibration.

8.3.4 Portability and Detectability

As described in Section 8.2, TDC sensors employ carry chain logic and require strict placement constraints, ensuring that the carry chain is correctly and vertically formed and that each carry output drives the corresponding flip-flop residing in the same FPGA slice. These constraints ensure the tapped delay line is fine-grained, as uniform as possible, and that only dedicated connections are used. Depending on the FPGA device on which the sensor is to be deployed, the constraints may need to be adjusted to account for a different sensor location and type of carry logic available (e.g., CARRY4 or CARRY8). As described in Sections 8.3.1 and 8.3.2, unlike TDC, neither VRDS, HRDS, nor RDS, require carry-chain logic to sense voltage variations. In the case of HRDS and VRDS, the sensor location, the placement of the flip-flops, and additionally, the resources to route the clock signal through the tapped delay line need constraining. The sensor location aside, in RDS, neither the placement of the flip-flops nor the choice of routing resources needs to be specified by the adversary, which makes RDS easier to deploy and port across FPGA devices.

For the purpose of calibration, both TDC and the routing delay sensors require means to adjust the clock phase. We provide it using LUTs and carry-chain logic (the latter requiring placement constraints). However, if the ease of implementation and portability are of importance, then other approaches, such as a PLL or an adjustable input-delay element [139], can be used instead. These alternative approaches are equally suitable for TDC as for the routing delay sensors.

Given the high interest in fault and power analysis attacks on cloud FPGAs, researchers have proposed bitstream checking tools [132, 133]. Their principle of operation is to, first, reverse engineer the bitstream into a design netlist and, second, search for potentially malicious patterns. A simple example of a malicious pattern is a combinational loop; some cloud FPGA providers are able to detect it during synthesis and flag it as a design error. This check effectively prevents LUT-based ring oscillators (either as sensors or power wasters) from being deployed on the cloud.

In the case of TDC sensors, bitstream checkers could look for a number of potential issues, starting with timing violations [132], because the clock propagating through the tapped delay line has to violate timing by construction. RDS sensors, similarly to TDC sensors, introduce timing violations in the output register. These timing violations can be bypassed, equally efficiently for TDC and RDS, by using programmable clock-generating circuits (i.e., PLLs or mixed-mode clock managers). It suffices to set a sufficiently low clock frequency at compile time, not to violate the timing constraints, and then change it to the desired value during runtime.

A clock-to-data path [132], inherent to both TDC and RDS, is another flag-raising netlist structure. Yet, if needed, it can be easily avoided; for example, by adding a T flip-flop on the clock path and using its output instead of the clock to drive the delay line.

Precisely constrained carry chains, when detected, can indicate the presence of a sensor. While the carry chains are the basic building blocks of TDCs, RDS can be entirely free of them and, therefore, pass the check.

Finally, connecting the clock signal to the flip-flops of the RDS sensor output register creates a relatively high fan-out signal, yet another feature checked by bitstream scanning tools in literature [133]. However, these tools look for orders of magnitude higher fan-out, common to power-wasting circuits. The fan-out of 128 (or less) in RDS is not uncommon in FPGA designs (e.g., for enable and reset signals of registers) and, as such, calls for no alarm [133].

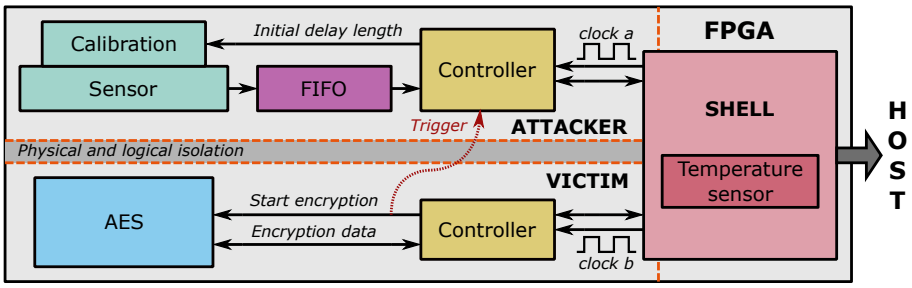


Figure 8.6: System architecture.

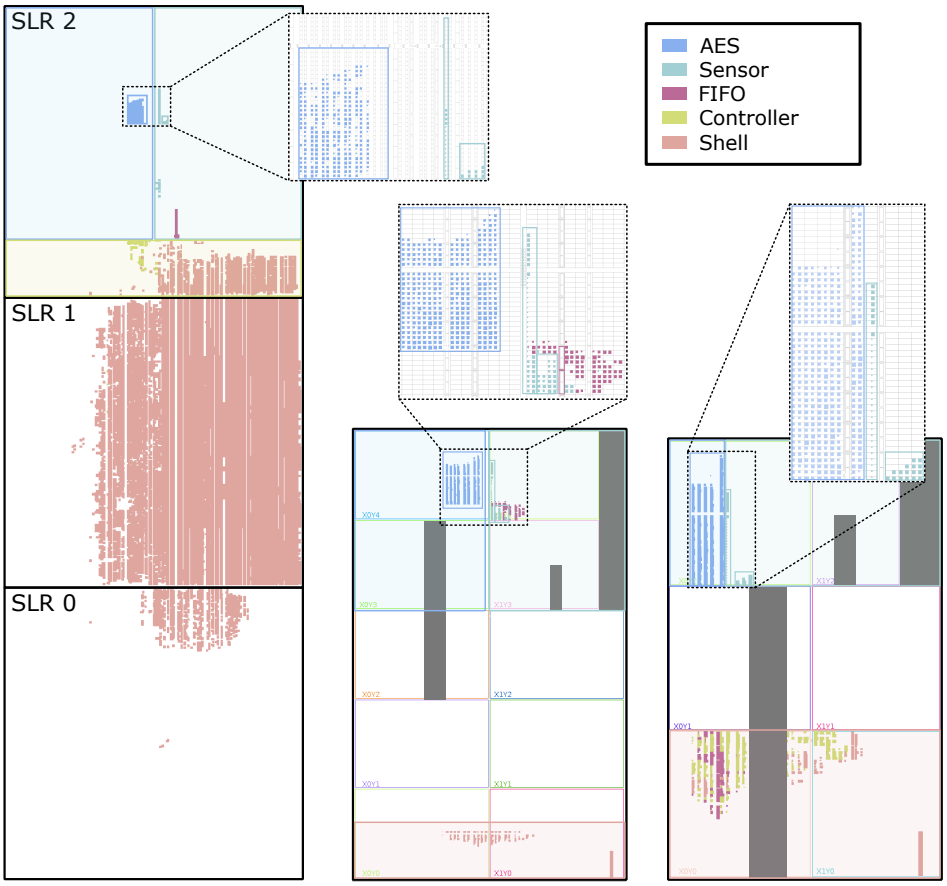
8.4 Experimental Evaluation

This section presents the experimental evaluation methodology. We first describe the architecture of the system used to record power traces. Then, we provide a detailed description of the experiments and, finally, explain the attack metrics used to evaluate and compare the effectiveness of the sensors in the remote power side-channel attack setting.

8.4.1 System Architecture and Floorplan

Our experimental setup consists of three different FPGA boards, allowing us to evaluate the RDS sensor across various FPGA families. For most experiments, we use the side-channel evaluation board Sakura-X [103], equipped with a AMD Kintex-7 FPGA (xc7k160tfbg676-1). Additionally, we perform experiments on AMD Alveo U200 datacenter accelerator card, having an AMD Virtex UltraScale+ FPGA (xcu200-fsgd2104-2-e). We use Vivado 18.03 for the Sakura-X board, and Vivado 2022.1 for the Alveo U200 design. For compilation, we use the default Vivado synthesis and implementation strategies.

Fig. 8.6 shows a block diagram of the system architecture used in all three FPGA boards. Despite the FPGA-specific implementation differences, the system architecture of all three setups has the same main components: The shell, responsible for the communication between the FPGA tenants (the adversary or the victim) and the host machine. The victim is an AES-128 hardware module [87]; it has an associated controller, for transferring plaintexts and ciphertexts, and for initiating encryption. The adversary, physically isolated from the victim, has a voltage-



a) Alveo U200 b) Sakura-X c) Basys 3

Figure 8.7: Floorplan of all three boards used in the experimental evaluation.

fluctuation sensor calibrated by a controller module, and a FIFO buffer where sensor samples are stored before they are offloaded to the host machine. To facilitate easy trace collection, we use the encryption initiation signal as a trigger for storing the sensor traces.

Fig. 8.7 shows the floorplan of the system on the three FPGAs. In all implementations, the attacker and victim reside in separate regions (Pblocks). The AES and the sensor are placed close to one another, simulating the worst-case scenario for the victim. Because of different FPGA/host communication constraints, the shell and the controller implementation differ for each FPGA. On Sakura-X, the shell occupies the smallest area, as the communication happens through a serial connection and an additional control FPGA [103], which reduces the noise of the communication process. On Alveo U200, the automatically inserted shell occupies the

Table 8.2: Key and plaintext values used in the experimental evaluation.

Key	Key value	Plaintexts
K1	0x7d266aecb153b4d5d6b171a58136605b	$PT_0 = 0$ $PT_{i+1} = CT_i$
K2	0xe3fb107fa4aaeb7130f411d4c88dbf6c	
K3	0xa89e2fd6926dc2478402b717631d08ce	
K4	0xa3a03d60c06457dc65d8afd5815f629c	
K5	0xe1055ac2abadea4fc7fc6be1310448d9	

largest share of the FPGA: it is static and prevents using a custom, smaller shell. Users are constrained to use the AXI-4 shell interface, making the design and implementation of the controllers more complex. The clock frequencies of the sensor and the AES are set to 200 MHz and 20 MHz, respectively.

8.4.2 Experimental Setup

In our experiments, we aim to break the secret key of an open-source AES-128 cryptographic module implementation [87], using CPA. We record N_{TRACES} power side-channel traces per experiment (the exact value of N_{TRACES} depends on the experimental platform). For each trace acquisition, we send the key K and the plaintext PT to the AES core and record all the sensor samples captured during the AES encryption. We use the current ciphertext as the next plaintext, to avoid plaintext repetition. For simplicity, as seen in Fig. 8.6, we use the start encryption signal to trigger the collection of a sensor trace. In real attacks, this signal is absent, and the attacker resorts to trace alignment and automatic triggering techniques described in previous work [12]. We repeat the experiments with five different keys to draw more substantial results. The keys and the plaintexts used in the experimental evaluation are listed in Table 8.2.

8.5 Results and Discussion

In this section, we present the results of the experiments. We start by comparing the power side-channel traces measured with the TDC and the RDS. Then, we compare the RDS sensor

Table 8.3: Resource utilization of TDC, VITI, and RDS.

Sensor	Initial delay (calibration)			Other resources		
	LUT	CARRY	Latch	LUT	CARRY	FF
TDC	32	0	32	0	32	128
VITI	32	0	32	4	0	4
VRDS	32	0	32	0	0	32
HRDS	32	0	32	0	0	16
RDS	32	24	32	0	0	128

with the TDC [65] by performing a power side-channel attack and a statistical analysis of the characteristics of the sensor traces. Finally, we compare the RDS with the TDC across a range of different placements for the sensor and the AES.

Table 8.3 lists the FPGA resource utilization of our implementations of the TDC, VITI, and RDS sensors. As described in Section 8.3.3 and similarly to previous work [65], we use the coarse (LUTs, latches) and fine (carry) elements for calibration. With 32 LUTs and latches, and 24 fast carry elements for fine phase shift tuning, every sensor in Table 8.3 can be calibrated. Because of the high sensitivity of the TDC and the RDS, we set the output register size to 128 bits ($N = 128$). Unlike the TDC, VITI has a short observable delay line and uses only 4 LUTs and FFs [139]. For VRDS and HRDS, we set the output register size, N, to 32 and 16, respectively.

We constrain the RDS register to a Pblock having $\approx 2\times$ more flip-flops than required and let Vivado complete the placement and routing (P&R). We conjecture that if the assigned Pblock is not overly resource-limited, the P&R will not be a hard task and, hence, Vivado will place the flip-flops and route the signals in a close to optimal way.

8.5.1 RDS Sensors Versus TDC and VITI

As a first step in the experimental analysis, in Fig. 8.8 we visualize the waveforms of the RDS and the TDC traces recorded during the encryption of a single plaintext on Sakura-X. The traces are placed side by side for easier comparison. In both of them, the AES rounds are clearly visible. The traces have 128 samples, covering the entire duration of one AES-128

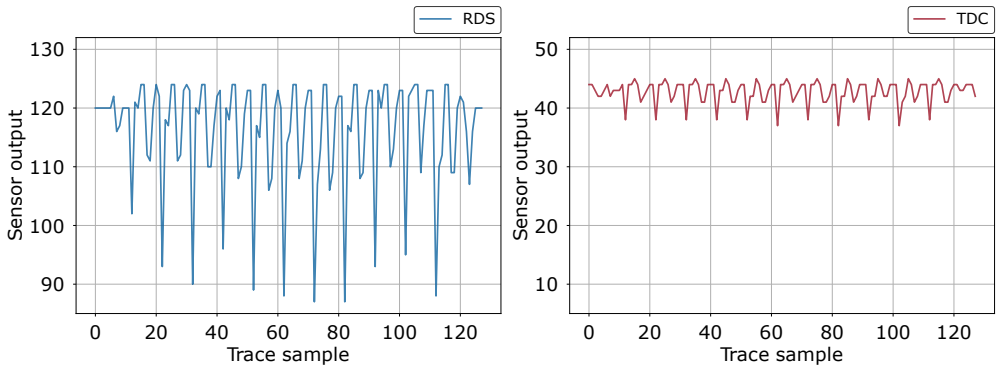


Figure 8.8: One side-channel trace recorded with the RDS (left) and the TDC (right) sensor.

encryption. Because of two independent calibrations, the vertical offsets of the traces differ. What is more important to notice is that the peak-to-peak amplitude of the RDS trace is higher (37) than the peak-to-peak amplitude of the TDC trace (8). The higher RDS trace variation suggests that an attacker with the RDS may be able to break the secret key faster. To evaluate this hypothesis, we run additional experiments.

We record 100k traces (corresponding to the encryption of 100k plaintexts) for each of the five keys in Table 8.2, as described in Section 8.4.2. We run the experiments for the TDC, VITI, and the three variants of our RDS sensors. Fig. 8.9 visualizes the results of the statistical analysis of the sensor samples recorded for the key $K1$. On the left, we compare the number of output bits with nonzero variance. For VITI, VRDS, and HRDS, there are only one or two bits that toggle. In the case of TDC, there are 11 bits. Finally, the RDS has the highest number of bits toggling: 47. This result explains the waveforms in Fig. 8.8, as the higher peak-to-peak value is in direct relation with the number of bits toggling.

The right part of Fig. 8.9 depicts the variance of the output bits of the RDS and the TDC. As expected, the TDC has a cluster of bits with nonzero variance, where the rising clock edge lands in the delay line. This figure also highlights the difference between the RDS and the TDC: replacing the delay line with free routing in the RDS results in a higher number of bits toggling. Additionally, the toggling bits are not necessarily clustered closely together.

Let us now compare the SNR for the RDS and TDC. The SNR is a side-channel evaluation

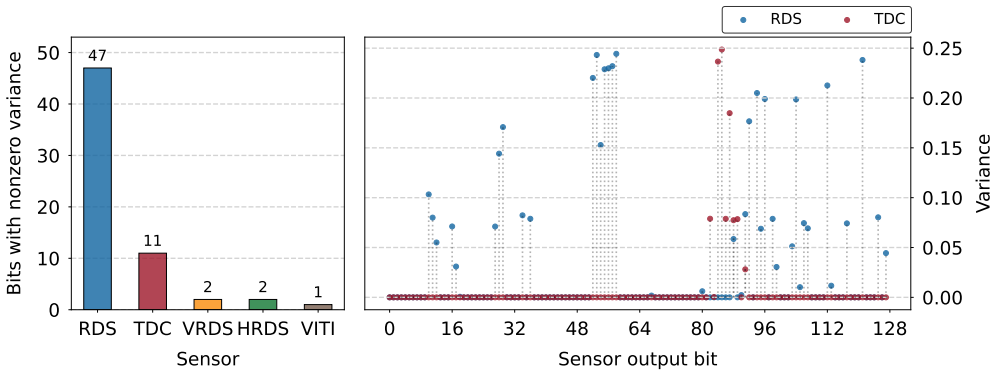


Figure 8.9: Number of bits toggling during trace acquisition for every sensor (left), and the variance of the bits for RDS and TDC sensors (right).

metric defined as the ratio between the useful signal, i.e., the variance of the data-dependent power consumption, and noise. It can be obtained from the power side-channel traces without performing an attack and is most commonly used to identify trace samples with significant leakage (i.e., samples that are commonly linked to the secret key). To compute the SNR, we follow the procedure outlined by Papagiannopoulos et al. [70]. Fig. 8.10 shows the results corresponding to the least-significant byte of the output of the ninth AES round, for the keys in Table 8.2. For both sensors, we can observe two peaks: in sample 102 (the beginning of the last AES round) and in sample 112 (the end of the last round, i.e., the moment when the ciphertext is saved in the state register). RDS is clearly superior to TDC, as the SNR approximately doubles in these two points of interest. Across all the experiments and every byte of the intermediate value, SNR in sample 112 for RDS is consistently higher than for TDC, by a factor of 1.57× on average, with a maximum of 2.87×.

To compare the sensors in the power side-channel attack scenario, we attack the traces using CPA and the key rank estimation metric, and repeat the experiment five times (each time with a different key). Fig. 8.11 shows the attack results. As outlined in Chapter 3, the key rank is estimated as a range. The dashed and dotted lines represent the lower and the upper bounds of this range, averaged over all the experiments. The shaded areas indicate the entire range of the key rank (min, max), observed across all the runs. The results demonstrate that the RDS and the TDC, as predicted, are superior. The coarse delay lines of VITI, HRDS, and VRDS

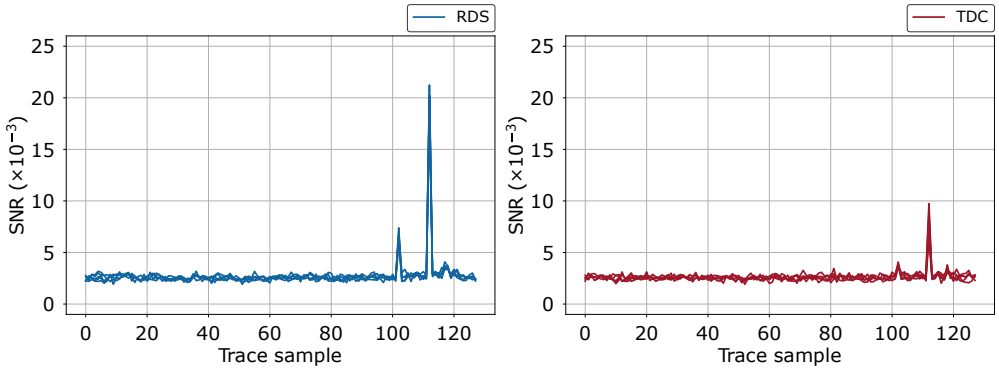


Figure 8.10: Signal-to-noise ratio for the RDS (left) and TDC (right), computed on the least-significant byte of the output of the ninth AES round.

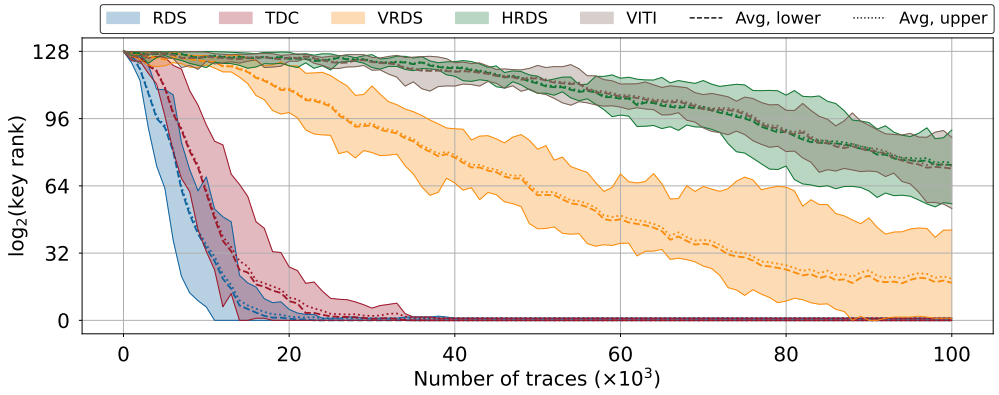


Figure 8.11: Key rank estimation for TDC, VITI, and our three RDS variants.

result in lower sensor sensitivity, making it difficult for the sensor to capture small voltage fluctuations. HRDS and VITI give very similar results, suggesting that both the horizontal wires and the LUTs have a similar response to voltage fluctuations. VRDS, however, is superior. This result is not surprising, as vertical routing uses shorter wires than horizontal, thanks to the absence of heterogeneous blocks (DSPs, memory) within an FPGA column.

Finally, and most interestingly, Fig. 8.11 shows that, on average, the RDS sensor outperforms the TDC. As the RDS sensor has more output bits with nonzero variance and the RDS trace has higher peak-to-peak amplitude, an attack with the RDS requires fewer traces to recover the full key.

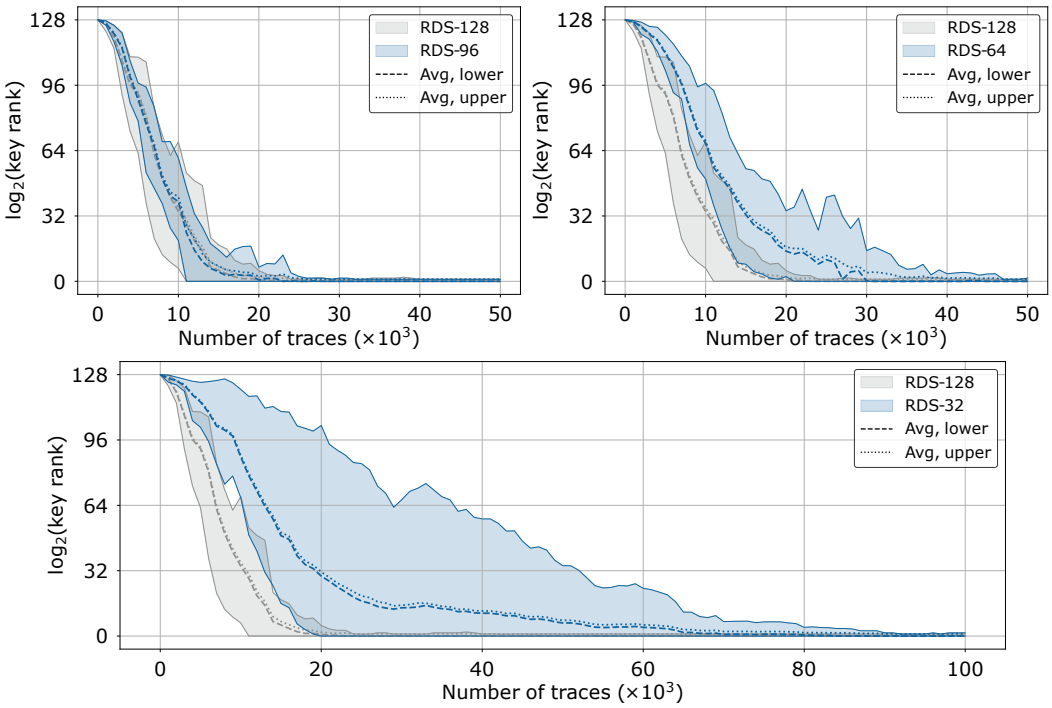


Figure 8.12: Comparison of RDS sensors with the number of bits used.

8.5.2 Impact of The RDS Size on The Attack Success

The number of FFs in the output register of the TDC does not impact its performance—provided that the sensor is correctly calibrated. This is not the case for the RDS. The left side of Fig. 8.13 shows the number of bits with nonzero variance for RDS with 128, 96, 64, and 32 FFs in the output register. The right side shows the variance of each bit in the output register, computed across 100k traces (corresponding to 100k AES-128 encryptions on Sakura-X, with the key K1). We see that reducing the output register size results in fewer bits that toggle.

Fig. 8.12 shows the results of the key rank analysis. In general, increasing the number of bits in the output register leads to fewer number of traces needed to break the full AES key. However, for 96 and 128 bits in the output register, there is no notable difference. These results correlate well with the per-bit variance shown in Fig. 8.13 and the intuition that the more bits with the nonzero variance, the more effective the attack.

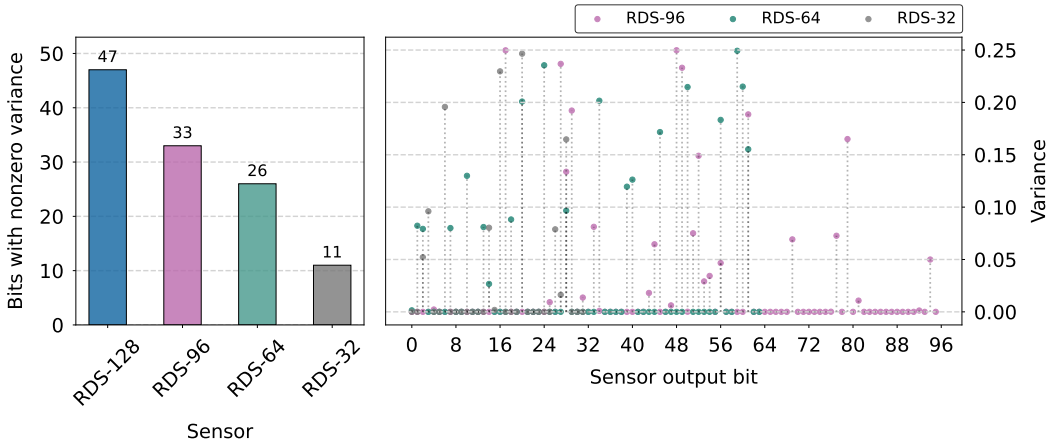


Figure 8.13: Left, the number of bits toggling during 100k trace acquisitions for the RDS with 128, 96, 63, and 32 bits in the output register. Right, the variance per bit.

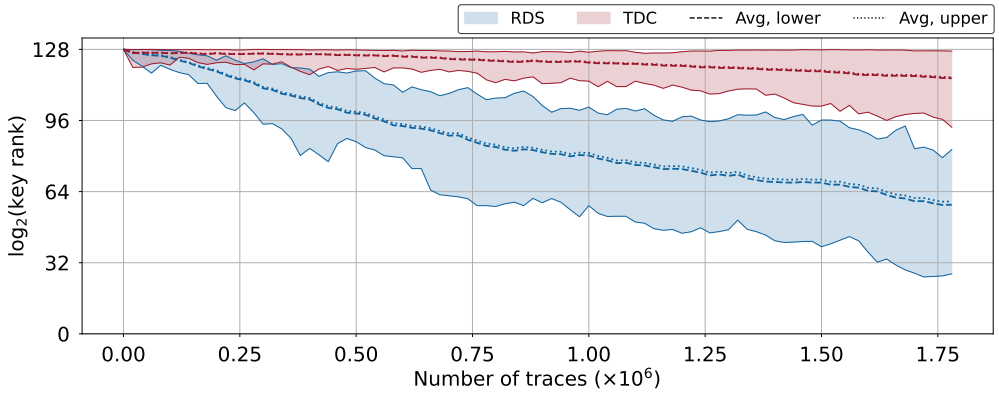


Figure 8.14: Key rank estimation for the TDC and RDS sensors on the Alveo U200 board.

8.5.3 RDS Versus TDC on The Alveo U200 Datacenter Card

To evaluate the RDS sensor on a cloud-scale FPGA, we deploy our system on the AMD Alveo U200 datacenter accelerator card (as explained in Section 8.4.1). Cloud-scale FPGAs are large in size, which makes it more difficult to sense switching activities on the shared PDN [14]. Therefore, we record 1.8 million traces, and repeat the trace collection for each of the keys in Table 8.2. Furthermore, we repeat all the experiments five times.

The results are summarized in Fig. 8.14. Similarly to Fig. 8.11, the average of the lower and the upper bounds of the key rank are shown with dashed lines. The shaded areas correspond to the minimum and the maximum key rank values observed across all the attacks. Again, we see

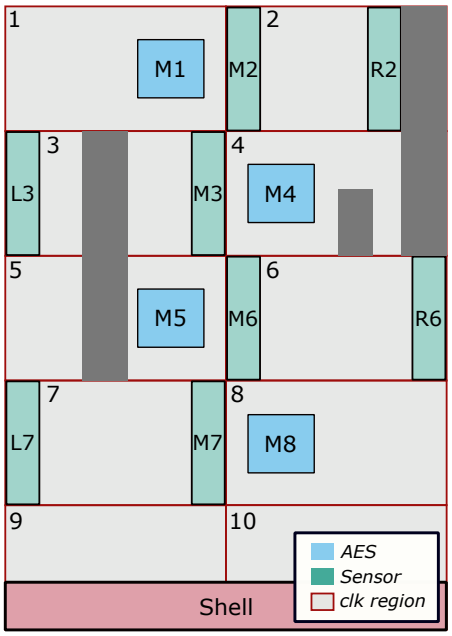


Figure 8.15: Floorplans for the chosen sensor and AES placements.

that the RDS outperforms the TDC, even more clearly than in the experiments discussed in Section 8.5.1. Therefore, a remote adversary equipped with the RDS instead of the TDC can steal a secret with fewer measurements.

8.5.4 Varying the Placement

To further evaluate and compare the RDS and the TDC, we record the power side-channel traces for a number of different sensor and AES placements. Fig. 8.15 provides a conceptual overview of the Sakura-X floorplan, with the sensor and AES positions marked in green and blue, respectively. The FPGA has ten clock regions, out of which the shell, which is always kept at a constant location, occupies the bottom two. In the remaining eight clock regions, we define eight sensor locations (in regions 2, 3, 6, and 7) and four AES locations (in regions 1, 4, 5, and 8). In all experiments, AES and the sensors are in separate regions, in line with the threat model. For a pair of sensor-AES placements, we record 100k traces, once for the key K1 and once for the key K4 in Table 8.2.

In the first set of experiments, we fix the AES to the location labeled as M1 and vary the sensor

placement. In total, we collect the data from 32 experiments (eight sensor locations, two secret keys, and two sensors). Then, we compute the key rank metric and record the number of traces required for breaking all the bytes of the secret key (i.e., the number of traces for which the log(key rank) first time drops to zero). To compare the effectiveness of the RDS with the TDC, we normalize the results and visualize them in Fig. 8.16 (left part). We see that, in most cases, the RDS outperforms or performs equally well as the TDC. On average, the number of traces that the RDS requires to break the key is 60% of the number of traces that the TDC requires (in other words, the attack with the RDS needs 40% fewer traces than the attack with the TDC, to break the entire secret key). Finally, we observe that the number of traces when the log(key rank) drops to zero can vary across the experiments (irrespective of the chosen key). The same can be observed in Figs. 8.11 and 8.14: When the key rank drops to zero and the slope of the curve significantly reduces, likely due to the low SNR of many bits of the secret key, the range of the obtained results widens. Figs. 8.11 and 8.14 also show that the extent of this variability, for both TDC and RDS, reduces for the bits of the secret key which are less impacted by noise and, consequently, broken earlier.

In the second set of experiments, we fix the location of the sensors at M2 and vary the placement of the AES. The results are shown on the right side of Fig. 8.16. Again, in most cases, the RDS requires fewer or an equal number of traces to break the key. In this experiment, the average ratio is 0.76 (i.e., the RDS needs 24% fewer traces than the TDC to break the key, on average). If we take all the experiments into account, then we can say that an attack with the RDS requires 35% fewer traces than the attack with the TDC, for all the bytes of the secret key to be recovered.

Table 8.4 lists the obtained results: the first two rows contain data per sensor and location (averaged across the corresponding experiments with two different keys). The third row aggregates the results per region, both sensors considered. Looking at the last row of data, we can note the correlation between the number of traces to break the key and the locations of the sensor and the AES with respect to one another: When the AES is in the first region, and the sensors change places, the attack is fastest with sensors in the second, third, then sixth,

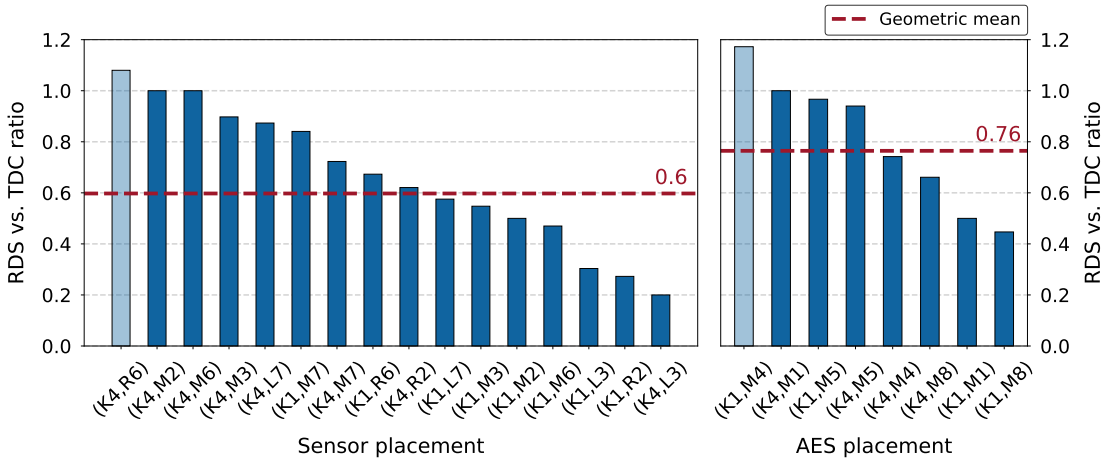


Figure 8.16: Ratio between the number of traces needed to break the key with the RDS and the TDC sensor when varying the placement of the sensor (left) or the AES (right). In dark blue are the results where an attack with RDS is at least as efficient as with TDC. The dashed red line corresponds to the geometric mean.

Table 8.4: Number of traces required to break the full AES 128-bit key, for different sensor placements (AES placed at M1) and different AES placements (sensor placed at M2). The last row shows the average number of traces to break the key per FPGA region, both RDS and TDC sensors considered.

Sensor	Traces to break key ($\times 10^3$)											
	Sensor placement								AES placement			
	M2	R2	L3	M3	M6	R6	L7	M7	M1	M4	M5	M8
RDS	16	18	17	29	46	31	56	59	16	29	91	41
TDC	23	48	71	41	73	39	76	76	23	30	95	77
Average	26.3		39.5		47.3		66.8		19.5	29.5	93.0	59.0

and finally the seventh region; with respect to the AES, regions two and three are close by, while six and seven are further away. When the sensors are in the second region, and the AES changes place, the attack is fastest when AES is in the first, then fourth, then eighth, and finally the fifth region; with respect to the sensors, the first and the fourth region are close by, while the fifth and the eighth regions are further away.

8.6 Chapter Summary

This work presents a novel FPGA-based voltage sensor design, fundamentally different from TDC and RO sensors. Our new routing delay sensor leverages routing resources for sensing

voltage variations. We present three variants of the new sensor: one vertically constrained (VRDS), one horizontally constrained (HRDS), and one free of any placement or routing constraints (RDS). We evaluate the performance of the RDS sensor using the correlation power analysis attack and the key rank estimation metric, in an attack against an AES-128 hardware cryptographic module. The results, computed for a number of different sensor and AES placements on the Sakura-X board, show that the RDS outperforms the TDC: on average, an attack with the RDS requires 35% fewer traces to break the secret key. The observation that the RDS is superior is confirmed with extensive experiments on the cloud-scale FPGA (the Alveo U200 datacenter card). Future work will investigate the avenues for further improvements of the RDS sensor performance.

9 Temperature Impact on Remote Power Analysis Attacks

As we have shown in parts I and II of this thesis, malicious parties can mount remote power SCA attacks using FPGA-based voltage sensors such as TDCs and ROs. However, attackers implement these on-chip sensors using FPGA fabric, which, in turn, is vulnerable to temperature-induced delay changes [91, 142]. Even in the cloud, changes in the on-chip temperature are inevitable: they can happen due to the inertness of the cooling systems or even be a remnant of previous tenant computation on the same FPGA [106]. In general, these temperature changes are considered too slow to impact individual power traces, which are in the nanosecond range. Therefore, in previous work, authors assumed negligible temperature changes during the operation of the victim circuit. However, in the case of ML-based profiling attacks or attacks on protected cryptographic circuits, trace acquisition can take a very long time (even days [143]); therefore, temperature variations unavoidably occur.

As we demonstrate in this Chapter, ignoring the temperature variations and the correct trace acquisition guidelines can lead to erroneous observations, even in the case of remote FPGAs in the cloud. For example, in the presence of temperature variations, ML models, instead of learning the target side-channel leakage, may inadvertently learn temperature effects that are otherwise not present in a real-life setup. Although remote power SCA attacks can succeed at

This chapter is based on the work of a paper published at the 2023 Design, Automation & Test in Europe Conference & Exhibition [35].

different temperatures [139, 144], we find a better understanding of the temperature effects is necessary for future work.

The key contributions of this Chapter are:

1. *Analysis of the temperature impact on sensor measurements.* We show, mathematically and experimentally, that sensor traces suffer from the same drifting offset seen in oscilloscope traces [143], and that the variance of trac:w e samples is temperature dependent. These effects directly impact the measured side-channel leakage and are reflected in the success of a CPA attack on an AES encryption module.

2. *Analysis of the temperature impact on the accuracy of ML-based profiling attacks.* When the power side-channel leakage is limited, and the trace acquisition takes a non-negligible time, we show that incautious trace acquisition can lead to ML models biased by temperature, resulting in misleadingly high accuracy. We analyze this unwanted effect and quantify the impact of correct trace acquisition techniques on accuracy.

In the remainder of the chapter, Section 9.1 discusses the impact of temperature on the sensor output. Section 9.2 describes our experimental evaluation methodology. In Section 9.3, we present the results, while Section 9.4 concludes the chapter.

9.1 Temperature Impact on TDC Sensors

The cell delay impacts circuit performance and limits the maximum operating frequency. In a simple circuit delay model, e.g., the alpha-power law [145], the cell delay is inversely proportional to the drain current I_d of a CMOS transistor. I_d can be expressed as

$$I_d \propto \mu_e(T)(V_{dd} - V_{th}(T))^{\alpha}, \tag{9.1}$$

where μ_e represents the mobility, V_{dd} the supply voltage, V_{th} the threshold voltage, α a small positive constant, and T the temperature [142]. The threshold voltage and mobility

decrease with the rise of temperature, leading to two opposite effects on the drain current: decreased mobility reduces the drain current, while lower threshold voltage increases it. At high voltages, the mobility dominates Equation (9.1), resulting in a delay increase with the temperature. In contrast, at lower voltages, V_{th} becomes the dominant factor, resulting in a delay decrease at higher temperatures. The voltage at which the temperature dependence inverts is called the *crossover voltage*, and it depends on the fabrication technology. This inverse temperature dependence (ITD) phenomenon was thoroughly studied across different fabrication technologies [142, 146].

Let us now formalize the temperature impact on the TDC sensor. As previously explained, the delay of a logic circuit $d(T,V)$ is indirectly proportional to the drain current I_d. Depending on the technology and the voltage V, I_d can be directly or inversely proportional to the temperature. In addition to the carrier mobility and threshold voltage, I_d depends on the Johnson-Nyquist thermal noise, which is constant across the spectrum and increases with temperature [147–149]. Moreover, the sub-threshold leakage current in lower technology nodes represents a source of noise that increases with the temperature [150]. Therefore, we can formalize the delay of a circuit under constant voltage as

$$d(T+\Delta T) \propto d(T) + \Delta d(\Delta T) + \delta(\Delta T), \tag{9.2}$$

where ΔT is the temperature change, Δd is the change in delay, and δ is the thermal noise. When ΔT is positive, and the carrier mobility dominates Equation (9.1), $\Delta d(\Delta T)$ decreases. Otherwise, $\Delta d(\Delta T)$ increases with the temperature when the threshold voltage dominates.

The TDC sensor measures the number of delay elements through which an input clock has propagated during one sampling period t_{sample}. The relationship between the sampling period and the sensor output can be represented as $t_{sample} = O(T)d(T)$, where $d(T)$ is the delay of one delay element and $O(T)$ is the sensor output, i.e., the number of delay elements the input clock has traversed. When the temperature changes by ΔT, the sensor output

becomes

$$O(T+\Delta T) = \frac{t_{sample}}{d(T+\Delta T)} = \frac{O(T)d(T)}{d(T)+\Delta d(\Delta T)+\delta(\Delta T)}. \tag{9.3}$$

Therefore, when the delay increases with the temperature, the clock propagates through fewer elements in the delay line, resulting in lower sensor output. Otherwise, in the ITD case, temperature increase results in higher sensor output.

From (9.3), we find the expressions for the trace DC offset μ (i.e., the mean of all the samples in a trace) and the variance σ^2 (i.e., the dispersion of the values in a sensor trace):

$$\mu = \frac{1}{N}\sum_{i}^{N} O_i(T+\Delta T) \sim \frac{1}{\Delta d(\Delta T)+\delta(\Delta T)}, \tag{9.4}$$

$$\sigma^2 = \frac{1}{N}\sum_{i}^{N} (O_i(T+\Delta T)-\mu)^2 \sim \frac{1}{\Delta d^2(\Delta T)+\delta^2(\Delta T)}. \tag{9.5}$$

Here, N is the number of sensor samples per trace. From (9.4) and (9.5), we can conclude that the trace DC offset is inversely proportional to the logic delay, while the variance is inversely proportional to the delay squared.

9.2 Evaluating The Impact of Temperature

In the context of remote power SCA attacks, we evaluate the impact of temperature on the sensor leakage and the ML-based power side-channel attacks.

9.2.1 Leakage Analysis

In our first experiment, we evaluate how the sensor trace statistics change in the function of the temperature. In a thermal chamber, we start with a constant 40°C, and while recording AES encryptions, we increase the temperature in steps of 5°C up to 60°C. We measure the DC offset and variance of the sensor traces, two critical statistical parameters for SCA attacks.

Sudden ambient temperature variations—and their potential impact on the DC offset and

variance of the sensor traces—could cause degradation in the Pearson correlation coefficient in the CPA attack, resulting in a higher number of traces to break the secret key. Therefore, in our second experiment, using the key rank estimation metric [70], we evaluate how transient temperature changes impact the success of the CPA attack against an AES hardware module. If the impact is significant, the temperature could severely interfere with conclusions between two different experiment runs (e.g., comparing the side-channel security of two cryptographic implementations).

Finally, we analyze the difference in side-channel leakage for traces recorded at different stable temperatures. In a thermal chamber with stable operating temperatures above 35°C, we record ten runs at 40°C, 45°C, 50°C, 55°C, and 60°C. For each temperature, we compute the average number of traces needed to break the key using the CPA attack and the key rank estimation metric [70]. Significantly varying leakage at different external temperatures indicates a potential problem with lengthy experiments: traces acquired over a long time may result in skewed ML models, which are either degraded by the thermal noise or learn the temperature patterns instead of the actual leakage.

9.2.2 ML Accuracy Evaluation

To evaluate the influence of the temperature and the trace acquisition method on ML classification problems, we devise three attack scenarios, i.e., victim workloads, each with a different classification complexity:

- *Hardware workload classification.* The victim contains several hardware modules, with only one running at a time. We choose four encryption cores: AES, PRESENT, KLEIN, and CRYPTON. All implementations are open source and available in the SCABox repository [151]. Using these cores, the attacker can train an ML model to identify the currently running hardware operation. This classification problem is considered easy [105], as the power consumption traces of entirely different hardware cores usually contain particular identifiers.

- *Soft-core CPU workload classification.* The victim is an open-source soft-core RISC-V CPU executing eight code snippets on random data. Each code snippet is intensive in one of the RV32I ISA instruction types: `load`, `store`, `branch`, `arith`, `compare`, `shift`, `logic`, and `jump`. The attacker, having access to the same CPU design and code, profiles the code snippets on many executions with random data inputs and trains a model to identify the one the victim is running. Gobulukoglu et al. showed that distinguishing between different soft-core CPU workloads is a difficult classification problem, and achieved an average classification accuracy of ~ 50% [105].

- *Soft-core CPU instruction subset classification.* Here, the attacker is trying to identify instructions from the subset of the RV32I instruction set. The attacker trains on 10k instruction templates where the target instruction has randomized operands and data, and is surrounded by `nop` instructions. The templated instructions are `jal`, `add`, `xor`, `sll`, `lw`, `sw`, `bne` (not taken), `bne` (taken), and `slt`. Because individual CPU instructions have a short execution time, if the sensor has the same sampling frequency as the CPU, the leakage is limited, and the classification problem is considered hard.

To evaluate the temperature impact on the ML classification accuracy at room temperature, we use two trace acquisition methods for each workload, one incorrect and one recommended for power side-channel evaluation [70]:

- *Consecutive acquisition, room temperature (CR).* In this method—contrary to the recommended trace acquisition guidelines [70]—the traces of each ML class are acquired separately, by first recording all traces of class 1, then class 2, etc. When there is a large number of traces per class, and the trace acquisition takes hours, each class (i.e., specific workload) can be considered as recorded at a distinct temperature.

- *Interleaved acquisition, room temperature (IR).* In this recommended trace acquisition method—commonly used in power side-channel evaluation methods such as the *t*-test [70]—the traces of each ML class are acquired in an interleaved fashion, by recording a single trace of each class, in a randomized order, before continuing the acquisition of

the next group of power traces. For many traces per class, interleaving the traces ensures equal temperature effects across all classes.

To evaluate model robustness and simulate exaggerated temperature changes during trace acquisition, we record two trace sets for hardware workload classification in a thermal chamber:

- *Consecutive acquisition, thermal chamber (CT).* The traces of each ML class are acquired separately. However, to exaggerate temperature variations, each class is recorded at a different but stable temperature: PRESENT at 38°C, AES at 43°C, KLEIN at 48°C, and CRYPTON at 53°C.

- *Interleaved acquisition, thermal chamber (IT).* The trace acquisition is interleaved, spreading the significant temperature changes across all classes. There are four sets of traces recorded at different stable temperatures: 38°C, 43°C, 48°C, and 53°C.

For classification, we implement five ML models commonly used in previous work: convolutional neural network (CNN1 and CNN2, a large and a small model), MLP, LSTM, and random forest classifier (RFC) [96, 105, 144]. Table 9.1 lists their architectural details. When training, we set the batch size to 64 and use the Adam optimizer while monitoring the loss to adapt the learning rate. We train on 90% of the dataset and use the remaining 10% for testing. The test/train split is performed randomly and in a stratified fashion. We train for 50 and 100 epochs for the hardware and software workload classification, respectively.

Table 9.1: Architecture details of the ML models.

Model	Architecture
LSTM	LSTM(100 units) + Dropout(0.2) + Dense(100 units, ReLU) + Dense(Softmax)
CNN1	Conv1D(X filters, kernel size of Y) + MaxPool(2) | (X,Y) = ((32, 12), (45, 10), (64, 8), (128, 4)) + Dropout(0.2) + Dense(100 units, ReLU) + Dense(Softmax)
CNN2	Conv1D(64 filters, kernel size of 10) + MaxPool(2) + Conv1D(64 filters, kernel size of 4) + MaxPool(2) + Dropout(0.2) + Dense(100 units, ReLU) + Dense(Softmax)
RFC	number of estimators = 100
MLP	Dense(X units, ReLU) | X = (250, 350, 150, 50) + Dropout(0.2) + Dense(100, ReLU) + Dense(Softmax)

9.3 Results and Discussion

This section presents the results of the experimental analysis. We first show how temperature changes impact the sensor output and the success of a CPA attack, and then demonstrate the impact of temperature on ML-based profiling attacks.

9.3.1 Leakage Analysis

Following the methodology in Section 9.2, we first evaluate the temperature impact on the sensor traces and the success of the CPA attack. We use a Digilent Basys3 (AMD Artix-7 XC7A36T FPGA): a cost-efficient FPGA platform suitable for potentially damaging thermal chamber experiments. With a single 128-bit TDC sensor (observable line with 128 elements) operating at 200 MHz, we record the power traces of an open-source AES-128 core clocked at 50 MHz [87]. To facilitate comparison between the experiments, we always use the same encryption key and the same set of plaintexts, and keep the sensor calibration constant.

In our first experiment, we record 900k AES traces in the thermal chamber, increasing the temperature over time: from 40°C to 60°C in steps of 5°C. Fig. 9.1 shows the trace DC offset, variance, and on-chip temperature in function of the elapsed time, as represented by the index of the recorded trace. We can observe that both the DC offset and the variance increase with the temperature; hence, the temperature-delay dependence lies in the ITD domain, where the threshold voltage dominates Equation (9.1). This experiment shows that the temperature significantly impacts the TDC sensor output and should not be overlooked when recording traces using on-chip sensors.

Next, we investigate how sudden temperature changes during the TDC sensor trace collection impact the success of the CPA attack. Before the experiment, we place the device in a cool place. Then, we record two datasets: 70k traces at a low temperature and 70k traces where the device is returned to room temperature after 10k traces to warm up gradually. Fig. 9.2 shows the key rank estimation when attacking the key using CPA, in the function of the number

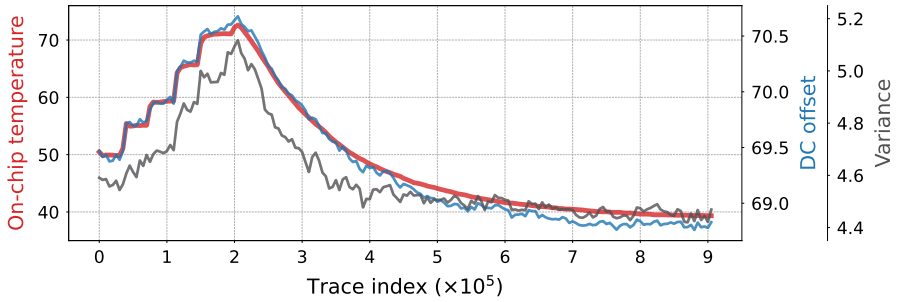


Figure 9.1: The trace DC offset and variance at different on-chip temperatures, in the function of elapsed time, i.e., the trace acquisition index.

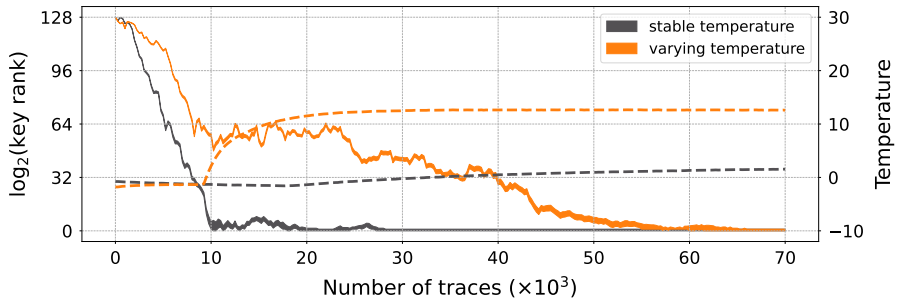


Figure 9.2: Transient temperature impact on the key rank estimation using CPA.

of traces used in the attack. The temperature change has a visible impact: the orange line stops following the gray one and stagnates instead of decreasing. Consequently, the key rank estimation drops to zero later, and the number of traces required to break the key increases. The reason is clear: as the CPA attack is performed using the Pearson correlation coefficient, any change in the trace DC offset and variance directly impacts the attack's success. This result shows that in security-sensitive experiments, such as comparing the side-channel security of cryptographic designs, it is important to consider environmental temperature changes and follow correct trace acquisition guidelines that minimize their impact [70].

To evaluate the transient temperature effects on the RDS sensor introduced in Chapter 8, we record 70k traces using a fixed key while keeping the calibration constant throughout the trace acquisition. First, we record the baseline results at a stable room temperature. Then, we turn to the following three temperature-varying scenarios: one temperature increase and two temperature drops. In all the experiments, we first record 9k traces at room temperature and only then start warming up or cooling the device.

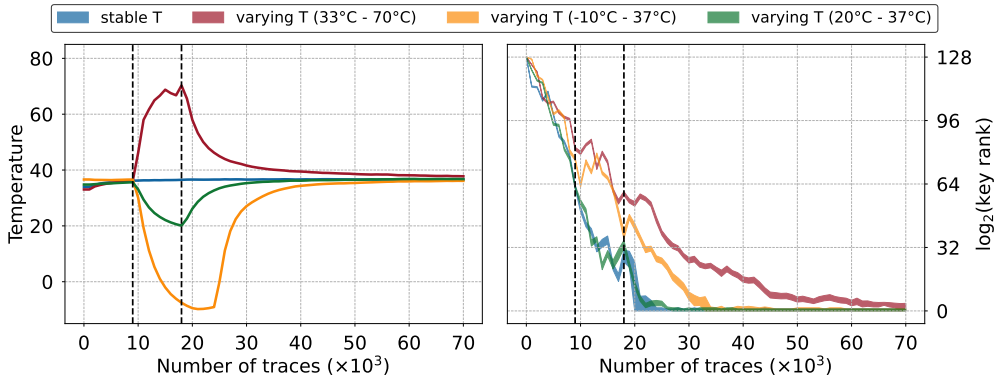


Figure 9.3: Key rank estimation for the RDS at different transient temperatures.

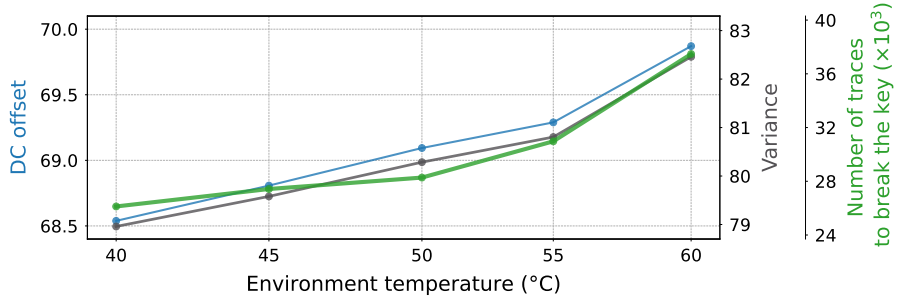


Figure 9.4: Impact of the environment temperature on the sensor trace DC offset, variance, and the number of traces needed to break the key using CPA.

Fig. 9.3 shows how the transient temperature changes impact the success of the attack in case of RDS. We can observe that the slight temperature drop of ~17°C (green line) does not significantly impact the attack success when compared to the baseline. By contrast, cooling the FPGA by ~47°C for 15k traces (yellow line) increases the attack effort to break the entire key by the additional 10k–15k traces. Finally, heating the FPGA by ~37°C (red line) has an even more pronounced impact on the success of the attack. From Fig. 9.3 we can observe that the impact of temperature on the RDS sensor is similar to that on the TDC sensor. Importantly, despite having a more sensitive calibration process, the RDS sensor remained calibrated even under significant temperature changes, and the clock edge did not drift away from the observable time window.

Additionally, we examine the impact of stable temperature on the attack's success, for the TDC sensor traces. For each temperature outlined in Section 9.2.1, we record ten experiment

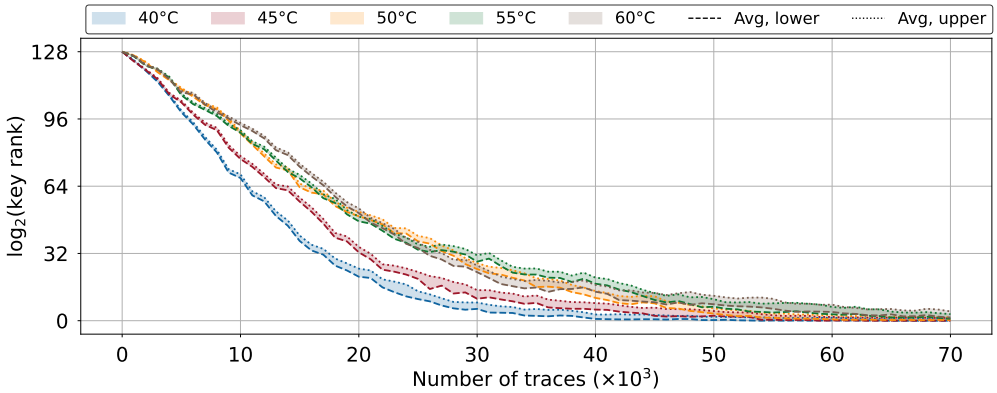


Figure 9.5: Key rank estimation for the RDS at different, but stable, environment temperatures.

runs in the thermal chamber and compute the average trace DC offset, variance, and the number of traces required for a successful attack (when the key rank estimation metric first drops to zero). Fig. 9.4 shows the results, averaged across ten runs. We can observe that more traces are required for a successful attack at higher temperatures and that thermal noise, more pronounced at higher temperatures, can increase the attack effort, resulting in approx. 1.4× more traces to break the key. Although the trace variance increases, the quality of sensor traces degrades at higher temperatures because the thermal noise becomes the dominant factor.

In our final set of experiments, we evaluate the impact of different stable environment temperatures on the RDS sensor traces. To this end, we record the traces with the FPGA in a thermal chamber. We set the chamber temperatures to 40, 45, 50, 55, and 60°C. At each temperature, we acquire 70k traces using a fixed key, while keeping the calibration constant and repeat the trace acquisition ten times. Fig. 9.5 shows the average upper and lower bounds of the key rank estimation metric. We see that, as with the TDC, at higher environmental temperatures, the attack effort increases due to thermal noise.

In the case of real datacenter FPGAs, the on-chip temperatures often exceed 80°C [106, 152]. In this Chapter, the results shown in Figs. 9.4 and 9.5 indicate that the thermal noise causes an increased attack effort at higher temperatures. Therefore, the thermal noise can potentially act as a *free* complementary technique against power analysis attacks in multitenant FPGAs, further increasing the effort needed to break the key successfully. For example, mitigation

techniques such as masking or active wire fences outlined in Chapter 7, could become more efficient at the higher temperatures in datacenter FPGAs.

9.3.2 ML Accuracy Evaluation

After demonstrating that the temperature can significantly impact the success of a remote CPA attack, we next evaluate the temperature impact on two common ML-based profiling attacks: hardware and software workload classification.

Hardware workload classification

For this experiment, we use SCABox [151], an open-source tool for side-channel evaluation on Digilent ZedBoard (AMD Zynq-7000 FPGA). We instantiate four cores working at 10 MHz: AES, PRESENT, KLEIN, and CRYPTON. As the AES is considerably larger than other cores, we replicate the other cores eight times to obtain hardware workloads of approximately the same size and avoid classes with significantly different features. The SCABox instantiates eight TDC sensors, operating at 200 MHz.

For each core, we record 10k traces for IR, CR, IT, and CT datasets, and train five ML models (Section 9.2.2). In all cases, our models achieve 100% accuracy, showing that hardware workload classification is an easy problem and that the temperature does not impact the accuracy.

To evaluate the robustness of the trained models, we validate them using traces not seen during training and testing (the validation dataset size is 10% of the corresponding dataset). Fig. 9.6 shows that the models trained on the interleaved traces generalize well, and achieve high validation accuracy when tested on all the other datasets. As interleaved traces contain data samples from a wider range of temperatures, they help build more robust and generalized models.

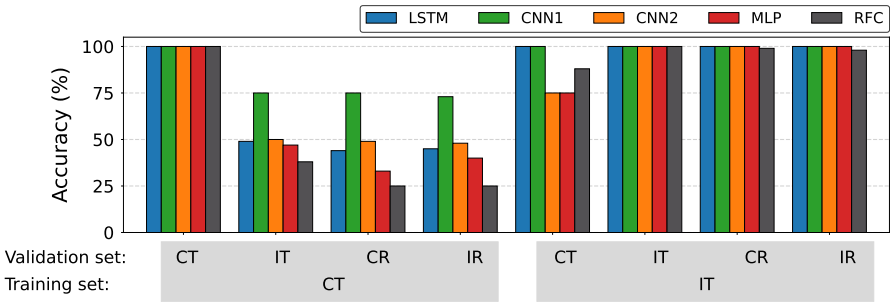


Figure 9.6: Impact of the trace recording methodology on the ML model accuracy, in the case of hardware workload classification.

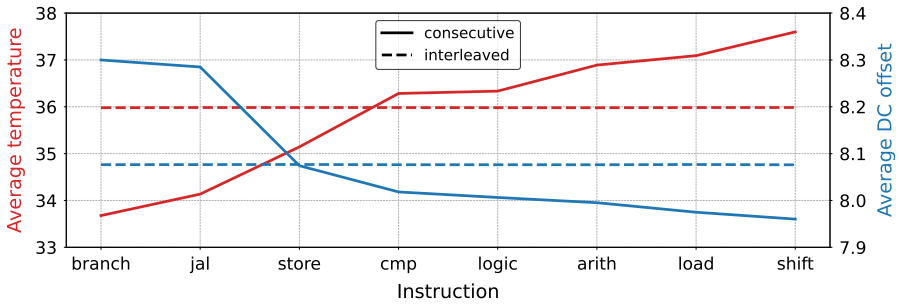


Figure 9.7: Impact of the temperature on the average DC offset of each ML class, in the case of code snippet power side-channel traces.

Software workload classification

In this experiment, we use a high-end FPGA to evaluate the temperature impact on cloud FPGAs. On an AMD Alveo U200 datacenter accelerator card (UltraScale+ XCU200 FPGA), we place a PicoRV32 CPU [95] and 30 16-bit TDC sensors running at 320 MHz.

We start by reinvestigating the temperature impact on the DC offset of the sensor traces. We record 10k traces for each of the eight code snippets described in Section 9.2.2. To reduce noise, instead of recording one execution trace for the given code and data it operates on, we record and average 1k traces. Fig. 9.7 shows the average DC offset and the temperature of the traces of each class, for consecutive and interleaved trace acquisition. First, we can observe a direct temperature-delay dependence, because the sensor output drops as the temperature increases. Second, the DC offset of the traces recorded in the interleaved fashion does not correlate with the temperature, because the temperature variations impact all classes equally.

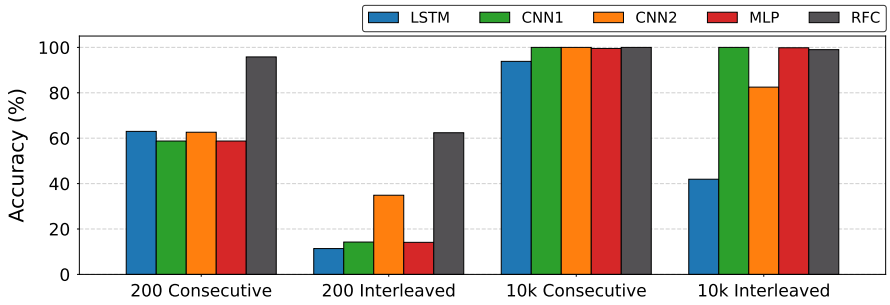


Figure 9.8: Impact of the dataset size on the accuracy of the ML models, in the case of soft-core CPU workload classification.

Next, we examine if the temperature impact on the classification accuracy changes with the dataset size (i.e., the difficulty of the classification problem). Using the acquired traces of the eight code snippets, we train the ML models twice: once with all 10k traces per code snippet and once with only 200 randomly selected traces per code snippet. We repeat the training with five random seeds and average the results, for more general conclusions. In addition to having randomized training parameters, the smaller dataset results in a unique random subset for each seed. The results in Fig. 9.8 show that, when using the entire dataset, both interleaved and consecutive datasets result in good accuracy (though lower for LSTM and CNN2 as they fail to converge for some seeds). However, training on incorrectly acquired traces results in considerably higher accuracy with the reduced dataset size, as the ML models learn the temperature effects instead of the leakage.

Let us now look at the training accuracy evolution. We record 10k traces for the nine instruction templates described in Section 9.2.2, interleaved and consecutive. Fig. 9.9 shows the results. Once again, we see that for complex classification problems (here, limited leakage of a single CPU instruction), incorrectly (i.e., consecutively) recorded traces mislead the ML models into learning the temperature effects instead of the actual leakage. In contrast, training on traces recorded in an interleaved fashion results in a lower, but more realistic classification accuracy.

Finally, we evaluate if common preprocessing techniques can alleviate the unwanted temperature effects from recorded traces. Table 9.2 shows the ML model accuracy when DC removal, filtering (high-pass with a 5 MHz cutoff), and normalization (MinMaxScaler) are applied. We

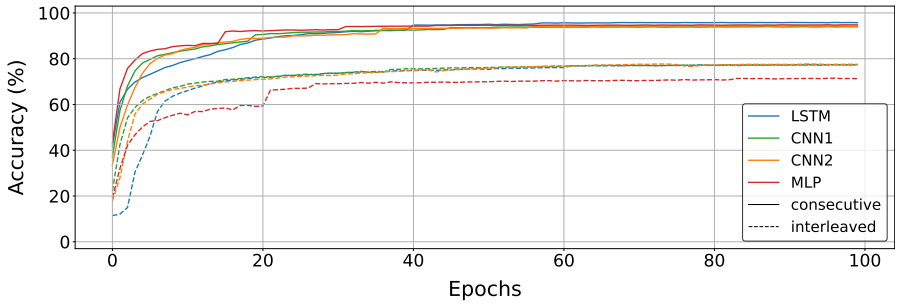


Figure 9.9: Impact of trace acquisition on the evolution of the training accuracy, in the case of soft-core CPU instruction classification.

Table 9.2: Accuracy of ML models with and without preprocessing.

Dataset	Preprocessing	Test accuracy (%)				
		CNN1	CNN2	LSTM	MLP	RFC
Interleaved	None	77.2	78.4	78.2	72.5	46.0
Consecutive	None	94.4	94.9	96.4	95.4	92.0
Consecutive	DC removal	94.5	94.6	92.4	94.7	81.0
Consecutive	Filtering	94.6	94.6	96.1	95.5	93.0
Consecutive	Normalization	98.4	98.2	98.0	98.3	92.0

see that neither of the three approaches significantly impacts the model's accuracy. Therefore, we can conclude that the temperature impacts not only the sensor measurements but possibly the power side-channel leakage generated by the victim.

Our results demonstrate that the impact of temperature on the TDC sensor measurements is important because it can lead to incorrect conclusions if trace acquisition guidelines are not followed. In the case of ML-based profiling attacks specifically, it can skew the accuracy and show better-than-expected results. Interleaving the trace recordings—the proper method of acquiring power traces—is necessary for spreading the temperature effects equally across the dataset.

9.4 Chapter Summary

Varying environmental temperature impacts power side-channel traces recorded with TDC sensors. Our findings confirm that the temperature influences the sensor output and that this dependence varies across different FPGA families. We demonstrate that the temperature

changes during trace acquisition impact the attack's success, as CPA requires more traces to break the AES encryption key if the temperature increases. Further, due to temperature, the trace acquisition method can significantly impact the robustness and the generality of models in ML-based profiling attacks. We demonstrate that for easily distinguished classes, (i.e., datasets with models converging to a 100% accuracy), trace acquisition has little to no impact on the final accuracy. However, for harder classification problems, ML models of incorrectly recorded traces learn temperature variations instead of leakage, resulting in misleadingly higher accuracy. Our research highlights the importance of adhering to appropriate trace acquisition guidelines, even in the context of shared FPGAs, if robust models and a realistic measure of classification accuracy are to be obtained.

Part IV

Related Work and Conclusions

10 Related Work

In this Chapter, we present the related work most relevant to our research. Related work includes monitoring of security primitives, remote power analysis attacks, voltage-drop sensor architectures, and active fences.

10.1 Monitoring of Security Primitives

Most of the previous work on monitoring security primitives targets resistance against fault attacks or tolerance to errors. The most natural example of such monitors is the error correction circuitry included in hardware and software implementations of cryptographic primitives. A typical example is parity codes applied to the AES algorithm [153]. However, albeit adapted to security primitives, this functionality monitoring is still an application of classical error-correcting codes.

The first security primitive monitored with a dedicated hardware watchdog is a true random number generator (TRNG): to ensure that the sequences produced by TRNGs respect strict statistical properties, Yang et al. [75] designed on-the-fly statistical tests suitable for hardware implementation on FPGAs. Checking the statistical properties of the results has also been proposed to counteract fault attacks on other cryptographic primitives, such as lattice-based ones. Howe et al. [76] used a battery of statistical tests to verify if the distribution produced

by the sampler is the expected one (Gaussian or binomial). These statistical tests have been designed for FPGAs and included as hardware monitors in the samplers in lattice-based algorithms.

While valuable, these monitors focus only on *active* attacks. However, evaluating the leakage exploitable with *passive* attacks using monitors is a task largely unexplored in literature. The most relevant work related to our contribution is the one of Sonar et al. [81]: they implement a hardware watchdog to measure the side-channel leakage of a cryptographic primitive. Our work extends their initial ideas, providing various novel contributions. Instead of using a model of leakage, which requires training and does not accurately model unexpected changes in the environment, we measure the actual leakage from the device (we design a sensor and integrate it into the user design as a part of our monitor). Thus, we replace the training phase with a simple calibration procedure, which tailors our monitor to different cryptographic algorithms and implementations. By performing the calculation on real power traces, we remove the strong assumptions of a constant mean and a negligible variance of the fixed plaintext traces used for the t-test, which could lead to erroneous results. Finally, we improve the computation of the t-test statistic since we measure leakage in multiple trace samples instead of in only one.

10.2 Power Analysis Attacks on Shared FPGAs

Zhao et al. characterized the RO and TDC voltage monitors on an AMD Zynq-7000 SoC and successfully used them in an SPA attack against a collocated RSA cryptomodule and RSA exponentiation running on the ARM processor. The traces they recorded had visibly different amplitude and duration, depending on whether the processed RSA key bit had a binary value of `0` or `1`. In a concurrent study, Schellenberg et al. demonstrated a CPA attack in which, instead of an oscilloscope, they used a TDC sensor collocated with an AES crypto module on an AMD Spartan-6 FPGA [47]. In our work, we refined the TDC sensor design and ported it on an Amazon EC2 F1 cloud instance (AMD Virtex UltraScale+ FPGA) to showcase a successful

CPA attack against a 128-bit AES module.

Similarly to Zhao et al., Gravellier et al. targeted AMD Zync-7000 SoC; with correlation power analysis, they recovered the secret key of a bare-metal implementation of Tiny AES and OpenSSL AES [48]. These seminal works showed that physical access is no longer required for power side-channel attacks and that shared FPGAs are vulnerable to power analysis attacks. However, unlike our work on the leakage of soft-core CPU processors, all the attacks mentioned above are statistical-based attacks that depend on thousands or millions of victim execution traces for a successful attack.

Another class of power side-channel attacks on shared FPGAs concerns profiling and reverse engineering of another common FPGA workload: neural network accelerators. Given the size of a neuron and the network as a whole, the change in network topology or size can have a considerable (i.e., lasting and distinguishable) impact on the power supply voltage. In a remote attack scenario involving a shared FPGA, it has already been shown that an adversary can infer the activation function, the weights, the number of neurons and layers, the width and depth of convolutional layers, the width of pooling layers, filter sizes, and the stride of convolutional and pooling layers [15, 96–98, 144]. Unlike statistical-based attacks, these profiling attacks require a small number of victim execution traces. However, since the victims are ML accelerators occupying a large portion of the FPGA logic, a good SNR results in easily exploitable side-channel leakage and high attack accuracy. Our work analyzes the leakage of a soft-core CPU, which is a significantly smaller victim than ML accelerators.

In the context of side-channel attacks on ML accelerators, the work of Tian et al. [15] is most relevant to us, as the authors exploit instruction-level leakages of an ML accelerator. The authors use TDC sensor traces to attack a versatile tensor accelerator (VTA) on an AMD Zynq-7000 FPGA. VTA is a generic and customizable deep learning accelerator, which realizes an ML model as a set of VTA instructions and collates them into instruction groups, each containing a mix of `LOAD`, `GEMM`, `ALU`, or `STORE` instructions. Firstly, Tian et al. have observed that all TDC traces recorded during 25,000 clock cycles (120 MHz clock frequency) for GEMM,

ALU-Add, and LOAD-and-STORE unit tests have distinctly different shapes, allowing SPA attacks. Additionally, SPA on the traces recorded during GEMM instructions allows the reverse engineering of the instruction parameters by finding the time interval between adjacent peaks, counting the number of peaks, and measuring the amplitude of the voltage drop in the sensor trace. In our work, we analyze the leakage of a soft-core CPU, considerably smaller than VTA. Moreover, the traces corresponding to the CPU instructions are orders of magnitude shorter in time, and many CPU instructions give remarkably similar sensor trace waveforms. Consequently, more than a visual analysis of the traces is required. To analyze the instruction-level leakages, we deploy different ML classifiers.

Instead of assuming that the victim is a cryptographic core or a neural network, Gobulukoglu et al. used TDC sensor traces to determine whether a cotenant application is present and what type of application it may be [105]. On an AMD Zynq-7000 FPGA, they deployed one TDC sensor and nine scenarios: one without any cotenant, one power-hungry tenant, and others covering several implementations of AES and PRESENT (a custom IP core, software running on Microblaze, ORCA, and PicoRV *soft-core* processors). They collected 250 sensor traces for each scenario, transformed them into two-dimensional images, and trained the ResNet50 classifier to predict workloads. The reported classification accuracy ranged between 33% and 99%, with an average of around 70%. The lowest classification accuracy was reported between AES and PRESENT running on the same type of soft-core CPU. The highest was achieved when distinguishing between very different implementations: an AES core and a soft-core CPU. It is worth noting that the soft-core processors were running at 5 MHz, while the sensor was clocked at 100 MHz. In this work, we target a considerably more challenging classification problem; not only is our target soft-core CPU working on a higher frequency, but the information required to determine the instruction the CPU executes is also contained in a significantly smaller number of sensor samples (shorter trace). Nevertheless, we show that instruction leakage in the power traces is sufficient to achieve an accuracy higher than 80%.

10.3 Power Side-Channel Disassembly Attacks

A body of research covers power side-channel attacks on cryptographic computations, whether executed by a CPU or implemented as an ASIC or FPGA circuit. Similarly, researchers investigated whether power side-channel or electromagnetic side-channel emanations can be used to determine the instructions executed by a CPU.

Vermoen et al. were the first to recover the code executed on a Java 4 MHz SmartCard, by correlating the average power traces (recorded with a 200 MS/s oscilloscope with a set of templates [113]. Instead of power, Strobel et al. measured the EM emanations of an 8-bit PIC16F687 MCU, running at 4 MHz [115]. The first to use ML methods, the authors deployed LDA coupled with the k-NN algorithm and obtained instruction classification accuracy of 96% and 87% for the test and the real codes, respectively. Cristiani et al. focused on the instruction fetch stage of a 14-bit PIC16F15376 MCU operating at a significantly higher frequency than in previous work: 20 MHz [112]. To compensate for a higher CPU frequency, they used a 10 GS/s oscilloscope, averaged 1,000 traces per template to reduce noise, and were the first to record EM side-channel traces at multiple chip locations. Using LDA for dimensionality reduction and a QDA classifier, they reported 95% instruction recognition accuracy. Park et al. targeted an 8-bit ATmega328p MCU (16 MHz, two-stage pipeline) and recorded the power side-channel traces with a 2.5 GS/s oscilloscope [116]. The authors were the first to deploy frequency analysis for disassembly and used CWT to find the differences between the instructions not observable in the time domain. Park et al. then applied Kullback-Leibler (KL) divergence to identify important features, PCA for dimensionality reduction, and a hierarchical classification approach. On the test codes, they reported 99% instruction opcode recognition accuracy. Krishnankutty et al. were the first to find the instruction execution boundaries in a side-channel trace of an MSP430 MCU [114]. Their hierarchical classification based on SVM resulted in 86% opcode recognition accuracy.

Common to the above works is that the victim CPU was running at frequencies orders of magnitude lower than the sampling rate of the oscilloscope for measuring the side-channel traces.

On FPGAs, voltage-drop sensors cannot reach the sampling frequencies of an oscilloscope. In our experimental setup, in one case, only four sensor samples were available per one CPU clock cycle, whereas in the other, only one sensor sample was available per CPU clock cycle. Additionally, unlike power disassembly attacks which depend on only one source of power traces, i.e., the oscilloscope, our work leverages multiple remote on-chip sensors to increase the signal. Despite that, we show that ML methods used in power disassembly attacks are not optimal for remote leakage evaluation. We present new DL time-series classifiers that can determine the type of instruction executed with an accuracy higher than 80%. Our work not only presents new DL methods beneficial for future power side-channel disassembly attacks, but also shows the need to deploy countermeasures against power disassembly attacks, even in a remote scenario.

10.4 Power Wasters

As another security vulnerability in shared FPGAs, researchers investigated the consequences of excessive power wasting. Gnad et al. [49] demonstrated that a large number of ROs activated in a repetitive pattern could reset the FPGA, resulting in a DoS attack. Mahmoud and Stojilović [51] showed that careful activation of ROs could introduce faults in neighboring circuits. Consequently, some cloud service providers, such as Amazon AWS, do not allow combinational loops in FPGA designs. As an alternative, researchers investigated power wasters free of combinational loops [53, 136, 154], typically less effective than ROs. In the active fence scenario, power wasters generate noise to reduce the SNR; the noise must not be excessive, as the victim needs to operate correctly in its presence. If the fence is deployed by cloud service providers, both synchronous and combinational power wasters can be considered [59], and their number must be limited.

The most relevant power wasters to our work are the NAND-based ROs, used to build and evaluate an active fence in previous work [59]. The NAND-based ROs consist of a single LUT, programmed as a NAND gate, where the output is connected back to one of the two inputs,

while the free input acts as an enable signal. When the enable signal is active, the RO oscillates at a high frequency and draws current. An enhanced version of a NAND-based RO, ERO, was later proposed by La et al. and used to perform a remote DoS attack [154]. In the case of EROs, the output of one RO is connected to an unused LUT input of another nearby RO, thereby enhancing the effects of the switching activity thanks to the capacitance of the local interconnect. As we will later show, unlike EROs, our wire-based wasters primarily use longer, global wiring to enhance the effects of the RO switching.

10.4.1 Active Fences

Following the discovery of remote power SCA attacks in multitenant FPGAs, Krautter et al. [59] presented a new mitigation technique: active fencing. This hiding countermeasure employs ROs to generate noise in the PDN and, consequently, reduce the SNR measured by the attacker. The fence was designed as a set of RO banks (where each contains an equal number of ROs and can be independently enabled) placed between the victim and the attacker. The output signals of a PRNG or a TDC were used to control the fence activation. The result was a significant increase in the number of traces needed for a successful attack. Such a fence is easily portable and does not depend on the underlying victim design. However, the increased security comes at the cost of FPGA resources used by the fence. In previous work, the fence was dimensioned to occupy approximately the same number of slices as the victim (i.e., incurring 100% area overhead) [59]. We adopt a similar approach when comparing the efficiency of the wire fence to previous work: we implement a fence having the same type of ROs (NAND-based), use a PRNG to independently enable parts of the fence, and size the fence to occupy approximately the same number of resources as the victim to be protected.

10.5 Voltage-Drop FPGA Sensors

In literature, the TDCs and the RO-based sensors have been used in the context of power side-channel attacks [13, 14, 43, 155] and crosstalk side-channel attacks [44, 156]. They have

been shown to be effective as covert communication receivers, where the sender is the FPGA, CPU, or even GPU sharing the common power delivery network [46]. The TDC sensors have been used in a correlation power analysis attack on Amazon AWS F1 instances [14] and to recover the inputs to a neural network deployed on the same cloud FPGA instances [138]. On-chip sensors were shown effective in capturing the side-channel leakage across integrated circuits sharing the same board [47] and against a CPU within the same system-on-chip [155]. As ring oscillators are less effective for side-channel attacks [157] and ring-oscillator structures are relatively easy to detect [132, 133], we focus on the TDC-based sensors in this work.

Zick et al. [11] were the first to introduce a phase shift (e.g., with a phase-locked loop) between the clock propagating through the delay line and the clock sampling the output register. This design choice allowed for reducing the length of the observable (i.e., tapped) part of the delay line and helped fit the TDC inside an FPGA column. In the early TDC designs of Zick et al. [11] and Gnad et al. [49], transparent latches were used to capture the sensor samples. In later works, latches were replaced by flip flops [14, 138, 158].

Various implementations of clock phase shifting have been proposed. To reduce jitter, Gnad et al. replaced the phase-locked loop with a chain of LUTs, latches, and fine carry elements [65]. They implemented a self-calibrated sensor (i.e., with the calibration in hardware): while monitoring the sensor output, the number of coarse delay elements (LUTs, latches) or fine delay elements (carry) is tuned as long as the desired calibration is not achieved. Udugama et al. followed this example and implemented another variant of a voltage-fluctuation sensor calibrated in hardware. For tuning the initial delay, Udugama et al. chose the adjustable input delay FPGA elements (IDELAY). In our work, we perform phase shifting using the coarse and fine delay elements, similarly to Gnad et al. [65]; however, we control the calibration from the software because it allows us to have much better control and flexibility, which is desirable when experimenting.

Recently, Udugama et al. proposed VITI [139], a delay-line-based sensor built with the goals of minimizing the use of logic resources and escaping from the strict placement constraints of

the carry chains in TDCs. In VITI, the delay line is implemented using LUTs as delay elements. Additionally, LUTs are freely placed, and the connections between them are freely routed. By saving the resources and by using the LUTs instead of the carry-propagation logic, the resolution and the sensitivity of the sensor were sacrificed. Yet, the authors have shown that an adversary armed with VITI can, sometimes partially and sometimes fully, break the secret key of an AES-128 hardware module. Our proposed routing delay sensors are very different because they primarily use the FPGA routing resources to sense the on-chip voltage variations. The third variant of our sensor is free of any placement and routing constraints, similar to VITI, making it simple to deploy and difficult to detect. And this last variant, on average, has better sensitivity than the TDC and allows breaking the secret key faster. In this work, we implement one TDC [65] and one VITI [139] and use them as references for comparison with our implementations of routing delay sensors.

10.6 Temperature Effects

Researchers often leverage ML models for remote power SCA attacks. Usually, attackers record one set of sensor traces, randomly splitting it into training (to profile the victim and train the ML model) and test traces (to evaluate the final accuracy). Gobulukoglu et al. used short-term Fourier transform and image classifiers to distinguish between cloud FPGA workloads [105], achieving high accuracy of 97.6%. They indicated that identifying soft-core CPU applications is challenging, resulting in comparatively low accuracy of approx. 50%. Zhang et al. showed that ML models could predict hyperparameters of a DNN accelerator with an accuracy of up to 100% [96]. In addition, Meyers et al. found that ML models can recover neural network folding [144].

To show model robustness, Meyers et al. [144] trained the model on traces recorded at room temperature and tested it on traces recorded at different ambient temperatures. The reported high accuracy (almost 100%) indicates that the traces contained substantial side-channel leakage, independent of temperature. We take a step further and show that when the leakage

is limited (e.g., a small victim circuit), the temperature impact on the classification accuracy can be significant.

As the logic delay depends on both the voltage and the temperature, when voltage is constant, FPGA sensors can be used to measure temperature variations. Zick et al. [150] used ROs distributed across the FPGA die to sense temperature variations, and modeled the temperature as a function of the supply voltage and the RO frequency. They found that, for a Xilinx Virtex-5 FPGA (manufactured in 65nm technology), the RO circuit delay increases with the temperature. However, when comparing their results with even earlier work, they noticed that the temperature impact grows weaker in newer FPGAs, concluding that with newer technologies and voltage scaling, the temperature dependence will likely reach the crossover voltage point. Tian et al. used the ROs to sense the FPGA temperature and create a temperature-based covert communication channel [159]. They experimented with an Intel Stratix V FPGA (28nm technology) and observed that the RO delay increases with the junction temperature.

11 Conclusions and Future Work

Today's heterogeneous systems rely on hardware acceleration to improve performance and energy efficiency. The reconfigurability and parallel architecture of FPGAs have led to their integration into various remotely-accessible systems, from embedded and cyber-physical systems to datacenters. However, remote access to low-level FPGA hardware raises several security issues, primarily because of the electrical-level coupling between the FPGA logic and the PDN. In this thesis, we focus on power side-channel leakage, and using FPGA-based voltage sensors, we investigate methods to evaluate, exploit, and hide power side-channel leakage in remote FPGAs.

In the first part of the thesis, we presented the first built-in system for remote leakage evaluation based on Welch's t-test, with results comparable to those obtained using state-of-the-art lab equipment. Our system allows, for the first time, a real-time assessment of power side-channel leakage during the device operation in the field, moving the leakage evaluation boundary from the lab environment to devices in the field. Our system is complementary to tamper and attack detection mechanisms, and tamper detection methods can trigger our leakage evaluation test when the device is compromised. Our work paves the way for future system designers to design more advanced leakage evaluation systems, as we have shown that FPGA-based sensors are sufficient for leakage evaluation. For example, system designers can leverage the additional CPU computing power in FPGA-based SoCs for advanced statistical

analysis.

In the second part of this thesis, we investigated the limits of power side-channel leakage in multitenant FPGAs. We demonstrated the first remote power side-channel attack on Amazon F1 instances, revealing that even large datacenter-scale FPGAs are susceptible to power analysis attacks. Our findings underscored the relevance of remote power analysis attacks, contributing to the discourse and research surrounding potential countermeasures in cloud environments. Furthermore, we showed that the effort required for a power analysis attack increases on larger datacenter FPGAs, necessitating two to three orders of magnitude more traces compared to lower-end FPGAs. Our research has since inspired a growing body of work, demonstrating attacks not only on local FPGA boards but also on datacenter-scale FPGAs.

Our research on soft-core CPU instruction-level power side-channel leakage analysis revealed that exploitable leakage extends beyond statistical attacks on cryptographic circuits and profiling attacks on ML accelerators. Our findings underscore the presence of power side-channel leakage in small soft-core CPUs in multitenant FPGAs. Although instruction-level leakage exists, future work has yet to demonstrate a full exploit or a code extraction attack. Since our work shows that averaging is crucial for high accuracy, exploring methods for detecting branches to fold loops and average power traces should also be analyzed in future work. Additionally, proper methods for template window alignment with low SNR are needed for code extraction to be possible. Nevertheless, demonstrating the existence and potential exploitability of power side-channel leakage across various devices—from small soft-core CPUs and encryption cores to large ML accelerators—highlights the need for suitable mitigation strategies in FPGA-based systems.

Prior to the widespread deployment of multitenant FPGAs, it is imperative to establish effective security solutions for power analysis attacks. As a step towards this goal, our research introduces a universal hiding technique against remote power analysis attacks: active wire fences. Our results show a six-fold increase in power side-channel security compared to RO-based fences, without incurring additional resource overheads. Future work can investigate

ways to implement even more effective power wasters. For example, sinks can also be ROs instead buffers. The RO sinks can then drive long wires back to the source RO to increase the power-wasting capabilities further. Moreover, future work can further investigate the impact of wire wasters on congestion and routability in large designs, and add support for including active wire fences after the implementation step, using tools such as RapidWright. Overall, active wire fences are a versatile protection method that cloud service providers or users can implement without necessitating modifications to the protected device. More importantly, their application-agnostic design enables seamless integration with other mitigation techniques—such as masking schemes or bitstream scanning tools—paving the way to more secure FPGA-based systems in future research.

In the last part of our thesis, we explored new voltage sensor architectures on shared FPGAs, and presented the first stealthy, routing-based FPGA sensor that outperforms the state-of-the-art TDC. RDS is free of placement and routing constraints, and thus easy to deploy and port between FPGA families. More importantly, detecting and eliminating RDS with bitstream checking tools is significantly more challenging than the TDC, as it consists of entirely valid FPGA constructs. RDS demonstrates that bitstream checking is a cat-and-mouse game, where preventing state-of-the-art sensor architectures does not guarantee security, as electrical-level coupling means new sensor architectures can be designed to bypass the tools. Therefore, a more holistic approach for guaranteeing security is needed in future work, potentially combining multiple approaches proposed in previous work, including hiding techniques, bitstream checking, PDN design, and others [33].

Finally, this thesis systematically evaluated how temperature effects can impact FPGA-voltage sensors and remote power analysis attacks. We show the importance of adhering to appropriate trace acquisition guidelines, even in shared FPGAs, if robust ML models are to be obtained. Our work can guide researchers in better interpreting their results and obtaining more realistic attack results, while future work can focus on evaluating the impact of aging and other external effects on the sensors.

This thesis provides insights into the possibilities and pitfalls of including FPGAs in remote heterogeneous systems, focusing on power side-channel security. Our work on cyber-physical and cloud FPGA systems shows that low-level FPGA logic can be effectively used to evaluate, exploit, and hide power side-channel leakage in remote FPGAs. Perhaps the most important insight of this thesis is the importance of a more comprehensive approach to designing secure FPGA-based systems. Current countermeasures are either application-specific or address only a subset of vulnerabilities, lacking a holistic approach to electrical-level security. Our work has uncovered many issues that future research should address: low generality, in-depth exploration of interoperability and its implications, and the general absence of experiments in a real cloud setting. Finding the proper countermeasure—or a combination of them—remains an open problem. The best approach to solving it, we think, is through collaboration between and contribution from all the involved parties: researchers (by uncovering new attacks and developing new protection strategies), FPGA vendors (with enhanced FPGA architecture and tools), and the CSPs (by preventing or locating the attackers, while allowing everyone else to deploy their protections, if they wish). This thesis is an essential step in this direction.

Bibliography

[1] J. Hennesy and D. Patterson, "A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development," Turing Lecture at ISCA 2018, 2018, https://iscaconf.org/isca2018/docs/HennessyPattersonTuringLectureISCA4June2018.pdf.

[2] "Azure cloud services," https://learn.microsoft.com/en-us/azure/virtual-machines/sizes-field-programmable-gate-arrays, Microsoft, 2023.

[3] *Amazon EC2 F1*, Amazon AWS, 2019, https://aws.amazon.com/ec2/.

[4] Alibaba, "Compute optimized instance families with FPGAs," alibabacloud.com/help/doc-detail/108504.htm, Alibaba.

[5] D. Kohlbrenner, S. Shinde, D. Lee, K. Asanovic, and D. Song, "Building open trusted execution environments," *Proceedings of the IEEE Symposium on Security and Privacy*, vol. 18, no. 5, pp. 47–56, May 2020.

[6] J. Singh, J. Cobbe, D. L. Quoc, and Z. Tarkhani, "Enclaves in the clouds: Legal considerations and broader implications," *Communications of the ACM*, vol. 64, no. 5, pp. 42–51, Apr. 2021.

[7] D. G. Mahmoud, V. Lenders, and M. Stojilović, "Electrical-level attacks on CPUs, FPGAs, and GPUs: Survey and implications in the heterogeneous era," *ACM Computing Surveys*, vol. 55, no. 3, pp. 1–40, Feb. 2022.

[8] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO '99*, Santa Barbara, CA, USA, Aug. 1999, pp. 387–97.

[9] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems—CHES '04*, Cambridge, MA, USA, Aug. 2004, pp. 16–29.

[10] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. New York, NY: Springer, 2007.

[11] K. M. Zick, M. Srivastav, W. Zhang, and M. French, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *Proceedings of the 21th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2013, pp. 101–4.

[12] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Dresden, Germany, Mar. 2018, pp. 1111–16.

[13] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *39th IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, May 2018, pp. 805–20.

[14] O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Are cloud FPGAs really vulnerable to power analysis attacks?" in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Grenoble, France, Mar. 2020, pp. 1007–10.

[15] S. Tian, S. Moini, A. Wolnikowski, D. Holcomb, R. Tessier, and J. Szefer, "Remote power attacks on the versatile tensor accelerator in multi-tenant FPGAs," in *Proceedings of the 29th IEEE Symposium on Field-Programmable Custom Computing Machines*, Orlando, FL, USA, May 2021, pp. 242–6.

[16] S. Moini, S. Tian, J. Szefer, D. Holcomb, and R. Tessier, "Remote power side-channel attacks on CNN accelerators in FPGAs," arXiv, Nov. 2020, arXiv: 2011.07603.

[17] I. Levi, D. Bellizia, and F.-X. Standaert, "Reducing a masked implementation's effective security order with setup manipulations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, pp. 293–317, Feb. 2019.

[18] O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Built-in self-evaluation of first-order power side-channel leakage for FPGAs," in *Proceedings of the 28th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Seaside, CA, USA, Feb. 2020, pp. 204–10.

[19] S. Byma, J. G. Steffan, H. Bannazadeh, A. Leon-Garcia, and P. Chow, "FPGAs in the cloud: Booting virtualized hardware accelerators with openstack," in *Proceedings of the 22nd IEEE Symposium on Field-Programmable Custom Computing Machines*, Boston, MA, USA, May 2014, pp. 109–16.

[20] M. Asiatici, N. George, K. Vipin, S. A. Fahmy, and P. Ienne, "Virtualized execution runtime for FPGA accelerators in the cloud," *IEEE Access*, vol. 5, no. 1, pp. 1900–10, Feb. 2017.

[21] F. Chen, Y. Shan, Y. Zhang, Y. Wang, H. Franke, X. Chang, and K. Wang, "Enabling FPGAs in the Cloud," in *Proceedings of the 11th ACM Conference on Computing Frontiers*, New York, NY, USA, May 2014, pp. 1–10.

[22] A. Vaishnav, K. D. Pham, and D. Koch, "A survey on FPGA virtualization," in *Proceedings of the 28th International Conference on Field-Programmable Logic and Applications*, Dublin, Ireland, Aug. 2018, pp. 131–8.

[23] Z. István, G. Alonso, and A. Singla, "Providing Multi-tenant Services with FPGAs: Case Study on a Key-Value Store," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, Dublin, Ireland, Aug. 2018, pp. 119–24.

[24] Y. Zha and J. Li, "Virtualizing FPGAs in the cloud," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, Mar. 2020, pp. 845–58.

[25] A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schkufza, and C. J. Rossbach, "Sharing, Protection, and Compatibility for Reconfigurable Fabric with AmorphOS," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA, Oct. 2018, pp. 107–127.

[26] M. Paolino, S. Pinneterre, and D. Raho, "FPGA virtualization with accelerators overcommitment for network function virtualization," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, Dec. 2017, pp. 1–6.

[27] S. Pinneterre, S. Chiotakis, M. Paolino, and D. Raho, "vFPGAmanager: A virtualization framework for orchestrated FPGA accelerator sharing in 5G cloud environments," in *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, Valencia, Spain, Jun. 2018, pp. 1–5.

[28] J. Weerasinghe, F. Abel, C. Hagleitner, and A. Herkersdorf, "Enabling FPGAs in hyperscale data centers," in *IEEE 12th Intl. Conf. on Ubiquitous Intelligence and Computing and 12th Intl. Conf. on Autonomic and Trusted Computing and 15th Intl. Conf. on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, Beijing, China, Aug. 2015, pp. 1078–86.

[29] J. M. Mbongue, D. T. Kwadjo, A. Shuping, and C. Bobda, "Deploying multi-tenant FPGAs within linux-based cloud infrastructure," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 2, pp. 1–31, Jun. 2022.

[30] H. Yu, A. M. Peters, A. Akshintala, and C. J. Rossback, "Automatic virtualization of accelerators," in *Workshop on Hot Topics in Operating Systems (HotOS)*, Bertinoro, Italy, May 2019, pp. 58–65.

[31] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M. Herbordt, H. Shahzad, P. Hofste, B. Ringlein, J. Szefer, A. Sanaullah, and R. Tessier, "The future of FPGA acceleration in datacenters and the cloud," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 3, pp. 1–41, Feb. 2022.

[32] S. S. Mirzargar and M. Stojilović, "Physical side-channel attacks and covert communication on FPGAs: A survey," in *Proceedings of the 29th International Conference on Field-Programmable Logic and Applications*, Barcelona, Spain, Sep. 2019, pp. 202–10.

[33] O. Glamočanin, D. G. Mahmoud, F. Regazzoni, and M. Stojilović, "Shared FPGAs and the Holy Grail: Protections against side-channel and fault attacks," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Grenoble, France, Feb. 2021, pp. 1645–50.

[34] D. Spielmann, O. Glamocanin, and M. Stojilovic, "RDS: FPGA routing delay sensors for effective remote power analysis attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 2, pp. 543–67, Mar. 2023.

[35] O. Glamočanin, H. Bazaz, M. Payer, and M. Stojilović, "Temperature impact on remote power side-channel attacks on shared FPGAs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Grenoble, France, Mar. 2020, pp. 1007–10.

[36] O. Glamočanin, A. Kostić, S. Kostić, and M. Stojilović, "Active wire fences for multitenant FPGAs," in *Proceedings of the 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems*, Tallinn, Estonia, May 2023, pp. 1–8.

[37] *Tencent Cloud*, Tencent FPGA, 2022, https://intl.cloud.tencent.com/document/product/213/11518#OTHER.

[38] *Baidu Cloud*, Baidu FPGA, 2022, https://cloud.baidu.com/product/fpga.html.

[39] *Project Catapult*, Microsoft Research, 2019, https://www.microsoft.com/en-us/research/project/project-catapult/.

[40] M. Zhao, M. Gao, and C. Kozyrakis, "ShEF: Shielded enclaves for cloud FPGAs," in *27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Lausanne, Switzerland, Feb. 2022, pp. 1070–85.

[41] F. Turan and I. Verbauwhede, "Trust in FPGA-accelerated cloud computing," *ACM Computing Surveys*, vol. 53, no. 6, pp. 1–28, Dec. 2020.

[42] I. Giechaskiel, S. Tian, and J. Szefer, "Cross-VM covert- and side-channel attacks in cloud FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 16, no. 1, pp. 1–29, Dec. 2022.

[43] 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 *Proceedings of the 26th IEEE Symposium on Field-Programmable Custom Computing Machines*, Boulder, CO, USA, May 2018, pp. 1–8.

[44] I. Giechaskiel, K. B. Rasmussen, and K. Eguro, "Leaky wires: Information leakage and covert communication between FPGA long wires," in *Proceedings of 13th ACM ASIA Conference on Information, Computer and Communications Security (ASIACCS)*, Songdo, Incheon, Republic of Korea, Jun. 2018, pp. 15–27.

[45] I. Giechaskiel and J. Szefer, "Information leakage from FPGA routing and logic elements," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Virtual Event, USA, Nov. 2020, pp. 1–9.

[46] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C3APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage," in *2020 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, May 2020, pp. 1728–41.

[47] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "Remote inter-chip power analysis side-channel attacks at board-level," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, New York, NY, USA, Nov. 2018, pp. 114:1–114:7.

[48] J. Gravellier, J.-M. Dutertre, Y. Teglia, P. Loubet-Moundi, and F. Olivier, "Remote side-channel attacks on heterogeneous SoC," in *18th Smart Card Research and Advanced Applications Conference (CARDIS 2019)*, Prague, Czech Republic, Nov. 2019, pp. 109–25.

[49] D. R. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *Proceedings of the 27th International Conference on Field-Programmable Logic and Applications*, Ghent, Belgium, Sep. 2017, pp. 1–7.

[50] D. G. Mahmoud, W. Hu, and M. Stojilović, "X-Attack: Remote activation of satisfiability don't-care hardware Trojans on shared FPGAs," in *Proceedings of the 30th International Conference on Field-Programmable Logic and Applications*, Gothenburg, Sweden, Aug. 2020, pp. 185–92.

[51] D. Mahmoud and M. Stojilović, "Timing violation induced faults in multi-tenant FPGAs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Florence, Italy, Mar. 2019, pp. 1745–50.

[52] 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*, vol. 2018, no. 3, pp. 44–68, Aug. 2018.

[53] G. Provelengios, D. Holcomb, and R. Tessier, "Power wasting circuits for cloud FPGA attacks," in *Proceedings of the 30th International Conference on Field-Programmable Logic and Applications*, Gothenburg, Sweden, Aug. 2020, pp. 231–235.

[54] 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)*, Atlanta, GA, USA, Aug. 2019, pp. 48–55.

[55] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. USA: Kluwer Academic Publishers, 1999.

[56] "Ultrascale architecture configurable logic block user guide (UG574)," https://docs.xilinx.com/v/u/en-US/ug574-ultrascale-clb, Xilinx.

[57] "Versal AI core series," https://www.xilinx.com/products/silicon-devices/acap/versal-ai-core.html, Xilinx.

[58] A. L. Masle, G. C. T. Chow, and W. Luk, "Constant power reconfigurable computing," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, New Delhi, India, Jan. 2011, pp. 1–8.

[59] 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 *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Westminster, CO, USA, Nov. 2019, pp. 1–8.

[60] N. L. Masle and W. Luk, "Detecting power attacks on reconfigurable hardware," in *Proceedings of the 22nd International Conference on Field-Programmable Logic and Applications*, Oslo, Norway, Aug. 2012, pp. 14–19.

[61] *AWS EC2 FPGA GitHub*, Amazon, 2019, https://github.com/aws/aws-fpga/blob/master/ERRATA.md.

[62] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "Measuring long wire leakage with ring oscillators in cloud FPGAs," in *Proceedings of the 29th International Conference on Field-Programmable Logic and Applications*, Barcelona, Spain, Sep. 2019, pp. 45–50.

[63] G. Provelengios, D. Holcomb, and R. Tessier, "Characterizing power distribution attacks in multi-user FPGA environments," in *Proceedings of the 29th International Conference on Field-Programmable Logic and Applications*, Barcelona, Spain, Sep. 2019, pp. 194–201.

[64] S. S. Mirzargar, G. Renault, A. Guerrieri, and M. Stojilović, "Nonintrusive and adaptive monitoring for locating voltage attacks in virtualized FPGAs," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, Maui, HI, USA, Dec. 2020, pp. 1–2.

[65] D. R. E. Gnad, C. D. K. Nguyen, S. H. Gillani, and M. B. Tahoori, "Voltage-based covert channels using FPGAs," *ACM Transactions on Design Automation of Electronic Systems*, vol. 26, no. 6, pp. 1–25, Jun. 2021.

[66] A. Moradi and O. Mischke, "Glitch-free implementation of masking in modern FPGAs," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, San Francisco, CA, USA, Jun. 2012, pp. 89–95.

[67] A. Moradi, O. Mischke, and T. Eisenbarth, "Correlation-enhanced power analysis collision attack," in *Cryptographic Hardware and Embedded Systems - CHES 2010*, Berling, Germany, Aug. 2010, pp. 125–39.

[68] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked AES hardware implementations," in *Cryptographic Hardware and Embedded Systems—CHES '05*, Berlin, Germany, Aug. 2005, pp. 157–71.

[69] M. Azouaoui, R. Poussier, F. Standaert, and V. Verneuil, "Key enumeration from the adversarial viewpoint," in *18th Smart Card Research and Advanced Applications Conference (CARDIS 2019)*, Prague, Czech Republic, Nov. 2019, pp. 252–67.

[70] K. Papagiannopoulos, O. Glamočanin, M. Azouaoui, D. Ros, F. Regazzoni, and M. Stojilović, "The side-channel metrics cheat sheet," *ACM Computing Surveys*, vol. 55, no. 10, pp. 1–38, Feb. 2023.

[71] G. Cezary, G. Vincent, P. Romain, S. Joachim, and S. François-Xavier, "Simpler and more efficient rank estimation for side-channel security assessment," in *International Workshop on Fast Software Encryption*, Istanbul, Turkey, Mar. 2015, pp. 117–29.

[72] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side-channel resistance validation," in *NIST Non-Invasive Attack Testing Workshop*, Nara, Japan, Sep. 2011, pp. 1–15.

[73] T. Schneider and A. Moradi, "Leakage assessment methodology," *Journal of Cryptographic Engineering*, vol. 6, no. 2, pp. 85–99, Jun. 2016.

[74] I. Levi, D. Bellizia, and F.-X. Standaert, "Reducing a masked implementation's effective security order with setup manipulations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, pp. 293–317, Feb. 2019.

[75] B. Yang, V. Rožić, N. Mentens, W. Dehaene, and I. Verbauwhede, "Total: Trng on-the-fly testing for attack detection using lightweight hardware," in *Proceedings of the Design,*

Automation and Test in Europe Conference and Exhibition, Dresden, Germany, Mar. 2016, pp. 127–132.

[76] J. Howe, A. Khalid, M. Martinoli, F. Regazzoni, and E. Oswald, "Fault attack countermeasures for error samplers in lattice-based cryptography," Cryptology ePrint Archive, Report 2019/206, 2019, https://eprint.iacr.org/2019/206.

[77] N. Gattu, M. N. I. Khan, A. De, and S. Ghosh, "Power side channel attack analysis and detection," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Virtual Event, USA, Nov. 2020, pp. 1–7.

[78] T. Mosavirik, P. Schaumont, and S. Tajik, "Impedanceverif: On-chip impedance sensing for system-level tampering detection," Cryptology ePrint Archive, Paper 2022/946, 2022, https://eprint.iacr.org/2022/946. [Online]. Available: https://eprint.iacr.org/2022/946

[79] N. Khan, S. Nitzsche, R. Frank, L. Bauer, J. Henkel, and J. Becker, "Amplifying side channel leakage by hardware modification of Xilinx Zynq-7 FPGA evaluation boards," in *The Thirteenth International Conference on Emerging Security Information, Systems and Technologies*, Nice, France, Oct. 2019, pp. 1–7.

[80] L. Mazur and M. Novotný, "Differential power analysis on FPGA board: Boundaries of success," in *6th Mediterranean Conference on Embedded Computing (MECO)*, Bar, Montenegro, Jun. 2017, pp. 1–4.

[81] S. Sonar, D. B. Roy, R. S. Chakraborty, and D. Mukhopadhyay, "Side-channel watchdog: Run-time evaluation of side-channel vulnerability in FPGA-based crypto-systems," Cryptology ePrint Archive, Report 2016/182, 2016, https://eprint.iacr.org/2016/182.

[82] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, Aug. 1962.

[83] T. F. Chan, G. H. Golub, and R. J. Leveque, "Algorithms for computing the sample variance: Analysis and recommendations," *The American Statistician*, vol. 37, no. 3, pp. 242–247, Aug. 1983.

[84] R. C. for Information Security, "Sasebo-GII quick start guide," http://satoh.cs.uec.ac.jp/SASEBO/en/board/sasebo-g2.html, National Institute of Advanced Industrial Science and Technology, 2009, accessed: 2019-20-01.

[85] *Virtex-5 Family Overview*, Xilinx, 2015. [Online]. Available: https://www.xilinx.com

[86] C. Drewes, O. Weng, K. Ryan, B. Hunter, C. McCarty, R. Kastner, and D. Richmond, "Turn on, tune in, listen up: Maximizing side-channel recovery in time-to-digital converters," in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2023, pp. 111–22.

[87] *AES Encryption Core*, http://www.aoki.ecei.tohoku.ac.jp/crypto/, AIST and Tohoku University, 2019.

[88] F. Regazzoni, W. Yi, and F.-X. Standaert, "FPGA implementations of the AES masked against power analysis attacks," in *Proceedings of 2nd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, Darmstadt, Germany, Feb. 2011, pp. 1–11.

[89] S. Trimberger and S. McNeil, "Security of FPGAs in data centers," in *2nd International Verification and Security Workshop (IVSW)*, Thessaloniki, Greece, jul 2017, pp. 117–22.

[90] J. Wu, "Several key issues on implementing delay line based TDCs using FPGAs," *IEEE Transactions on Nuclear Science*, vol. 57, no. 3, pp. 1543–8, Jun. 2010.

[91] D. R. Gnad, F. Oboril, S. Kiamehr, and M. B. Tahoori, "Analysis of transient voltage fluctuations in FPGAs," in *Proceedings of the IEEE International Conference on Field Programmable Technology*, Xi'an, China, Dec. 2016, pp. 12–19.

[92] Y.-k. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "A quantitative analysis on microarchitectures of modern CPU-FPGA platforms," in *Proceedings of the 53rd Annual Design Automation Conference*, New York, NY, USA, 2016, pp. 1–6.

[93] "MicroBlaze Processor," https://www.xilinx.com/content/dam/xilinx/support/documentation/white_papers/wp501-microblaze.pdfs, Xilinx, 2021.

[94] "Nios®v processor," https://www.intel.com/content/www/us/en/products/details/fpga/nios-processor/v.html, Intel®FPGA, 2022.

[95] *PicoRV CPU*, https://github.com/YosysHQ/picorv32, Yosys HQ, 2022.

[96] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. A. Faruque, "Stealing neural network structure through remote FPGA side-channel analysis," in *Proceedings of the 29th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Virtual, Feb. 2021, pp. 4377–88.

[97] A. S. Rakin, X. X. Yukui Luo, and D. Fan, "Deep-Dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenant FPGA," in *Usenix Security Symposium*, Virtual, Aug. 2021, pp. 1919–36.

[98] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Power side-channel attacks on BNN accelerators in remote FPGAs," arXiv, Apr. 2021, arXiv: 2011.07603.

[99] J.-J. Quisquater and D. Samyde, "Automatic code recognition for smart cards using a Kohonen neural network," in *Smart Card Research and Advanced Applications Conference, CARDIS 2002*, San Jose, CA, USA, Nov 2002, pp. 1–9.

[100] J. Park and A. Tyagi, "Using power clues to hack IoT devices: The power side channel provides for instruction-level disassembly." *IEEE Consumer Electronics Magazine*, vol. 6, no. 3, pp. 92–102, 2017.

[101] *RV32I Instruction Set Architecture*, https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf, RISC-V Foundation, 2022.

[102] K. Subramanian, "RISCY processor," https://github.com/mongrelgem/RISCY, 2021, accessed: 2021-9-20.

[103] S. Lab, "Sakura X side-channel evaluation board," https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html, 2021, accessed: 2021-9-20.

[104] Noname, "Instruction-Level Power Side-Channel Leakage Evaluation of Soft-Core CPUs on Shared FPGAs: Artifacts," Available: in-preparation.com, 2023.

[105] M. Gobulukoglu, C. Drewes, W. Hunter, R. Kastner, and D. Richmond, "Classifying computations on multi-tenant FPGAs," in *Proceedings of the 58th Design Automation Conference*, San Francisco, California, USA, Feb. 2021, pp. 1261–66.

[106] S. Tian, W. Xiong, I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "Fingerprinting cloud FPGA infrastructures," in *Proceedings of the 28th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Seaside, CA, USA, Feb. 2020, pp. 58–64.

[107] A. Moradi and T. Schneider, "Side-channel analysis protection and low-latency in action - case study of PRINCE and Midori," in *Advances in Cryptology - ASIACRYPT*, Hanoi, Vietnam, Dec. 2016, pp. 517–47.

[108] A. Moradi, "Side-channel leakage through static power - should we care about in practice?" in *International Workshop on Cryptographic Hardware and Embedded Systems*, Busan, South Korea, Sep. 2014, pp. 562–79.

[109] D. W. Todd, "Tightly coupling the PicoRV32 RISC-V processor with custom logic accelerators via a generic interface," Master Thesis, 2021, accessed: 2023-05-29. [Online]. Available: https://tigerprints.clemson.edu/cgi/viewcontent.cgi?article=4559&context=all_theses

[110] "The rocket chip generator," https://github.com/chipsalliance/rocket-chip, EECS Department, University of California, Berkeley, Apr. 2016.

[111] J. Krautter, D. R. E. Gnad, and M. B. 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. 121–46, Jun. 2020.

[112] V. Cristiani, M. Lecomte, and T. Hiscock, "A bit-level approach to side channel based disassembling," in *18th Smart Card Research and Advanced Applications Conference (CARDIS 2019)*, Prague, Czech Republic, Nov. 2019, pp. 143–58.

[113] D. Vermoen, M. Witteman, and G. N. Gaydadjiev, "Reverse engineering Java card applets using power analysis," in *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, Berlin, Heidelberg, May 2007, pp. 138–49.

[114] D. Krishnankutty, Z. Li, R. Robucci, N. Banerjee, and C. Patel, "Instruction sequence identification and disassembly using power supply side-channel analysis," *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1639–53, Nov. 2020.

[115] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "SCANDALee: A side-channel-based disassembler using local electromagnetic emanations," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Grenoble, France, Mar. 2015, pp. 139–44.

[116] J. Park, X. Xu, Y. Jin, D. Forte, and M. Tehranipoor, "Power-based side-channel instruction-level disassembler," in *Proceedings of the 55th Design Automation Conference*, San Francisco, CA, USA, Jun. 2018, pp. 1–6.

[117] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a side channel based disassembler," in *Transactions on Computational Science X: Special Issue on Security in Computing, Part I*, vol. 6340, Berlin, Heidelberg, 2010, pp. 78–99.

[118] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–80, Nov. 1997.

[119] P. Bashivan, I. Rish, M. Yeasin, and N. Codella, "Learning representations from EEG with deep recurrent-convolutional neural networks," arXiv, 2015, https://arxiv.org/abs/1511.06448.

[120] B. Wang, T. Jiang, X. Zhou, B. Ma, F. Zhao, and Y. Wang, "Time-series classification based on fusion features of sequence and visualization," *Applied Sciences*, vol. 10, no. 12, pp. 13 911–32, Jun. 2020.

[121] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–63, 2019.

[122] D. R. Jiménez, "Time series classification using deep learning hybrid architectures," https://github.com/danyrubiano/deepTSC, 2021, accessed: 2021-9-20.

[123] R. Gwinn, M. A. Matties, and A. D. Rubin, "Wavelet selection and employment for side-channel disassembly," arXiv, 2021, arXiv: 2107.11870.

[124] N. Debande, Y. Souissi, M. A. E. Aabid, S. Guilley, and J.-L. Danger, "Wavelet transform based pre-processing for side channel analysis," in *45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops*, Vancouver, BC, Canada, Mar. 2012, pp. 32–38.

[125] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Paris, France, Feb. 2004, pp. 1–6.

[126] E. D. Mulder, S. Gummalla, and M. Hutter, "Protecting RISC-V against side-channel attacks," in *Proceedings of the 56th Design Automation Conference*, Las Vegas, NV, USA, Jun 2019, pp. 1–4.

[127] M. A. K. F, V. Ganesan, R. Bodduna, and C. Rebeiro, "PARAM: A microprocessor hardened for power side-channel attack resistance," in *2020 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, San Jose, CA, USA, Dec 2020, pp. 23–34.

[128] L. Yan, Y. Guo, X. Chen, and H. Mei, "A study on power side channels on mobile devices," in *Proceedings of the 7th Asia-Pacific Symposium on Internetware*, New York, NY, USA, Nov 2015, pp. 30–38.

[129] A. G. Bayrak, N. Velicković, P. Ienne, and W. Burleson, "An architecture-independent instruction shuffler to protect against side-channel attacks," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, pp. 20:1–20:19, Jan. 2012.

[130] T. Güneysu and A. Moradi, "Generic side-channel countermeasures for reconfigurable devices," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 6917, no. 1, pp. 33–48, Sep. 2011.

[131] P. Sasdrich, A. Moradi, O. Mischke, and T. Güneysu, "Achieving side-channel protection with dynamic logic reconfiguration on modern FPGAs," in *2015 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Washington, DC, USA, May 2015, pp. 130–36.

[132] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "Mitigating electrical-level attacks towards secure multi-tenant FPGAs in the cloud," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, no. 3, pp. 1–26, Aug. 2019.

[133] T. M. La, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch, "FPGADefender: Malicious self-oscillator scanning for Xilinx UltraScale + FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 13, no. 3, pp. 1–31, Sep. 2020.

[134] D. R. E. Gnad, V. Meyers, N. M. Dang, F. Schellenberg, A. Moradi, and M. B. Tahoori, "Stealthy logic misuse for power analysis attacks in multi-tenant FPGAs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Virtual Conference & Exhibition, Feb. 2021, pp. 1012–15.

[135] 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 *Proceedings of the IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, May 2007, pp. 1–15.

[136] 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)*, Fayetteville, AR, USA, May 2020, pp. 65–69.

[137] "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators," https://docs.xilinx.com/v/u/en-US/xapp052, Xilinx, accessed: 2023-3-16.

[138] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Remote power side-channel attacks on BNN accelerators in FPGAs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, Feb. 2021, pp. 1–6.

[139] B. Udugama, D. Jayasinghe, H. Saadat, A. Ignjatovic, and S. Parameswaran, "VITI: A tiny self-calibrating sensor for power-variation measurement in FPGAs," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 1, pp. 657–78, Nov. 2021.

[140] D. Spielmann, O. Glamočanin, and M. Stojilović, "RDS: FPGA routing delay sensors for effective remote power analysis attacks. Artifacts (v1.0)," https://doi.org/10.5281/zenodo.7534002, Jan. 2023, accessed: 2023-1-13.

[141] I. Ahmed, L. L. Shen, and V. Betz, "Optimizing FPGA logic circuitry for variable voltage supplies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 890–903, Apr. 2020.

[142] A. Dasdan and I. Hom, "Handling inverted temperature dependence in static timing analysis," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 2, pp. 306–324, Apr. 2006.

[143] A. Heuser, M. Kasper, W. Schindler, and S. Marc, "A new difference method for side-channel analysis with high-dimensional leakage models," in *Topics in Cryptology—CT-RSA 2012*, Berlin, Heidelberg, Jan. 2012, pp. 365–82.

[144] V. Meyers, D. R. Gnad, and M. Tahoori, "Reverse engineering neural network folding with remote FPGA power analysis," in *Proceedings of the 30th IEEE Symposium on Field-Programmable Custom Computing Machines*, New York City, NY, USA, May 2022, pp. 1–10.

[145] T. Sakurai and A. R. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 25, no. 2, pp. 584–94, Apr. 1990.

[146] T. Tsai, H.-C. Lin, and P.-W. Li, "Temperature-dependent narrow width effects of 28-nm CMOS transistors for cold electronics," *IEEE Journal of the Electron Devices Society (J-EDS)*, vol. 10, pp. 289–96, Mar. 2022.

[147] S. Tedja, J. Van der Spiegel, and H. Williams, "Analytical and experimental studies of thermal noise in MOSFET's," *IEEE Transactions on Electron Devices*, vol. 41, no. 11, pp. 2069–75, Nov. 1994.

[148] D. Triantis, A. Birbas, and D. Kondis, "Thermal noise modeling for short-channel MOSFETs," *IEEE Transactions on Electron Devices*, vol. 43, no. 11, pp. 1950–55, Nov. 1996.

[149] J. Schurr, H. Moser, K. Pierz, G. Ramm, and B. P. Kibble, "Johnson-Nyquist noise of the quantized hall resistance," *IEEE Transactions on Instrumentation and Measurement (IEEE I&M)*, vol. 60, no. 7, pp. 2280–85, 2011.

[150] 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 (TRETS)*, vol. 5, no. 1, pp. 1–26, Mar. 2012.

[151] E. S. LAB, "SCABox," https://github.com/emse-sas-lab/SCAbox, Mines Saint-Etienne.

[152] D. M. Mehta, M. Hashemi, D. S. Koblah, D. Forte, and F. Ganji, "Bake it till you make it: Heat-induced leakage from masked neural networks," Cryptology ePrint Archive, Paper 2023/076, 2023, https://eprint.iacr.org/2023/076. [Online]. Available: https://eprint.iacr.org/2023/076

[153] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, "Error analysis and detection procedures for a hardware implementation of the advanced encryption standard," *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 492–505, Apr. 2003.

[154] 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, pp. 441–64, Jul. 2021.

[155] J. Gravellier, J.-M. Dutertre, Y. Teglia, and P. Loubet-Moundi, "High-speed ring oscillator based sensors for remote side-channel attacks on FPGAs," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, Dec. 2019, pp. 1–8.

[156] G. Provelengios, C. Ramesh, S. B. Patil, K. Eguro, R. Tessier, and D. Holcomb, "Characterization of long wire data leakage in deep submicron FPGAs," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, Monterey, CA, USA, Feb. 2019, pp. 292–97.

[157] S. Moini, X. Li, P. Stanwicks, G. Provelengios, W. Burleson, R. Tessier, and D. Holcomb, "Understanding and comparing the capabilities of on-chip voltage sensors against remote power attacks on FPGAs," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, Springfield, MA, USA, Aug. 2020, pp. 941–44.

[158] 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 & Test*, vol. 37, no. 2, pp. 111–19, Jan. 2020.

[159] S. Tian and J. Szefer, "Temporal thermal covert channels in cloud FPGAs," in *Proceedings of the 27th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, New York, NY, USA, Feb. 2019, pp. 298–303.

Ognjen **Glamočanin**

Computer Science PhD Student

(+41) 078-948-15-35 | ognjen.glamocanin@epfl.ch | OgacNS94 | ognjen-glamocanin

Education

EPFL, Ecole Polytechnique Fédérale de Lausanne — *Lausanne, Switzerland*

PhD in Computer Science — ***Graduated: June 14, 2023***

- Thesis topic: Power side-channel security of remote FPGAs
- Thesis advisors: Dr. Mirjana Stojilović and Prof. Babak Falsafi

Sorbonne Université, Paris VI — *Paris, France*

M.S. in Computer Science — *2017 – 2018*

University of Novi Sad, Faculty of Technical Sciences — *Novi Sad, Serbia*

B.S. with Honours in Electrical Engineering — *2013 – 2017*

Work Experience

ARM — *Sophia Antipolis, France*

CPU Microarchitecture and Design Intern — *Mar 2018 – Aug 2018*

- Analyzed CPU microarchitectural events for the purposes of **power consumption estimation** during cycle-accurate simulation
- Used **Python sklearn** to model correlation between CPU events and power consumption simulated in **Cadence Joules**
- Enabled power estimation in early microarchitecture design stages by integrating power prediction in a **C/C++** cycle-accurate simulator

FROBAS D.O.O. — *Novi Sad, Serbia*

Machine Learning Hardware Acceleration Intern — *Nov 2016 – Jun 2017*

- Used **VHDL** to design and verify a hardware accelerator for multi-layer perceptron (MLP) artificial neural networks (ANNs)

ELSYS EASTERN EUROPE — *Belgrade, Serbia*

Hardware Functional Verification Intern — *Jul 2016 – Oct 2016*

- Used **SystemVerilog** and the **UVM** methodology to build a functional verification environment for an OCP2UART bridge

Technical Skills

Digital design and FPGA development:	RTL design, FPGA design (AMD 7-series, UltraScale+ in Alveo boards), UVM
Programming and scripting languages:	C/C++ (10yrs), Python (6yrs), SystemVerilog, MATLAB, Bash, TCL
Hardware description languages:	VHDL (9yrs), Verilog, SystemVerilog, SystemC
CAD EDA tools:	Xilinx ISE, Xilinx Vivado and Vitis, QuestaSim, Cadence NCSim
ML tools:	Python (Keras, TensorFlow, Weights and Biases, Pandas), Docker, Kubernetes
Cloud frameworks:	AWS EC2, Microsoft Azure, Google Cloud, CoreWeave

Publications

Instruction-Level Power Leakage Evaluation of Soft-Core CPUs on Shared FPGAs — *HaSS*

(under review)

O. Glamočanin, Shashwat Shrivastava, Jinwei Yao, Nour Ardo, Mathias Payer, M. Stojilović — *2023*

- Evaluated the instruction-level power leakage of **RISC-V softcore CPUs** in shared FPGAs using deep learning techniques in **Python Keras**.
- Used **Python WandB**, **Bash**, **Docker**, and **Kubernetes** to streamline and automate the training and exploration of ML model hyperparameters.
- Evaluated of the impact of the **FPGA** family, code template structure, preprocessing, and trace averaging on the model accuracy.

Active Wire Fences for Multi-Tenant FPGAs (Best Paper Award Nomination) — *DDECS*

O. Glamočanin, Anđela Kostić, Staša Kostić, M. Stojilović — *2023*

- Created a novel wire-based **FPGA** power waster architecture using **VHDL** and **XDC**, with no resource overhead compared to the state of the art.
- Deployed a **CUDA**-accelerated power analysis attack on CoreWeave cloud instances with Nvidia A100-80GB GPUs.
- Demonstrated that wire wasters, when used as active fences, outperform the state of the art against remote power analysis attacks.

RDS: FPGA Routing Delay Sensors for Effective Remote Power Analysis Attacks — *TCHES*

David Spielmann*, **O. Glamočanin***, M. Stojilović (* equal contribution) — *2023*

- Designed a novel routing-based FPGA voltage sensor architecture using **VHDL** and **Vivado**, with superior sensing than the state of the art.
- Designed an AXI4-Full **Vitis RTL kernel** for the **Alveo U200 FPGA card**, used for recording and saving encryption power traces.
- Implemented a **C++** interface for the RTL kernel to record millions of power traces and a **Bash** script to automate the trace collection process.

Temperature Impact on Remote Power Side-Channel Attacks on Shared FPGAs — *DATE*

O. Glamočanin, H. Bazaz, M. Payer, M. Stojilović — *2023*

- Analyzed the temperature impact on **FPGA** voltage sensors and remote power analysis attacks.
- Quantified the impact of temperature effects on statistical (CPA on AES encryption) and ML profiling power analysis attacks.

The Side-Channel Metrics Cheat Sheet — *CSUR*

K. Papagiannopoulos*, **O. Glamočanin***, M. Azouaoui*, D. Ros*, F. Regazzoni*, M. Stojilović* (* equal contribution) — *2022*

- Analyzed and compared methods for power side-channel security evaluation, both theoretically and experimentally.
- Contributed to MetriSCA, a **C++** open-source library of metrics for power side-channel analysis accompanying the publication.

Improving First-Order Threshold Implementations of SKINNY — *INDOCRYPT*

A. Caforio, D. Collins, **O. Glamočanin**, and S. Banik — *2021*

- Worked on an efficient threshold implementation protection against power side-channel attacks for the SKINNY cipher, written in **VHDL**.
- Implemented and evaluated the design on **FPGA** using **Xilinx Vivado**, showing no existence of first-order power side-channel leakage.

Shared FPGAs and the Holy Grail: Protections Against Side-Channel and Fault Attacks — *DATE*

O. Glamočanin*, D. G. Mahmoud*, F. Regazzoni, and M. Stojilović (* equal contribution) — *2021*

- Analyzed recently proposed methods for protection against side-channel and fault attacks in shared FPGAs.
- Provided insights on the versatility and inter-operability of the countermeasures, with an emphasis on future research directions.

Are Cloud FPGAs Really Vulnerable to Power-Analysis Attacks? — *DATE*

O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović — *2020*

- Implemented an **FPGA** voltage sensor on state-of-the-art cloud FPGAs (**Xilinx UltraScale+** on **AWS EC2 F1 instances**) using **VHDL** and **Vivado**.
- Demonstrated the first remote power side-channel attack on cloud-scale FPGAs.

Built-In Self-Evaluation of First-Order Power Side-Channel Leakage for FPGAs — *ISFPGA*

O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović — *2020*

- Used **SystemC** and **VHDL** to implement a fixed-point DSP system on **FPGA** to calculate the statistical *t*-test.
- Showed that FPGA-based voltage sensors and the *t*-test can be used for remote power side-channel leakage estimation.
- Designed the first remote power side-channel leakage assessment system, allowing side-channel security reevaluation on deployed devices.

Honors & Awards

2023 **Nomination for the EPFL Doctoral Program Thesis Distinction**, *Switzerland*
Award for the best 8% theses, 30% nomination rate

2018 **EPFL EDIC Fellowship**, *Switzerland*
Fellowship for first-year PhD students

2017 **French Government Scholarship for International Students**, *France*
Full scholarship for master studies in France

2016 **Dr Vladan Desnica Award**, *Serbia*
Best student of the microcomputer electronics track

Teaching Experience

EPFL — *Lausanne, Switzerland*

Teaching Assistant — *Feb 2019 – ongoing*

- **Computer Architecture**: Head TA, managing the course and lab sessions in CPU micro-architecture for 2nd year B.S. students
- **System Programming Project**: Leading lab sessions in C for 2nd year B.S. students
- **Information, Computation, Communication**: Head TA, managing the course and leading lab sessions in Python and C for 1st year B.S. students

University of Novi Sad — *Novi Sad, Serbia*

Teaching Assistant — *Sep 2016 – Jun 2017*

- **Electrical Circuit Theory**: Leading computer lab sessions in MATLAB for 2nd year B.S. students
- **Systems and Signals**: Leading computer lab sessions in MATLAB for 2nd year B.S. students

Languages

Serbian: Mother tongue
English: fluent (level C2)
French: fluent (level C1)
German: beginner (level A1)