Towards Secure Computation on Accelerated Cloud Systems

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Hassan Nassar

aus Gizeh (Ägypten)

Tag der mündlichen Prüfung: 02. Dezember 2024
1. Referent: Prof. Dr.-Ing. Jörg Henkel
2. Referent: Prof. Dr. sc. techn. Andreas Herkersdorf

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

Hassan Nassar

Acknowledgments

I would like to express my gratitude to Prof. Henkel, my supervisor, for giving me the opportunity to work on my dissertation at CES. I greatly appreciate his continued support and advice. I would like to thank him for giving me the freedom to work on interesting topics that interested me and allowing my creativity to grow. I would like to also thank Prof. Herkersdorf for accepting being the co-advisor of my dissertation and for his support.

During my dissertation work, I worked with and learned a lot from Dr.-Ing. Lars Bauer. I thank him for the long discussions we had, whether on technical, philosophical, or just interesting topics. I also would like to thank all my past and present colleagues at CES for enabling a positive work environment, working with all of you is an unforgettable experience. In particular, I want to thank Martin, Jeferson, Bakr, Heba, Kostas, Kilian, Mohamed, Zeynep, Benedikt, George, and Lokesh. In addition, the work presented in this dissertation would not have been possible without collaboration with the groups of Prof. Tahoori and Prof. Chen. I would like to thank both of them and members of their groups, past and present, in particular Dennis, Jonas, Kuan, Christian, and Nils.

To my Egyptian friends in Germany, Sarah, Atef, Ayman (Sakr & Khattab), Hesham, Dina, Ahmed (Mahmoud & Nabil), Omar (Sayad & Shamy), and Mustafa: You made me feel the warmth of home whenever it was needed. To my friends in Egypt, especially mes frères and GUCians: Your support and friendship helped me overcome my toughest times. To my mother Suzan, my grandmother Mervat, my sisters Salma & Jaidaa, and my niece Solana: I love you. Pursuing this dissertation is a decision that I would not have made if not for my father Amr, my grandmother Samia, my grandfathers Adel & Abdelrahman, and my uncle & role model Dr. Emad. You are no longer physically with me, but you are forever in my heart. To those who were part of my journey but left this world too soon, Karim (Sherif & Khouzam & Naguib), Hanna, Marwan, and Yahia: I miss you a lot.

وَكَانَ فَضْلُ اللَّهِ عَلَيْكَ عَظِيمًا

And ever has the grace of God upon you been great!

List of Publications

The following list contains publications published by Hassan Nassar, the author of this dissertation, during the course of his doctoral research. Publications [1–11] make **major contributions** to the dissertation; publications [12–25] make **minor contributions** to the dissertation:

[1] Hassan Nassar, Lars Bauer, and Jörg Henkel. "ANV-PUF: Machine-Learning-Resilient NVM-Based Arbiter PUF". In: *ACM Trans. Embed. Comput. Syst.* 22.5s (2023). ISSN: 1539-9087. DOI: 10.1145/3609388. URL: https://doi.org/10.1145/3609388.

[2] Hassan Nassar, Lars Bauer, and Jörg Henkel. "CaPUF: Cascaded PUF Structure for Machine Learning Resiliency". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.11 (2022), pp. 4349–4360. DOI: 10.1109/TCAD.2022.3197539.

[3] Hassan Nassar, Lars Bauer, and Jörg Henkel. "Effects of Runtime Reconfiguration on PUFs Implemented as FPGA-Based Accelerators". In: *IEEE Embedded Systems Letters* 15.4 (2023), pp. 174–177. DOI: 10.1109/LES.2023.3299214.

[4] Hassan Nassar, Philipp Machauer, Lars Bauer, Dennis Gnad, Mehdi Tahoori, and Jörg Henkel. "DoS-FPGA: Denial of Service on Cloud FPGAs via Coordinated Power Hammering". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. accepted. 2024.

[5] Hassan Nassar, Philipp Machauer, Dennis R. E. Gnad, Lars Bauer, Mehdi B. Tahoori, and Jörg Henkel. "Covert-Hammer: Coordinating Power-Hammering on Multi-tenant FPGAs via Covert Channels". In: *ACM/SIGDA ISFPGA*. Poster. 2024. DOI: 10.1145/3626202.3637613.

[6] Hassan Nassar, Simon Pankner, Lars Bauer, and Jörg Henkel. "Late Breaking Results: Configurable Ring Oscillators as a Side-Channel Countermeasure". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–2. DOI: 10.1109/DAC56929.2023.10247786.

[7] Hassan Nassar, Jeferson Gonzalez-Gomez, Varun Manjunath, Lars Bauer, and Jörg Henkel. "Through Fabric: A Cross-world Thermal Covert Channel on TEE-enhanced FPGA-MPSoC Systems". In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. accepted. 2025. DOI: 10.1145/3658617.3697767.

[8] Hassan Nassar, Lars Bauer, and Jörg Henkel. "HBMorphic: FHE Acceleration via HBM-Enabled Recursive Karatsuba Multiplier on FPGA". In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2024. DOI: 10.1109/FCCM60383.2024.00038.

[9] Hassan Nassar, Lars Bauer, and Jörg Henkel. "Turbo-FHE: Accelerating Fully Homomorphic Encryption with FPGA and HBM Integration". In: *IEEE Design and Test Magazine* (2025). accepted. DOI: 10.1109/MDAT.2025.3527368.

[10] Hassan Nassar, Jonas Krautter, Lars Bauer, Dennis Gnadd, Mehdi Tahoori, and Jörg Henkel. "Meta-Scanner: Detecting Fault Attacks via Scanning FPGA Designs Metadata". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024). DOI: 10.1109/TCAD.2024.3443769.

[11] Hassan Nassar, Hanna AlZughbi, Dennis Gnad, Lars Bauer, Mehdi Tahoori, and Jörg Henkel. "LoopBreaker: Disabling Interconnects to Mitigate Voltage-Based Attacks in Multi-Tenant FPGAs". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2021. DOI: 10.1109/ICCAD51958.2021.9643485.

[12] Hassan Nassar, Lars Bauer, and Jörg Henkel. "TiVaPRoMi: Time-Varying Probabilistic Row-Hammer Mitigation". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Virtual Event: IEEE, 2021, pp. 1711–1716. DOI: 10.23919/DATE51398.2021.9473912.

[13] Hassan Nassar, Rafik Youssef, Lars Bauer, and Jörg Henkel. "Supporting Dynamic Control-Flow Execution for Runtime Reconfigurable Processors". In: *2023 International Conference on Microelectronics (ICM)*. 2023, pp. 184–189. DOI: 10.1109/ICM60448.2023.10378905.

[14] Nils Hölscher, Christian Hakert, Hassan Nassar, Kuan-Hsun Chen, Lars Bauer, Jian-Jia Chen, and Jörg Henkel. "Memory Carousel: LLVM-Based Bitwise Wear-Leveling for Non-Volatile Main Memory". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.8 (2022), pp. 1–13. DOI: 10.1109/TCAD.2022.3228897.

[15] Lokesh Siddhu, Hassan Nassar, Lars Bauer, Christian Hakert, Nils Hölscher, Jian-Jia Chen, and Joerg Henkel. "Swift-CNN: Leveraging PCM Memory's Fast Write Mode to Accelerate CNNs". In: *IEEE Embedded Systems Letters* 15.4 (2023), pp. 234–237. DOI: 10.1109/LES.2023.3298742.

[16] Lilas Alrahis, Hassan Nassar, Jonas Krautter, Dennis Gnad, Lars Bauer, Jörg Henkel, and Mehdi Tahoori. "MaliGNNoma: GNN-Based Malicious Circuit Classifier for Secure Cloud FPGAs". In: *IEEE HOST*. 2024. DOI: 10.1109/HOST55342.2024.10545411.

[17] Jeferson Gonzalez-Gomez, Hassan Nassar, Lars Bauer, and Jörg Henkel. "LightFAt: Mitigating Control-Flow Explosion via Lightweight PMU-Based Control-Flow Attestation". In: *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2024, pp. 222–226. DOI: 10.1109/HOST55342.2024.10545348.

[18] Lars Bauer, Jürgen Becker, Jörg Henkel, Fabian Lesniak, and Hassan Nassar. "Adaptive Application-Specific Invasive Micro-Architectures". In: *Invasive Computing*. FAU University Pres, 2022.

[19] Jörg Henkel, Lokesh Siddhu, Hassan Nassar, Lars Bauer, Jian-Jia Chen, Christian Hakert, Tristan Seidl, Kuan-Hsun Chen, Xiaobo Sharon Hu, Mengyuan Li, Chia-Lin Yang, and Ming-Liang Wei. "(Invited Paper) Co-Designing NVM-based Systems for Machine Learning and Inmemory Search Applications". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. accepted. 2024.

[20] Lars Bauer, Hassan Nassar, Nadir Khan, Jürgen Becker, and Jörg Henkel. "Machine-Learning-based Side-Channel Attack Detection for FPGA SoCs". In: *IEEE Transactions on Circuits and Systems for Artificial Intelligence* (2024). DOI: 10.1109/TCASAI.2024.3483118.

[21] Simon Bothe, Hassan Nassar, Lars Bauer, and Jörg Henkel. "Client-Server Framework for FPGA Acceleration of Fan-Vercauteren-Based Homomorphic Encryption". In: *2024 International Conference on Microelectronics (ICM)*. 2024, pp. 1–5. DOI: 10.1109/ICM63406.2024.10815906.

[22] Mina Ibrahim, Martel Shokry, Lokesh Siddhu, Lars Bauer, Hassan Nassar, and Jörg Henkel. "An FPGA-Based RISC-V Instruction Set Extension and Memory Controller for Multi-Level Cell NVM". In: *2024 International Conference on Microelectronics (ICM)*. 2024, pp. 1–6. DOI: 10.1109/ICM63406.2024.10815826.

[23] Jayeeta Chaudhuri, Hassan Nassar, Dennis Gnad, Jörg Henkel, Mehdi Tahoori, and Krishnendu Chakrabarty. "Hacking the Fabric: Targeting Partial Reconfiguration for Fault Injection in FPGA Fabrics". In: *Asian Test Symposium*. accepted. 2024.

[24] Hassan Nassar, Ming-Liang Wei, Chia-Lin Yang, Jörg Henke, and Kuan-Hsun Chen. "REAP-NVM: Resilient Endurance-Aware NVM-based PUF against Learning-based Attacks". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. accepted as extended abstract. 2025.

[25] Can Lehmann, Lars Bauer, Hassan Nassar, Heba Khdr, and Jörg Henkel. "Hardware/Software Co-Analysis for Worst Case Execution Time Bounds". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. accepted as extended abstract. 2025.

Abstract

With the rise of Artificial Intelligence (AI), big data, and digital public services, traditional Central Processing Units (CPUs) are reaching their limits. This necessitates specialized hardware accelerators alongside expanded CPU capabilities, leading to a complex computing landscape. Heterogeneous computing systems are increasingly deployed. Cloud service providers like Amazon Web Services, Microsoft Azure, and Google Cloud Platform are embracing heterogeneity, integrating Field-Programmable Gate Array (FPGA) instances to accelerate various applications, e.g., AI. However, security concerns persist, especially for sensitive data like medical and financial records. This work focuses on ensuring trust in cloud computing and establishing a secure processing framework for cloud data to prevent breaches. This dissertation presents four fundamental contributions, addressing these challenges.

The secure communication between the client and server is essential. Therefore, the first contribution focuses on authenticating the client server communication. Lightweight security solutions such as Physical Unclonable Functions (PUFs) are an alternative to hash functions, leveraging Integrated Circuit (IC) differences for unique responses. Attackers employ Machine Learning (ML) to predict PUF responses accurately. This contribution tackles ML-resilient PUFs, utilizing Multi-Level Cell (MLC) properties of Non Volatile Memory (NVM) and cascade architectures for dynamic changes. Additionally, PUF's reliability when used as dynamic accelerators on FPGAs or when are based on NVM is studied.

Encryption prevents plaintext leakage, yet covert channels can compromise keys. All cryptographic schemes, depend on key security; if compromised, overall security collapses. The second contribution focuses on covert-channel attacks. Two novel covert-channel threats for cloud FPGAs are presented, followed by a software and a hardware-based countermeasures. The first attack targets Trusted Execution Environments (TEEs) on Multi-Processor System-on-Chips (MPSoCs) containing FPGAs, exposing a temperature-based covert channel leveraging benign hardware accelerators. The second covert channel exploits power usage in Multi-tenant FPGAs, establishing communication between different tenants. The hardware-based countermeasure uses configurable ring oscillators to increase noise which reduces the leaks. Finally, the software-based countermeasure add delays to stop the communication.

To combat data leakage, the third contribution introduces a solution for accelerating secure processing on the cloud by accelerating Fully Homomorphic Encryption (FHE). FHE allows processing of encrypted data, mitigating trust and legal concerns by preventing data leakage. However, FHE comes with high computational costs, compounded by memory bottleneck issues, especially with large datasets. This contribution develops an FHE hardware accelerator using a recursive Karatsuba multiplier, intelligently mapped to 3D High Bandwidth Memory (HBM) to address memory bottlenecks. Additionally, a custom control interface maximizes HBM bandwidth utilization, aligning with FHE's memory access patterns.

Data leakage is not the sole concern; attackers may inject faults or trigger Denial of Service (DoS), rendering cloud usage unfeasible. Therefore, the fourth contribution develops two fault injection countermeasures targeting Power-Hammering on Multi-tenant cloud FPGAs, where attackers exploit power wasters like ring oscillators. The first is an online approach, deactivating malicious FPGA partitions swiftly upon detection. The second, an offline approach, analyzes bitstream metadata to identify malicious designs, challenging due to new attacks using modified benign accelerators.

All four contributions advance the state of the art. The first presents PUFs resilient to four state-of-the-art attacks, with FPGA prototypes costing as low as 300 Look Up Tables (LUTs) for the cascaded PUF. In the second, novel attacks achieve error rates below 5%, countered effectively by a novel countermeasure. The third introduces an accelerator with over 100× speedup compared to baseline solutions. Lastly, the fourth detects 98. 2% of fault injection attacks and speeds up the protection by more than 10×.

Zusammenfassung

Mit dem Aufstieg von AI, Big Data und digitalen öffentlichen Diensten stoßen traditionelle CPUs an ihre Grenzen. Dies erfordert spezialisierte Hardware-Beschleuniger neben erweiterten CPU-Fähigkeiten, was zu einer komplexen Computerlandschaft führt. Heterogene Computersysteme werden zunehmend eingesetzt. Cloud-Dienste-Anbieter wie Amazon Web Services, Microsoft Azure und Google Cloud Platform setzen auf Heterogenität und integrieren FPGA-Instanzen, um verschiedene Anwendungen, wie z.B. AI, zu beschleunigen. Doch bleiben Sicherheitsbedenken, insbesondere bei sensiblen Daten wie medizinischen und finanziellen Aufzeichnungen, bestehen. Diese Arbeit konzentriert sich darauf, Vertrauen in das Cloud-Computing sicherzustellen und einen sicheren Verarbeitungsrahmen für Cloud-Daten zu schaffen, um Sicherheitsverletzungen zu verhindern. Diese Dissertation präsentiert vier Beiträge, die sich mit diesen Herausforderungen befassen.

Die sichere Kommunikation zwischen dem Client und dem Server ist unerlässlich. Der erste Beitrag konzentriert sich auf die Authentifizierung der Client-Server-Kommunikation. Leichte Sicherheitslösungen wie PUFs sind eine Alternative zu Hash-Funktionen und nutzen Unterschiede in ICs für eindeutige Antworten. Angreifer setzen ML ein, um PUF-Antworten genau vorherzusagen. Dieser Beitrag betont ML-resistente PUFs, die die Eigenschaften von MLC in NVM nutzen und Kaskadenarchitekturen für dynamische Veränderungen einsetzen. Darüber hinaus wird die Zuverlässigkeit von PUFs untersucht, wenn diese als dynamische Beschleuniger auf FPGAs verwendet werden.

Während Verschlüsselung die Preisgabe von Klartext verhindert, können verdeckte Kanäle weiterhin Schlüssel kompromittieren und damit die kryptografische Sicherheit untergraben. Dieser Beitrag untersucht Angriffe über verdeckte Kanäle und stellt zwei neue Bedrohungen für Cloud-FPGAs vor. Der erste Angriff zielt auf FPGA-MPSoCs ab und nutzt einen temperaturbasierten verdeckten Kanal über harmlose Hardware-Beschleuniger. Der zweite Angriff

nutzt den Stromverbrauch in Multi-Tenant-FPGAs, um die Kommunikation zwischen Mandanten zu ermöglichen. Als Gegenmaßnahmen werden eine Hardwarelösung mit konfigurierbaren Ringoszillatoren zur Erhöhung des Rauschens und Reduzierung der Lecks sowie eine Softwarelösung vorgeschlagen, die durch das Einfügen von Verzögerungen die Kommunikation stört.

Um Datenlecks zu verhindern, führt der dritte Beitrag eine Lösung zur Beschleunigung der sicheren Verarbeitung in der Cloud durch die Beschleunigung von FHE ein. FHE ermöglicht die Verarbeitung von verschlüsselten Daten und mindert Bedenken hinsichtlich Vertrauen und rechtlicher Fragen, indem Datenlecks verhindert werden. FHE ist jedoch mit hohen Rechenkosten verbunden, die durch Speicherengpässe, insbesondere bei großen Datensätzen, noch verschärft werden. Dieser Beitrag entwickelt einen FHE-Hardware-Beschleuniger, der einen rekursiven Karatsuba-Multiplizierer verwendet, der intelligent auf 3D-HBM abgebildet wird, um Speicherengpässe zu beheben. Darüber hinaus maximiert eine benutzerdefinierte Steuerungsschnittstelle die Bandbreitenausnutzung von HBM und passt sich an die Speicherzugriffsmuster von FHE an.

Datenlecks sind nicht die einzige Sorge; Angreifer können Fehler injizieren oder DoS auslösen, was die Nutzung der Cloud unmöglich machen kann. Der vierte Beitrag entwickelt zwei Gegenmaßnahmen zur Fehlerinjektion, die sich auf Power-Hammering in Multi-Tenant-Cloud-FPGAs konzentrieren, bei dem Angreifer Energieverschwender wie Ringoszillatoren ausnutzen. Die erste ist ein Online-Ansatz, der bösartige FPGA-Partitionen schnell nach ihrer Erkennung deaktiviert. Der zweite, ein Offline-Ansatz, analysiert Bitstream-Metadaten, um bösartige Designs zu identifizieren, was aufgrund neuer Angriffe mit modifizierten harmlosen Beschleunigern eine Herausforderung darstellt.

Alle vier Beiträge bringen den Stand der Technik voran. Der erste stellt PUFs vor, die gegen vier Angriffe resistent sind, wobei FPGA-Prototypen nur 300 LUTs für den kaskadierten PUF benötigen. Der zweite wehrt neuartige Angriffe mit Fehlerraten unter 5% durch eine neue Gegenmaßnahme ab. Der dritte führt einen Beschleuniger ein, der im Vergleich zu Basislösungen über 100× schneller ist. Der vierte erkennt 98,2% der Fehlerinjektionsangriffe und beschleunigt den Schutz um mehr als 10×.

Research at CES

Research at the Chair for Embedded Systems (CES) tackles critical computing challenges, focusing on resource management for multicore systems, machine learning in resource-constrained systems, cross-layer security in emerging systems, and reconfigurable computing. These areas aim to improve system performance, energy efficiency, hardware longevity, and computational model adaptability [28, 94].

Reconfigurable Computing

Reconfigurable computing offers a flexible and adaptive approach to multicore resource management, dynamically locating and retracting resources as needed [38, 173]. CES focuses on hardware-software codesign, enabling real-time resource adjustments to improve efficiency and scalability [65, 67]. By configuring hardware according to software demands, CES develops systems that optimize performance and energy use, particularly effective in approximate computing, where reduced precision saves resources [95].

Resource Management for Multicore Systems

Resource management in multicore systems is crucial for energy efficiency, thermal management, and mitigating hardware aging [93, 132]. CES researchers developed techniques for dynamic resource allocation and workload balance, reducing energy use and controlling temperature [60, 92]. The research also addresses hardware aging, aiming to extend system life by reducing the wear on transistors. With increasing core density, heat management becomes more challenging. CES emphasizes dynamic thermal management to evenly distribute heat and avoid performance-degrading hotspots while ensuring the long-term reliability of hardware components by reducing performance degradation over time [96, 116].

Machine Learning in Resource-Constrained Systems

CES focuses on machine learning for low-resource environments, such as embedded systems and IoT devices, which are limited in processing power, memory, and energy [61, 162]. Researchers design lightweight, optimized ML models using techniques such as model compression, approximate computing, and energy-efficient inference. These methods enable advanced ML algorithms to run on modest hardware. Approximate computing, which simplifies computations to save energy while maintaining accuracy, is crucial for applying ML in resource-constrained systems.

Cross-layer Security in Emerging Systems

Security is paramount in today's computing landscape. With the advent of the Internet of Things (IoT), systems are increasingly interconnected, leveraging advanced network capabilities from recent technologies such as 5G. Notable among these interconnected systems are AI applications, as well as Edge and Cloud Computing. These systems are susceptible to novel attack vectors that exploit both hardware and software resources in unexpected ways. CES addresses the security of these systems using a cross-layer approach, which includes both the software and the hardware domains [84, 86].

Alignment of the Dissertation with Research at CES

The dissertation is primarily aligned with the cross-layer security research topic at CES. Moreover, it addresses key challenges of efficient resource management in FPGA-accelerated cloud systems, aligning with CES's other focuses. It contributes to mitigating covert channel threats and fault injection vulnerabilities in multi-tenant FPGAs and enhances machine learning-resilient PUFs for low-resource device authentication. These contributions support CES's goals of system efficiency, security, and long-term reliability.

Contents

List of Figures

List of Tables

List of Abbreviations

AaaS Acceleration-as-a-Service

AES Advanced Encryption System

AI Artificial Intelligence

ANV-PUF Arbiter NVM-based PUF

APUF Arbiter PUF

ASIC Application Specific Integrated Circuit

AXI Advanced eXtensible Interface

BRAM Block Random Access Memory (RAM)

CA Customer Application

CaPUF Cascaded PUF

CCI Complex Configrubale Inverter (CI)

CGRA Coarse Grained Reconfigurable Array

CI Configrubale Inverter

CKKS Cheon-Kim-Kim-Song

CMA-ES Covariance Matrix Adaptation Evolution Strategy

CPU Central Processing Unit

CRC Cyclic Redundancy Check

CRP Challenge-Response-Pair

CSP Cloud Service Provider

DAC Digital to Analog Converter

DDoS Distributed Denial of Service

DES Data Encryption Standard

DoS Denial of Service

DPR Dynamic Partial Reconfiguration

DRAM Dynamic RAM

DSP Digital Signal Processing

ECC Error Code Correction

FaaS FPGA-as-a-Service

FFT Fast Fourier Transform

FHE Fully Homomorphic Encryption

FPGA Field-Programmable Gate Array

GPU Graphics Processing Unit

HACC Heterogeneous Accelerated Compute Cluster

HBM High Bandwidth Memory

HE Homomorphic Encryption

HRS High Resistance State

IaaS Infrastructure-as-a-Service

IC Integrated Circuit

ICR Imprecise Control Regulator

IoT Internet of Things

LFSR Linear Feedback Shift Register

LR Logistic Regression

LRS Low Resistance State

LSTM Long Short-Term Memory

LUT Look Up Table

ML Machine Learning

MLC Multi-Level Cell

MLP Multi Layer Perceptron

MPSoC Multi-Processor System-on-Chip

NN Neural Network

NTT Number Transform Theory

NVM Non Volatile Memory

PaaS Platform-as-a-Service

PBS Programmable Bootstrapping

PC Pseudo Channel

PCM Phase Changing Memory

PDN Power Distribution Network

PHE Partial Homomorphic Encryption

PL Programmable Logic

PLPUF Pseudo Linear Feedback Shift Register (LFSR) PUF

POK Physically Obfuscated Key

PRNG Pseudo Random Number Generator

PRR Partial Reconfigurable Region

PS Processing System

PUF Physical Unclonable Function

RAM Random Access Memory

RCRO Runtime-Configurable Ring Oscillator (RO)

RF Random Forest

RNG Random Number Generator

RO Ring Oscillator

RPU Real-time Processing Unit

RRAM Resistive RAM

SA Sense Amplifier

SCI Simple CI

SHA Secure Hash Algorithm

SHE Somewhat Homomorphic Encryption

SOTRAM Spin Orbit Torque RAM

SRAM Static RAM

STTRAM Spint Transfer Torque RAM

SVM Support Vector Machine

TA Trusted Application

TCC Thermal Covert Channel

TDC Time to Digital Converter

TEE Trusted Execution Environment

TFHE Fast Fully Homomorphic Encryption over the Torus

TMR Triple Modular Redundancy

TVLA Test Vector Leakage Assessment

1 Introduction

Cloud computing transforms industries by offering scalable and flexible infrastructure [88, 143]. It supports diverse applications, from data storage to real-time analytics. In healthcare, it stores and processes patient data for remote diagnostics and personalized treatments. Financial services use the cloud for trading, fraud detection, and transactions [191]. AI relies on cloud platforms for computational power to train models for applications ranging from autonomous vehicles to recommendation systems. IoT devices highlight the cloud's role in handling data from sensors in smart cities, healthcare, and industrial automation.

Cloud computing has rapidly transformed, driven by advancements in networking, virtualization, and distributed computing [66, 110]. It began from grid computing and evolved with virtualization technologies, allowing dynamic scaling of shared infrastructure. The early 2000s saw a significant shift with services like AWS introducing Infrastructure-as-a-Service (IaaS). This evolved to recently include Platform-as-a-Service (PaaS), and Acceleration-as-a-Service (AaaS) [58] simplifying application deployment and management. The ecosystem now includes serverless computing, edge computing, and hardware acceleration, which is essential for AI and big data analytics [139]. Cloud computing continues to evolve, enhancing flexibility and scalability.

The rapid evolution of cloud computing has brought both unprecedented computational capabilities and significant architectural challenges. As modern applications demand increasingly complex and efficient processing, cloud infrastructures must evolve to provide performance and flexibility. Hardware accelerators, such as FPGAs, Graphics Processing Units (GPUs), and Application Specific Integrated Circuits (ASICs), play a crucial role in this shift, enabling cloud providers to offer tailored acceleration for diverse workloads [40, 143, 185] leading to the introduction the concept of accelerated cloud systems [58].

A prominent example of accelerated cloud systems is the concept of FPGA-as-a-Service (FaaS), which allows users to leverage FPGA technology for customized tasks. Offering FaaS and other modern computation capabilities led to the rise of MPSoCs integrating FPGAs, as shown in Figure 1.1. These systems allow for runtime reconfiguration, enabling the modification of hardware implementations to adapt to changing workloads. This capability is especially relevant for cloud systems where multiple users share the same resource. In case of an FPGA, the resource can be shared by assigning each user its own Partial Reconfigurable Region (PRR) to accelerate its application [13, 18].

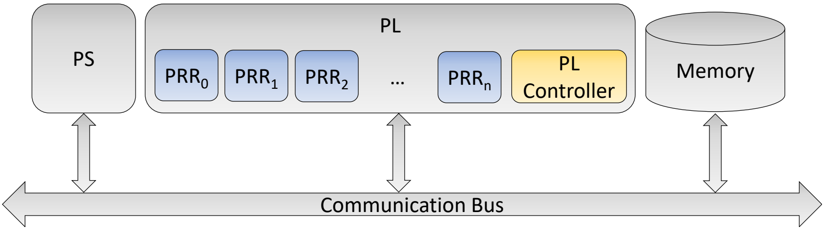


Figure 1.1: Example of a modern FPGA-MPSoC. The Processing System (PS), i.e., the CPUs and the Programmable Logic (PL), i.e., the FPGA are both integrated on the same chip. Moreover, both PS and PL can access the memory directly via the communication bus. The PL is divided into several PRRs to enable the acceleration of several applications at the same time.

While FaaS and AaaS [58] in general increase the efficiency of processing in cloud systems, they also introduce new attack possibilities, e.g. malicious exploitation of shared mediums in multi-user setups [16, 83]. Therefore, ensuring the security of these accelerated cloud systems is critical. Introducing mechanisms to safeguard sensitive computations, authenticate communication, and prevent unauthorized access in accelerated cloud systems became crucial. This creates a growing need for robust security solutions that can accommodate the evolving landscape of accelerated computing without compromising performance or flexibility [88, 130, 191].

1.1 Integration of Cloud Computing in Daily Life

To show how cloud computing became part of normal life, consider an IoT-based health monitoring system deployed in a hospital. The patients are using wearable medical devices that continuously record patient vitals. These resource-constrained devices rely on a cloud-based server for computationally intensive tasks, such as analyzing large datasets of medical data. Given the sensitive nature of the data, secure and efficient communication between the client devices and the cloud server is essential. Moreover, while computing sensitive data, the Cloud Service Providers (CSPs) have to ensure that no data leakage occurs.

Typically for such applications, the health provider will not have its own cloud infrastructure but will instead use a service from a public cloud such as AWS or Azure [88, 131]. This is a client-server architecture where the client is the resource-constrained IoT health device with limited computational power, necessitating the offloading of processing-intensive tasks to an external cloud-based server [31]. Figure 1.2 shows an example of this client-server architecture in accelerated cloud systems.

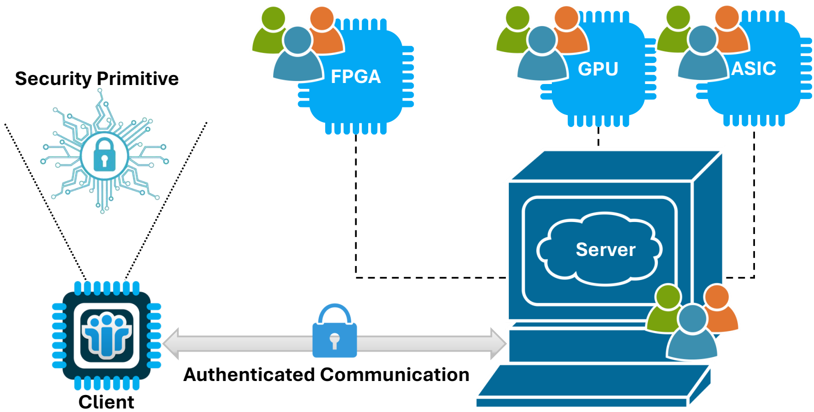


Figure 1.2: System model illustrating the client-server relationship in a cloud setup. The client, with limited resources, relies on the semi-trusted cloud server for computational tasks while ensuring authenticated communication. The cloud server shares the resources between the different users.

The client relies on the server for complex computations while maintaining control over sensitive information [31, 130]. The client has to generate cryptographic keys using a security primitive, e.g., PUF and prove its integrity, e.g., by attestation [70, 198]. Once authenticated, the client securely offloads tasks to the server, which executes them and returns the results [31]. This authenticated communication is important to ensure that no leakage, impersonation, or Distributed Denial of Service (DDoS) attacks occur.

The cloud server performs most of the heavy computations for several users at once. Therefore, the server is considered semi-trusted, meaning that, although it is not actively malicious, it remains susceptible to attacks and data leakage [191]. This is particularly critical in cases where resources are shared among multiple users [80]. The attack surface becomes wide and complex when the server uses several possible hardware accelerators such as GPUs [185], FPGAs [110], and ASICs [111].

Challenges for Accelerated Cloud Systems

In cloud-based client-server architecture with resource-constrained clients, several security and performance issues arise. These clients offload computationally expensive tasks to a cloud-based accelerated server, improving efficiency but introducing vulnerabilities from the shared, multi-user nature of clouds and the need for secure communication.

One primary issue is that resource-constrained client devices may lack the power for heavy cryptographic tasks, making them vulnerable to replay and impersonation attacks. Ensuring that only legitimate clients can access server resources is critical, but current authentication mechanisms may be too burdensome for low-power devices [160]. Resource-constrained devices might use Physical Unclonable Functions (PUFs) for authentication. However, PUFs are susceptible to ML-modeling attacks [39], where attackers can predict the PUF's behavior, compromising authentication.

Additionally, despite the advantages of hardware accelerators, their adoption in cloud systems presents several key challenges. The focus in this dissertation is on the challenges introduced by integrating FPGAs in the cloud systems as they open attractive opportunities to increase efficiency of the computation but at the same time introduce significant vulnerabilities [16, 40, 120, 198]. The flexibility of FPGAs can be exploited to introduce new attacks, especially in multi-user cloud systems where FPGAs are used as a shared medium. Consequently, the risk of security breaches and data leakages increases significantly [80, 119]. Moreover, when CSPs like AWS and Microsoft Azure offer customizable FPGA-based accelerators, these concerns are further amplified. This is because it allows clients to program and configure the hardware according to their specific needs, raising the potential for malicious configurations [31, 63, 161].

One threat in such multi-user systems is the exploitation of covert channels, such as power or thermal channels, by malicious users, leading to information leakage. These attacks are difficult to detect or mitigate with traditional security measures [84]. As mentioned above, the problem worsens with hardware acceleration using FPGAs where users can craft accelerators that leak information more efficiently than normal CPUs [81]. Such attacks are powerful, as they can occur remotely without physical access.

Further risks include attackers manipulating the shared power infrastructure or hardware vulnerabilities to induce faults in other users' computations, compromising data integrity and potentially causing system-wide failures. Again, this is a prominent threat for FPGAs. Several works show that if cloud providers use one FPGA for various workloads from several users, it is easy for them to perform remote DoS and fault injection attacks [80, 122].

Lastly, a critical opportunity is securing the processing of encrypted data in privacy-sensitive systems using HE. Integrating HE into accelerated cloud infrastructures is challenging as it is highly compute and memory intensive [53]. Therefore, existing solutions may not perform the computation in a timely manner for the user. FaaS offers an attractive opportunity [194]. By tailoring the accelerator and leveraging near-memory capabilities of FPGA systems [164], HE can be effectively accelerated.

In summary, the following challenges are tackled in this dissertation:

- **Machine Learning Threats to PUFs:** PUFs are used for device authentication and cryptographic key generation, but they are vulnerable to attacks that leverage machine learning techniques to model and predict their responses [160, 199].
- **Security Vulnerabilities in FPGAs:** The reconfigurability of FPGAs introduces vulnerabilities that can be exploited through covert-channel attacks. In multi-user systems, these threats are particularly concerning as attackers can easily exploit hardware accelerators to amplify the leaks used for the covert communication [78].
- **Computational Bottlenecks in FHE:** FHE is a promising cryptographic technique that allows computations on encrypted data, ensuring data privacy. However, FHE is computationally intensive, requiring

significant resources to perform encrypted additions and multiplications [54]. Accelerating FHE using hardware, such as FPGAs, is crucial to overcoming these bottlenecks.

- **Fault Injection Attacks:** Voltage-based fault injection attacks, such as Power-Hammering [4], present significant threats to the reliability and security of FPGAs. These attacks can induce faults that compromise the system's integrity and availability [119].

1.2 Contributions

This dissertation addresses the security and performance challenges of accelerated cloud systems with focus on FPGA-based acceleration by proposing the following four major contributions:

1. **Client-Server Authentication via ML-Resilient PUFs:** Novel PUFs are introduced, designed to withstand machine learning-based modeling attacks. By leveraging architectural and technological properties, these PUFs provide a more secure authentication method, suitable for lightweight client devices in cloud-based systems.
2. **Identifying and Mitigating Covert Channels on FPGA-Accelerated Cloud Systems:** New covert channel attack vectors in FPGA-accelerated cloud systems are identified and analyzed. Countermeasures are developed to prevent information leakage in multi-user cloud infrastructures, ensuring secure hardware-accelerated services.
3. **Data Leakage Mitigation in Cloud Systems Using FPGA-Accelerated Homomorphic Encryption:** A high-performance accelerator for Fast Fully Homomorphic Encryption over the Torus (TFHE) is developed, utilizing HBM-enabled FPGAs to overcome the memory bottlenecks inherent in FHE. The design includes a scalable recursive multiplier, significantly enhancing computation speed for encrypted data processing.
4. **Eliminating Fault Injection Threats in Multi-tenant FPGAs:** A combined defense mechanism is presented, integrating offline and online monitoring to mitigate remote fault injection attacks in cloud

FPGAs. This approach protects the integrity and availability of shared resources in multi-user systems.

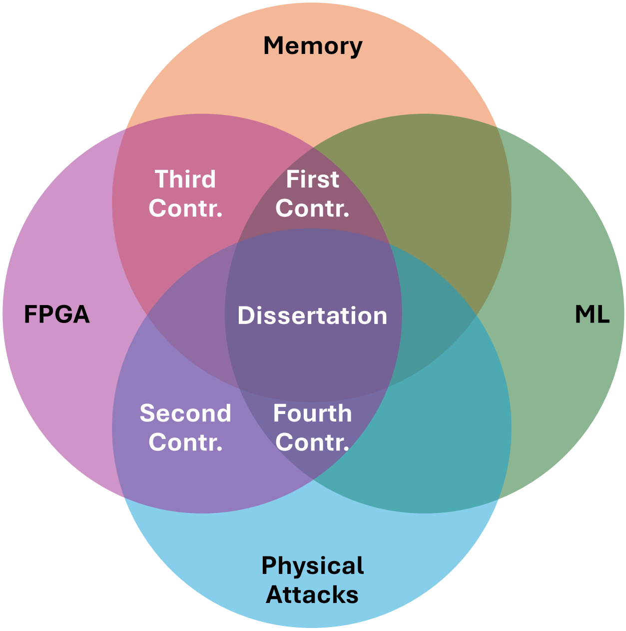


Figure 1.3: The domains tackled to achieve the contributions and face the challenges in the dissertation. Four domains: ML, FPGA Memory and Physical Attacks are tackled. Each domain is used in at least two contributions.

To address the challenges and achieve the contributions, four domains (Memory, FPGA, ML, and physical attacks) are studied as Figure 1.3 shows. Each domain is studied for more than one contribution and the domain of FPGAs is essential for all four contributions. Moreover, per contribution, at least one new tool or accelerator is developed.

The importance of this dissertation lies in its direct response to the growing use of FaaS for accelerating computational tasks in diverse applications in cloud systems. As the adoption of FaaS grows, the need for secure, efficient, and resilient systems becomes crucial. The vulnerabilities introduced by FPGAs, particularly in multi-user systems, pose a significant threat to data integrity and system reliability. This dissertation contributes to the field of hardware security by addressing these concerns, developing solutions that protect against potential attacks, and also enhancing the performance of secure computation on cloud-based systems.

Validation of the Contributions

To validate the contributions of this dissertation, several experiments are executed. As mentioned above, the primary focus and contribution of this dissertation is on hardware acceleration using FPGAs in accelerated cloud systems and the challenges and threats it faces. As such, the majority of the experimental efforts to validate the contributions are centered around the deployment and testing of FPGA-based systems. Whether a hardware solution can be adapted for general use or specifically building an FPGA accelerator, the FPGA remains the core of the experimental setup.

To closely mirror real-world conditions in cloud computing systems, a dual approach is employed. First, an FPGA development board is connected directly to a powerful in-house server, allowing full control over variables and providing a stable system for testing designs. Second, the experiments are extended to a more realistic cloud infrastructure using the Heterogeneous Accelerated Compute Cluster (HACC) from AMD at ETH Zurich [97]. HACC offers the computational resources and cloud infrastructure (as shown in Figure 1.4) necessary for evaluating the performance and security of the designs under conditions that closely resemble commercial cloud services.

In addition to FPGA-based experiments, the framework includes a suite of tools tailored for different aspects of the research. For any mathematical modeling required, such as the development of delay models for PUFs, the validation relies on MATLAB. It provides the computational power and flexibility needed to accurately describe the behavior of the devices through mathematical equations. When the research involves the detection of malicious code execution on the cloud via ML, the validation incorporates ML techniques developed in Python. Python's rich ecosystem of ML libraries enables the efficient analysis of large datasets and identify patterns indicative of malicious activities.

Finally, for circuit-level simulations, particularly when evaluating noise models or other low-level characteristics, SPICE is utilized. It allows to simulate

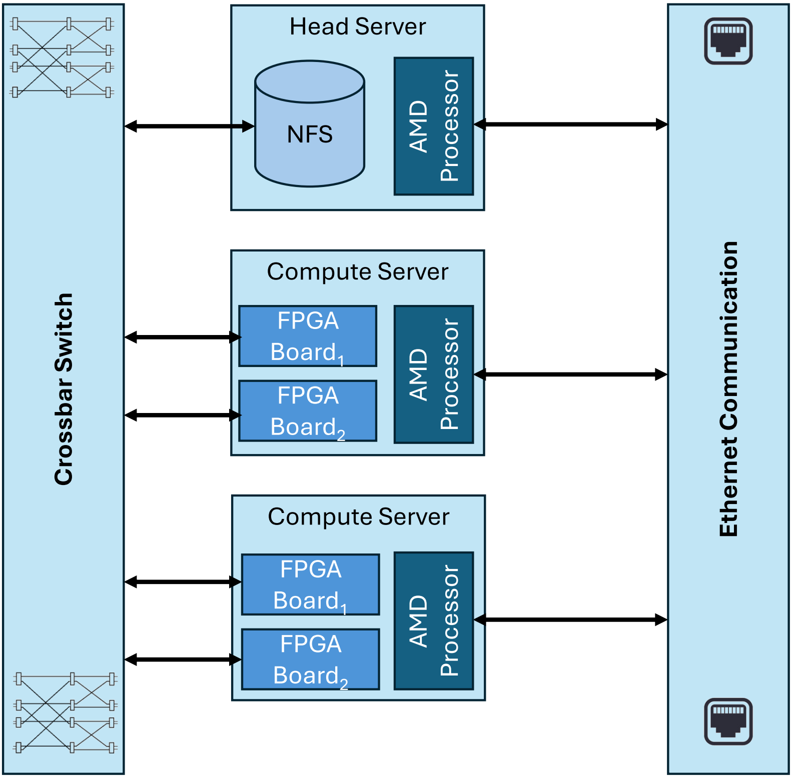


Figure 1.4: The architecture of HACC. Several computational nodes that are directly connected to FPGA boards. Communication between the boards is possible via a switch and nodes can communicate to the outer world via Ethernet [97].

the behavior of the circuits with high accuracy, providing insights into how they will perform under various conditions.

1.3 Dissertation Outline

The remainder of this dissertation is structured as follows:

- **Chapter 2** provides an overview of cloud computing architectures, with a focus on the use of FPGAs and hardware accelerators. It also discusses the security challenges in cloud systems, FHE, and PUFs.
- **Chapter 3** introduces the proposed PUFs that are resilient to machine learning attacks and their usage to authenticate the communication between clients and servers.
- **Chapter 4** presents the identification of new covert channels in FPGA-accelerated cloud systems and the countermeasures developed to mitigate the threats of data leakage.
- **Chapter 5** introduces the proposed hardware accelerator for FHE to eliminate the threat of data leakage.
- **Chapter 6** discusses various fault injection attacks on FPGAs in cloud systems and the countermeasures proposed to protect against them.
- **Chapter 7** recaps the contributions of the dissertation and suggests directions for future research.

2 Background

Modern computing is facing increasing challenges due to the growing demand for processing power in applications such as artificial intelligence, big data, and real-time analytics. Traditional architectures, which rely primarily on CPUs, are reaching their limits in terms of speed, efficiency, and scalability. This situation has necessitated a shift toward cloud computing, which offers scalable and flexible resources to meet these demands [13, 23, 25, 102, 110].

2.1 Solving the Memory Bottleneck

As data continues to grow exponentially, memory bandwidth becomes a critical bottleneck. Several works try to eliminate this bottleneck through near-memory and in-memory processing [19, 37, 128, 163]. HBM is one of the significant advancements designed to address this issue. HBM vertically stacks memory dies interconnected by through silicon vias, significantly reducing the physical footprint while greatly enhancing data transfer rates, as shown in Figure 2.1. This design shifts from a 2D to a 2.5D architecture, facilitating near-memory processing. The result is a substantial increase in bandwidth, which enables faster data access and transmission between the logic core, such as CPUs or FPGAs, and memory, optimizing overall system performance and efficiency [106, 127, 175].

In addition to HBM, NVM is another critical emerging technology that is changing the landscape of memory technologies [15, 19]. Unlike traditional Dynamic RAM (DRAM) and Static RAM (SRAM), NVM retains its state without requiring a constant power supply, which is crucial for energy efficiency and data retention in power-sensitive applications. NVM also supports MLC, which allows multiple states to be coded within a single cell, thereby increasing storage density and enabling a range of new computing possibilities,

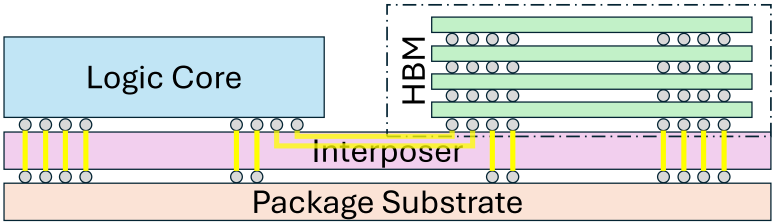
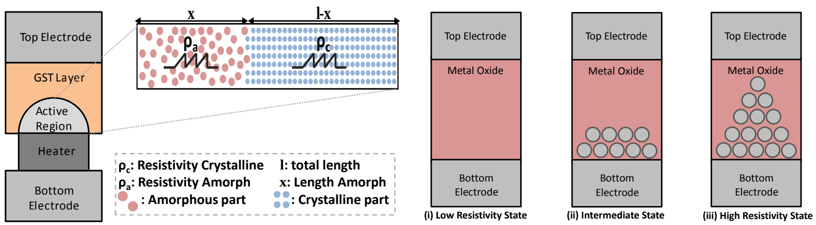


Figure 2.1: HBM Integration with chip in a 2.5D manner. The memory stack is directly connected with the logic core via an interposer on the same package.

such as processing in memory and the implementation of PUFs for security purposes [105].

Phase Changing Memory (PCM) and Resistive RAM (RRAM) are two of the most famous examples of NVM. PCM operates by heating a chalcogenide material to different temperatures, causing it to switch between amorphous and crystalline states. These states have distinct resistive properties, with the amorphous state having high resistivity High Resistance State (HRS) and the crystalline state having low resistivity Low Resistance State (LRS). By carefully controlling the heating and cooling process, intermediate states can be achieved, each corresponding to a different resistance level, as shown in Figure 2.2a [15, 151].



(a) PCM cell. The chalcogenide layer can be quenched to enter the amorphous state or heated to transition into the crystalline state.

(b) RRAM operation steps. The HRS and LRS correspond to the presence or absence of a metallic filament within the cell [196].

Figure 2.2: PCM and RRAM NVM technologies

Similarly, RRAM operates by forming or dissolving a metallic filament within a metal oxide layer between two electrodes. The formation of the filament reduces the cell's resistance to LRS, while its dissolution increases resistance to HRS. By applying a specific voltage, the filament can be partially formed or dissolved, creating intermediate resistance states. These multiple resistance states enable RRAM to store more information per cell, as illustrated in Figure 2.2b [196].

As these novel memory systems become integrated into modern computing infrastructures, they play a pivotal role in overcoming the limitations of traditional architectures. HBM and NVM do not only provide solutions to current memory bottlenecks but also pave the way for more advanced high-performance computing systems that are capable of handling the increasing demands of today's data-driven world [37]. Although these novel memory systems are a breakthrough in solving memory bottlenecks, the need for high computational capabilities is still a challenge in modern computing. A typical resource-constrained device is not capable of dealing with the high computation demands of AI, big data, etc.

2.2 Cloud Computing and Heterogeneous Systems

To address the modern computing challenges, cloud computing has become a critical infrastructure, enabling on-demand access to a shared pool of configurable resources such as networks, servers, storage, and applications. The ability to scale resources dynamically to meet varying demands has led to the widespread adoption of cloud services across multiple industries [102, 204].

One of the key developments in cloud computing is the integration of heterogeneous MPSoCs, which combine traditional CPUs with specialized hardware accelerators such as GPUs, Real-time Processing Units (RPUs), FPGAs, and Coarse Grained Reconfigurable Arrays (CGRAs) as Figure 2.3 shows. These heterogeneous systems offer significant performance improvements for tasks such as machine learning, data processing, and cryptography by leveraging the strengths of each hardware component [143, 185]. For instance, FPGAs are particularly valued in cloud systems due to their reconfigurability, allowing them to be customized to specific workloads and applications [31, 110].

The rise of AI and big data analytics has further fueled the demand for more heterogeneous computing in cloud systems. These applications require

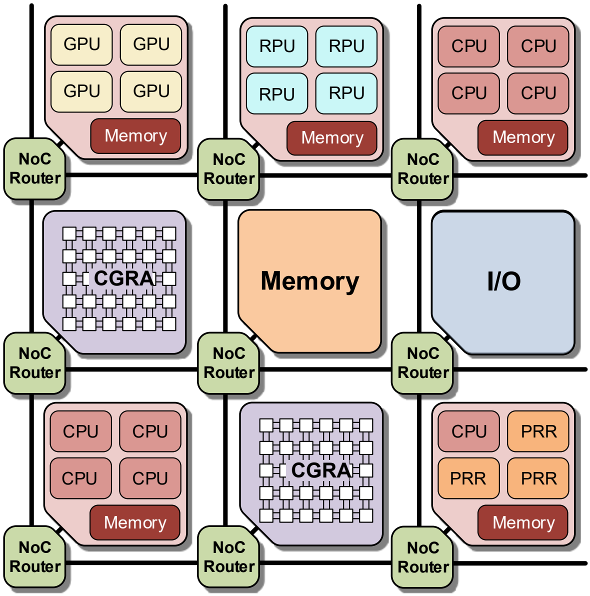


Figure 2.3: Heterogeneous computing system architecture integrating CPUs, RPUs, GPUs, CGRAs, and FPGAs in form of PRRs. Modified from the InvasIC System [181]

massive computational power and the ability to process large datasets in real-time, which traditional CPUs alone cannot efficiently handle. By offloading specific tasks to GPUs or FPGAs, cloud providers can significantly accelerate processing times and reduce the overall cost of computation [110, 204].

In addition to performance benefits, heterogeneous systems in cloud computing also offer advantages in terms of energy efficiency. FPGAs and ASICs, for example, can be optimized for power consumption, making them ideal for workloads that require high computational throughput with minimal energy use. This energy efficiency is particularly important in data centers, where reducing power consumption translates directly to lower operational costs and a smaller environmental footprint [102, 143].

However, the adoption of heterogeneous computing systems also introduces complexity in terms of resource management. Ensuring that different hardware components work seamlessly together requires sophisticated software frameworks and scheduling algorithms. These frameworks must manage the allocation of tasks to the appropriate hardware accelerator, handle data transfer between different processing units, and optimize performance across the entire system [31, 102]. The development of these frameworks is a critical area of research, as it directly impacts the scalability and efficiency of cloud services.

2.2.1 FPGA Integration in Cloud Computing

In this dissertation the focus is mainly on FPGAs as the hardware accelerator in cloud computing. FPGAs have transitioned from standalone devices to integral components of cloud infrastructure. Major CSPs like AWS and Microsoft Azure now offer FPGA-as-a-Service (FaaS), enabling users to deploy customized hardware for specific tasks [31, 161]. This trend reflects the growing demand for high-performance, low-latency computing solutions in areas such as real-time data processing, machine learning, and cryptography [110].

As the concept of FaaS expands in cloud systems, the development of specialized frameworks to manage these services has become crucial [97, 152]. Cloud providers have invested in creating tools and platforms that simplify the deployment, scaling, and management of FPGA resources. These platforms typically offer users the ability to program FPGAs remotely, select pre-configured accelerators for common tasks, and monitor the performance of their FPGA deployments in real time [31, 161]. Moreover, they also utilize FPGA-MPSoCs, as Figure 2.4 shows, to give the users the flexibility to choose what parts to be done in hardware and what parts to be done in software. This user-friendly approach has lowered the entry barrier for utilizing FPGA technology, allowing a broader range of industries to leverage the power of hardware acceleration.

Despite these advancements, the integration of FPGAs into cloud computing continues to evolve, with ongoing research aimed at further improving the efficiency and security of these systems. One area of focus is the development of dynamic reconfiguration techniques that enable FPGAs to adapt to changing workloads without requiring downtime. These techniques are critical for applications that demand high availability and reliability, such as

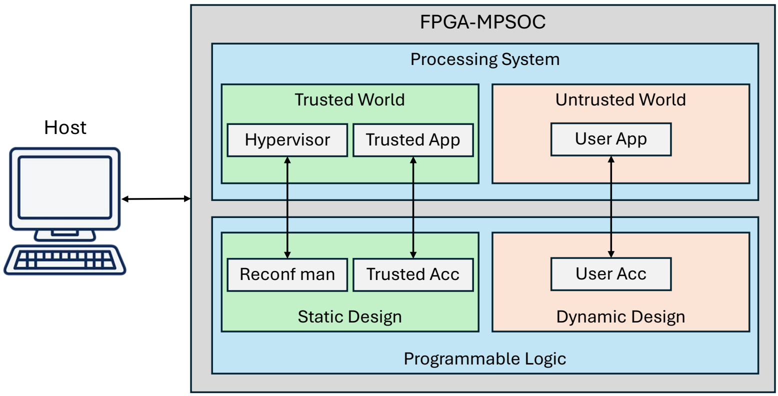


Figure 2.4: Overview of the setup for FPGA-as-a-Service (FaaS). Each host is connected to an FPGA-MPSoC. User Apps can trigger trusted apps to use trusted accelerators from the PL or they can dynamically reconfigure custom accelerators via the reconfiguration manager and the hypervisor.

financial trading systems and real-time analytics [13, 63]. Another research direction involves enhancing the security of FPGAs in cloud systems, particularly against covert-channel attacks and other forms of hardware-based threats [171].

2.2.1.1 Multi-Tenant FPGAs

The concept of multi-tenant FPGAs has emerged as a promising solution to maximize resource utilization and reduce costs in cloud computing systems that offer FaaS. As FPGAs continue to grow in capacity and performance, the ability to partition a single FPGA into multiple isolated regions, each serving a different user or application, has gained significant attention in both academia and industry [110, 204].

Multi-tenant FPGAs enable CSPs to offer FaaS to multiple clients simultaneously. This approach allows clients with varying computational requirements to share the same physical FPGA hardware without interfering with each other. The FPGA can be logically divided into several regions or partitions, each assigned to a different tenant. These partitions can be dynamically reconfigured to accommodate changing workloads, making multi-tenant FPGAs highly adaptable to diverse computational tasks [40, 57].

The implementation of multi-tenant FPGAs in cloud systems leverages partial reconfiguration, which allows individual PRRs of the FPGA to be reconfigured without affecting the operation of other partitions. This capability is crucial for achieving the flexibility and efficiency required in multi-tenant setups. Partial reconfiguration enables CSPs to update, reprogram, or reallocate resources on-the-fly, providing a seamless user experience and optimizing resource usage [40].

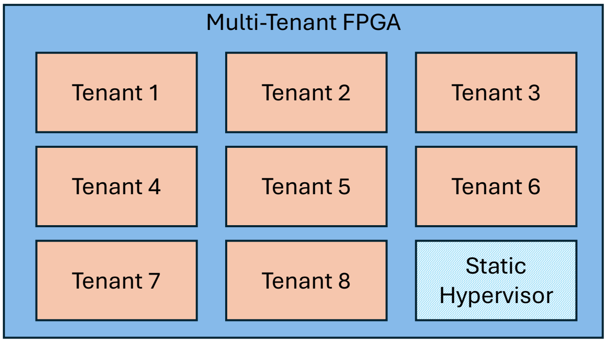


Figure 2.5: Conceptual representation of a multi-tenant FPGA setup. The FPGA is divided into multiple partitions, each allocated to a different tenant. These partitions can be dynamically reconfigured based on the tenants' needs.

One of the key challenges in realizing multi-tenant FPGAs is ensuring efficient resource management. Given the diverse and often unpredictable nature of workloads in a cloud system, CSPs must implement sophisticated scheduling algorithms and resource management frameworks to allocate FPGA resources effectively. These frameworks must balance the computational demands of different tenants, minimize latency, and prevent resource contention, all while maximizing the utilization of the FPGA fabric [110].

Moreover, the integration of multi-tenant FPGAs into existing cloud infrastructures requires careful consideration of compatibility with other components, such as CPUs, GPUs, and memory systems. The communication between these components must be optimized to ensure that the performance benefits of FPGAs are fully realized in a multi-tenant system. This includes the development of high-speed interconnects and efficient data transfer protocols that minimize overhead and latency [204].

2.3 Security Challenges for FPGA-as-a-Service in Cloud systems

The deployment of FPGAs in cloud computing systems presents several security challenges, including data leakage, covert-channel attacks, and fault injection attacks. These threats are exacerbated in multi-tenant settings, where multiple users share the same hardware resources.

As CSPs increasingly offer FPGA-based acceleration services, the need for robust security measures becomes paramount. The flexible nature of FPGAs allows users to reprogram hardware dynamically, which, while beneficial for performance, opens up vulnerabilities that can be exploited by attackers. These vulnerabilities can lead to unauthorized access, data breaches, and disruption of services. In particular, multi-tenant systems are susceptible to attacks where one tenant can potentially interfere with or extract data from another tenant's resources. Addressing these security challenges requires a combination of hardware-based protections, secure design practices, and continuous monitoring for anomalies [171, 198].

2.3.1 Data Leakage and Covert-Channel Attacks

The increasing integration of FPGAs into cloud systems has brought significant security concerns, particularly in multi-tenant systems where multiple users share the same physical hardware. One of the primary threats in such systems is data leakage, often facilitated by covert-channel attacks. These attacks exploit indirect information leaks through various physical channels, such as power consumption and temperature variations, to extract sensitive data [20, 80].

Covert-channel attacks on FPGAs are particularly concerning because of the flexibility and reconfigurability that these devices offer. Attackers can craft malicious circuits that, when deployed on shared FPGAs, manipulate shared resources such as Power Distribution Networks (PDNs) or thermal characteristics to create covert communication channels. These channels can be used to leak information between isolated workloads or from secure areas of the FPGA to an external observer [42].

One of the most studied covert-channel vectors is based on power-based attacks. In a multi-tenant FPGA system, power-based covert-channel attacks leverage the shared PDN to infer operations occurring in neighboring tenants' circuits. Attackers may deploy circuits that cause specific power consumption patterns, which can then be monitored to extract information. For instance, small variations in power usage, which correlate with different data being processed, can be amplified and measured to reconstruct the processed data. This method is particularly effective because it does not require physical access to the FPGA and can be executed remotely, making it a potent tool for attackers [77, 82].

Another critical covert-channel attack vector involves temperature-based, i.e., thermal covert channels. In these attacks, an attacker modulates the temperature of the FPGA by varying the activity levels of certain circuits. For example, by running intensive computations on one part of the FPGA, the temperature in that region increases. This change can be detected by other circuits on the FPGA, which are sensitive to temperature variations, effectively creating a covert communication channel between them. Thermal covert channels are particularly stealthy, as they exploit the natural heat dissipation properties of the chip, making them difficult to detect using traditional security mechanisms [137, 182].

The implementation of covert channels on FPGAs has evolved significantly, with recent studies demonstrating that these channels can achieve relatively high data transmission rates while maintaining low error rates. This is achieved by carefully controlling the modulation of the covert signal and optimizing the encoding schemes to reduce detection chances [82].

Both power-based and thermal covert channels pose severe risks to the confidentiality and integrity of data in FPGA-based cloud systems. These attacks exploit the shared nature of resources in multi-tenant systems, allowing malicious actors to bypass logical isolation mechanisms. The stealthy nature of these channels, especially thermal-based ones, makes them particularly challenging to detect and mitigate. As FPGAs continue to be integrated into cloud infrastructures, it is essential to develop robust countermeasures to protect against these sophisticated covert-channel attacks [82, 171].

2.3.2 Threat of Fault Injection Attacks in Cloud systems

Fault injection attacks are a critical threat to cloud systems, particularly with the increasing adoption of FPGAs in multi-tenant settings. Traditionally, fault attacks required physical access to the hardware, where attackers could manipulate clock signals or induce voltage drops to cause timing violations and other faults in ICs [195]. However, with cloud FPGAs, attackers no longer need physical proximity to execute these attacks. They can exploit the shared nature of cloud resources to induce faults remotely, affecting not just their own virtualized hardware, but also the resources of other tenants on the same physical device [80, 119].

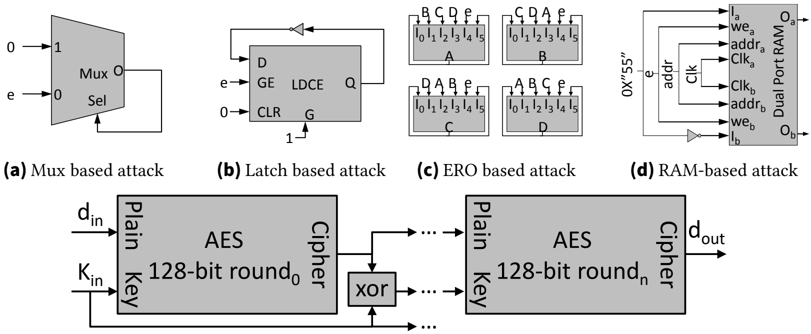
In cloud systems, fault injection attacks can lead to severe consequences, such as DoS, data corruption, and security breaches. The shared PDN in FPGAs makes them particularly vulnerable, as attackers can create high power-consuming circuits that destabilize the power supply, leading to faults in other tenants' computations. This type of attack not only disrupts service but can also result in significant financial losses for CSPs due to downtime and the need for manual intervention to restore services [125].

2.3.2.1 Power-Hammering on FPGAs

Power-hammering, also referred to as voltage-based attacks, is another significant threat in FPGA-accelerated cloud systems. These attacks involve manipulating the power consumption of an FPGA by creating circuits that consume excessive power, leading to voltage drops that can cause faults or DoS conditions [80]. The goal of power-hammering is either to disrupt the operation of the FPGA entirely (causing a DoS) or to introduce faults that can be exploited for malicious purposes.

The distinction between power-hammering attacks that aim to cause DoS and that aim to inject faults is important. DoS attacks typically involve circuits that create sustained high power consumption, leading to a significant voltage drop that eventually causes the FPGA to crash. In contrast, fault injection attacks are more subtle and precise, using circuits that generate transient power spikes timed to coincide with specific operations in the FPGA, causing faults without necessarily crashing the system [119, 159].

Figure 2.6 illustrates various power-wasting circuits that have been used in power-hammering attacks. These include self-oscillating circuits, which are particularly dangerous and can cause attacks even at low utilization. Multiplexers (Figure 2.6a), latches (Figure 2.6b), and even standard components like Block RAMs can be configured to consume excessive power, making them effective tools for power-hammering [29, 126, 176].



(a) Mux based attack **(b)** Latch based attack **(c)** ERO based attack **(d)** RAM-based attack

(e) Advanced Encryption System (AES)-based Power-Hammering attack from [159]. By xoring the key and the cipher and using a special input pattern and key pattern, a power-hammering attack is successful.

Figure 2.6: Different types of power wasters, suggested in [29, 121, 126, 159, 176].

Preventing power-hammering attacks requires advanced monitoring and detection mechanisms. Techniques such as dynamic voltage and thermal sensors can detect anomalies in power consumption and take corrective action, such as throttling the power supply or shutting down the affected regions of the FPGA [80, 158]. However, these tools are typically slow and need pre-identification or at least suspicion of maliciousness of the circuit. This is easy with simple attacks from above. Attackers responded with creating stealthier attacks that mimic legitimate circuit activity e.g., AES as Figure 2.6e shows. This stealthy nature makes them difficult to counter and identify these attacks [16].

2.4 Homomorphic Encryption

To combat data leakage in cloud systems, CSPs can leverage HE. The concept of HE, known as "privacy homomorphism," was proposed in 1978 [165] and practically implemented in 2009 [75]. Homomorphic Encryption is both encryption and a homomorphism, a function preserving group structure. Given two groups $(G, \cdot)$ and $(H, \times)$, function $h : G \longrightarrow H$ is a group homomorphism if

$$h(u \cdot v) = h(u) \times h(v),$$

for all u, v in G. This allows operations on encrypted data.

Let $(P, \diamond, C, \circ, e, d)$ be the homomorphic encryption scheme, P the plaintext group with operation $\diamond$, and C the ciphertext group with operation $\circ$. Functions e and d denote encryption and decryption algorithms, respectively. By definition, $e : P \longrightarrow C$, $d : C \longrightarrow P$, $\diamond : P \longrightarrow P$, and $\circ : C \longrightarrow C$ apply. Note that $\diamond$ is an arbitrary operation on P (e.g., addition or multiplication), while $\circ$ is defined by the homomorphic encryption scheme. Given plaintexts $a \in P$ and $b \in P$, the scheme satisfies

$$e(a) \circ e(b) = e(a \diamond b).$$

. This enables performing modified operation $\circ$ on encrypted data, producing the same result upon decryption as the intended operation $\diamond$ on plaintext, without data knowledge in between. The final result is evaluated at decryption as

$$d(e(a) \circ e(b)) = a \diamond b.$$

.

The mathematics behind HE relies on Eigenvalue and Eigenvector algebra. Noise n is added before encryption to prevent plaintext recovery via Gaussian elimination [75]. This noise grows with data operations and can eventually corrupt the data [54, 75], but decrypting before too many operations eliminates the noise.

2.4.1 Types of Homomorphic Encryption

To deal with the noise problem, several algorithms of HE exist. They can be categorized into three types. First is Partial Homomorphic Encryption (PHE) [167], which just supports one operation type. Schemes of this category can apply a single operation for an arbitrary amount of time without losing the ability to decrypt the data. It is usually the least practical type as it can only work for specific applications that perform the same operation over and over.

Second is Somewhat Homomorphic Encryption (SHE) [140], which supports multiple operation types. Schemes of this category can apply multiple operations, but only for limited amounts of time. If the limits are exceeded, the ability to decrypt the data is lost. It is suitable for applications where the number of operations is fixed at design time, i.e., has little to no data dependency.

Third is FHE [54, 75], which supports multiple operation types with mitigation strategies to limit the noise growth. Schemes of this category can apply multiple operations in any order for an arbitrary amount of times.

2.4.2 Fully HE over the Torus (TFHE)

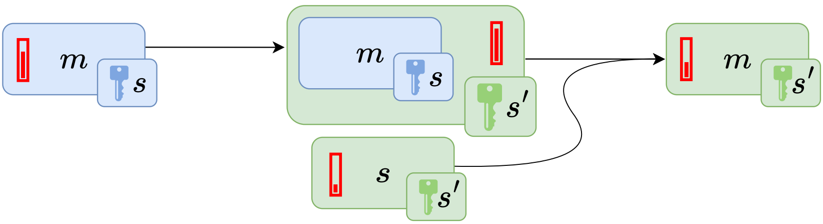


Figure 2.7: Bootstrapping operation of TFHE: The ciphertext of message m has a high noise (shown by the red bars) and is therefore re-encrypted homomorphically with a new key s' and then decrypted homomorphically using the old key s to restore the noise level.

TFHE is based on the Learning With Errors and Ring Learning With Errors problems [54]. The calculations for TFHE are done over the real Taurus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ in a quantized finite format $\mathbb{T}_q$ with $q = 2^{32}$. The numbers are represented as $\{0, \frac{2^0}{2^{32}}, \ldots, \frac{2^{32}-1}{2^{32}}\}$. $\mathbb{T}_q$ can be identified with $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ which is uniform and easier to compute. Therefore, the calculations are done over $\mathbb{Z}_q$. The polynomial ring used for TFHE is $\mathbb{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$ with $N = 2^9$ and the polynomial coefficients being modulo q.

To mitigate noise, TFHE performs bootstrapping (re-encryption with a new key then decryption using the old key) as shown in Figure 2.7. It works as follows: suppose that there is an SHE scheme that can support N operations and needs $N - m$ operations to perform decryption. Then it can perform m homomorphic operations normally. Afterward, it homomorphically encrypts the ciphertext again using a new key. Then, using the old key in homomorphically encrypted form, it decrypts the ciphertext. The result would be the ciphertext encrypted using the new key. Note that during the decryption, the noise is reduced by definition as the operations did not exceed N. Therefore, this new ciphertext now has a very low noise level and can continue m homomorphic operations.

Bootstrapping is computationally intensive, and decryption is usually complicated, therefore, doing it homomorphically is even more complicated. All FHE schemes have similar bootstrapping mechanisms. However, TFHE has an extra feature which is that it can perform computations during each bootstrapping. Therefore, bootstrapping is not a totally lost overhead and is called Programmable Bootstrapping (PBS). Compared to other schemes, this increases the efficiency of TFHE for certain tasks such as the evaluation of neural networks.

Even with TFHE performing computation during PBS it is still very computationally intensive. The main step where the highest overhead stems is the external product step. This is the step where the ciphertext $s(m)$ is re-encrypted to produce the double homomorphically encrypted ciphertext $s'(s(m))$. Therefore, this step is usually the focus of acceleration schemes. It involves, as its name suggests, multiplications.

2.5 Physical Unclonable Functions

In addition to the computation, a crucial part in cloud computing is establishing communication between clients and servers in an authenticated manner. One possible solution to perform the authentication is to use PUFs. PUFs leverage the inherent manufacturing variations of ICs to produce unique and unpredictable responses to given inputs, known as challenges. These deviations, which are impossible to clone or predict without physical access to the device, make PUFs a powerful tool for security applications, particularly in device authentication and cryptographic key generation.

PUFs can be built using several hardware primitives. Among these are electronic primitives [177], mechanical primitives [76, 189], optical primitives [134], analog primitives [133], and quantum primitives [156]. Memory can also be used to build PUFs [188]. One example is Butterfly [123], which reads the initial state of SRAM PUFs. Another example is to use the Row-Hammer effect [12] to build unique patterns as done in [169].

PUFs are generally classified into weak and strong categories. Weak PUFs generate a limited number of Challenge-Response-Pairs (CRPs) and are often used for tasks like cryptographic key storage. In contrast, strong PUFs can generate an exponential number of CRPs, making them suitable for more complex applications like device authentication [98, 168].

An example of strong PUFs is Pseudo LFSR PUF (PLPUF). It uses combinational elements instead of sequential registers in an LFSR-based design. Figure 2.8 presents a PLPUF for 32-bit challenges and responses. It has 32 elements L_i, each with an inverter connected to a multiplexer. The multiplexer selects between the initial challenge value and the previous element's output based on a select signal. The input for L_0 comes from an XOR gate with inputs mirroring a same-sized LFSR. The responses r_i are outputs of elements L_i, stored in Flip-Flops FF_i, which store the response one clock cycle post-challenge initialization using an enable signal. Besides the challenge, the stored response depends on L_i latency and clock frequency. The frequency is constant and typically known, but L_i latency depend on IC process variation.

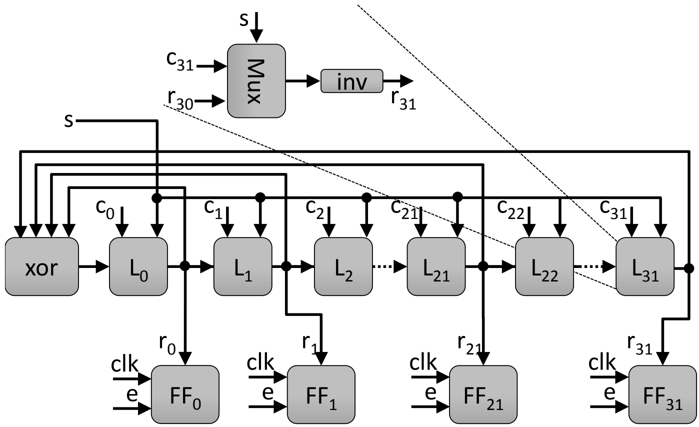


Figure 2.8: The design of PLPUF with a 32-bit challenge. The response depends on process variations in the circuit, providing unique outputs for different ICs [98].

2.5.1 PUF Quality Metrics

The reliability against noise is one of the three basic metrics used to evaluate PUFs. The other two metrics are uniformity and uniqueness. Each metric has an ideal value as follows. If the metrics of a PUF are near the ideal value, then the PUF is said to have good performance, otherwise, it has bad performance. The reliability is evaluated based on

$$\text{Reliability} = (1 - \frac{1}{N}\sum_{i=0}^{N-1}\frac{HD(R_s, R_i)}{m}) \times 100\%, \tag{2.1}$$

where N is the total number of measurements used to calculate the reliability, m is the bit-length of the response generated by the PUF, R_s is the stable response of the PUF at normal conditions, R_i is the response of the ith measurement, and HD is the Hamming distance between two responses, i.e., the number of different bits. Ideally, the reliability should be at 100%.

The uniformity metric measures the frequency of 1s in the response, i.e., its Hamming weight. The probability of 0 and 1 should be equal, i.e., ideally, the uniformity should be at 50%. Uniformity is calculated by Eq. (2.2), where $R(i)$ is i-th bit of a response binary string.

$$\text{Uniformity} = \frac{1}{m}\sum_{i=0}^{m-1} R(i) \times 100\% \tag{2.2}$$

The uniqueness metric measures how unique a PUF is compared to other PUFs. If PUF-responses are similar across different ICs, it means that the PUF design is not governed by the process variations but rather by the delay paths of the design itself. Ideal uniqueness should be at 50%. If the uniqueness is higher, this would mean that the responses are similar. A lower uniqueness would mean that the bits are also similar but with inverted values. Uniqueness is important, as an attacker might have a reference PUF at hand. If the uniqueness is bad, then the attacker can easily model the PUF based on the reference PUF. Uniqueness is calculated based on Eq. (2.3), where Z is the number of PUFs, P_i is the response of the i-th PUF, and P_j is the response of the j-th PUF.

$$\text{Uniqueness} = \frac{2}{Z(Z-1)} \sum_{i=0}^{Z-2} \sum_{j=i+1}^{Z-1} \frac{HD(P_i, P_j)}{m} \times 100\% \tag{2.3}$$

2.5.2 Machine Learning Modeling Attacks on PUFs

ML Modeling attacks significantly threaten the security of PUFs, especially those with predictable structures. Adversaries gather many CRPs to create a model that mimics the PUF, nullifying the security of PUF by generating valid responses without device access [99, 199]. The Arbiter PUF (APUF), shown in Figure 2.9, is vulnerable to ML attacks due to its linear delay model. It comprises switches controlled by challenge bits, with an arbiter determining the output based on the delay of two signals. Although APUF can generate many CRPs, the linear challenge-response relationship makes it susceptible to ML attacks like Logistic Regression (LR) and Support Vector Machine (SVM)[39].

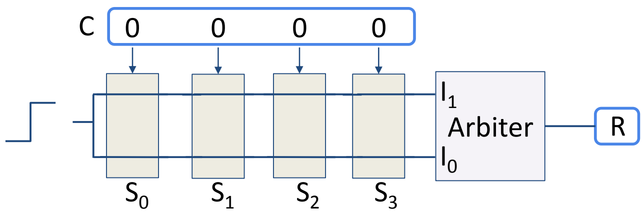


Figure 2.9: Design of APUF, which is vulnerable to ML attacks due to its linear delay model [99].

In an ML attack, the attacker first collects a large dataset of CRPs from APUF. Using this data, the attacker trains a machine learning model to predict the response for any given challenge. Because the delay model of APUF is linear, the trained model can achieve high accuracy with relatively few CRPs, making this type of attack highly efficient and effective. As a result, APUF and similar PUFs are often considered insecure against adversaries capable of performing ML modeling attacks [146, 180].

2.5.3 Machine Learning-Resilient PUFs

To counter ML modeling attacks, various strategies have been proposed to enhance the resilience of PUFs. These strategies fall into three groups. The first group uses cryptographic operations in the PUF response generation process. For example, using a hash function to encode the PUF response hides the relationship between challenge and response, making it harder for attackers to model the PUF. Other techniques include using Error Code Correction (ECC) to mask noise effects and prevent insights from repeated queries [108, 160].

The second group modifies the architecture of the PUF itself to increase its complexity and reduce its vulnerability to ML attacks. This can involve combining multiple PUFs to create a more complex response, as seen in XOR-PUFs, or introducing additional randomness into the PUF structure. For example, the CT-PUF uses a combination of Ŗ-PUF, APUF, and BiPUF to complicate the model, while the NoPUF design introduces intentional noise into the response to make it harder to predict it [146, 187].

The third group uses NVM memory with MLC capabilities, which represents a significant advancement in memory design by allowing each cell to store more than just a binary state. In NVMs like PCM and RRAM, MLC enables multiple resistance levels, encoding several bits of information per cell [19, 70]. This enhances the ML resilience of PUFs by utilizing the inherent resistance variability of NVM cells to generate unique responses. MLC increases the complexity of PUF responses, making them more resistant to ML attacks [89, 202].

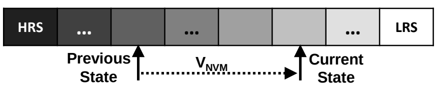


Figure 2.10: Reconfiguration of PCM cell states in an NVM-based PUF. The change in state modifies the PUF response, enhancing ML resilience [124].

As Figure 2.10 shows, in NVM-based PUFs, each memory cell can be set to one of several resistance levels, depending on the applied voltage. By varying the challenge input and reconfiguring the NVM cells, a wide range of unique responses can be generated. This dynamic behavior significantly increases the difficulty of modeling the PUF using ML techniques [124].

3 Client-Server Authentication via ML-Resilient PUFs

The first focus of the dissertation is on the communication between the client side, typically a resource-constrained device, and the server side in a cloud setup. To authenticate communication between the client and the server securely, the chapter explores lightweight and efficient PUFs. PUFs are widely investigated for security applications, such as attestation [17, 118, 172], RFID tags [41], IoT [33], electronic transaction protocols [47], secure FPGA reconfiguration [27], and secure code execution [114, 115, 149]. These applications, particularly in cloud and edge systems, often run on resource-constrained ICs, necessitating lightweight PUFs [33, 47, 136].

The use of PUFs in cryptography has introduced new attack vectors. Attackers with physical access or the ability to eavesdrop (e.g., through network interception) can collect CRPs and build ML models to predict responses [39, 71, 99, 187]. New PUF designs with cryptographic techniques counter these attacks, but often introduce resource and latency overheads [59, 108] as the cryptographic techniques are costly. Alternatively, modifications like XOR-PUF increase ML modeling complexity by combining outputs from several internal PUFs [177]. However, this also adds overhead by using several PUFs in parallel. A common PUF primitive is memory, particularly NVM [117, 169]. NVM technologies like PCM, RRAM, Spin Orbit Torque RAM (SOTRAM), and Spint Transfer Torque RAM (STTRAM) switch states without constant power [150]. Due to the non-linear relationship of MLC, NVM-based PUFs are resilient to ML-based attacks [89, 202] but often are weak PUFs [1]. This chapter proposes two novel PUF designs: Cascaded PUF (CaPUF), an ML-resilient lightweight PUF, and Arbiter NVM-based PUF (ANV-PUF), a strong NVM-based PUF with robust ML resistance while remaining lightweight.

This chapter is based on contributions from [1–3].

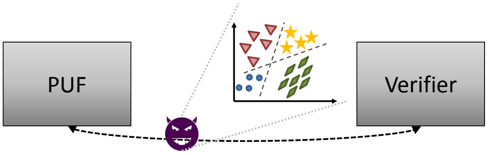


Figure 3.1: ML modeling attacks on PUFs, by listening on the communication channel between PUF and the verifier, the attacker is capable of modeling the behavior of PUFs and predicting the responses for unseen challenges. Consequently, the authentication can be compromised.

3.1 Motivational Example

As a motivational example, consider the case of authentication using PUFs, a *verification* entity sends an input *challenge* to a PUF and collects its output *response*. If an attacker observes the PUF and collects challenges with their corresponding responses to build and train a machine learning (ML) model, they can accurately predict the responses of this PUF for unseen challenges. Such an attack, as shown in Figure 3.1, is usually based on listening to the communication channel between the verifier and the PUF. To mitigate such attacks, one solution is to use a cryptographic hardware module such as Hash to randomize the output. However, this comes with a significant overhead area, which is undesirable for PUFs [2, 160].

3.2 Threat Model

The target scenario in this chapter is one in which a PUF is used to authenticate the identity of a prover (client) to a verifier (server of the CSP) to stop DDoS attacks in a cloud setup. This is a common PUF application [33, 160]. The attacker's target is to impersonate the PUF. Therefore, the PUF has to be strong in order to avoid replay attacks. Replay attacks occur when challenges sent to the PUF are repeated. However, with a strong PUF, repeating challenges is less likely [2].

The aim is to combat the threat of ML-modeling attacks on PUFs as well. In such a case, the attacker would be able to eavesdrop on the communication channel between the PUF and the verifier to send and receive CRPs. The attacker may even have the device in their possession for some time, impersonating a verifier and collecting CRPs.

Based on the CRPs collected, the attacker would be able to train an ML model to predict the response to unseen challenges. Consequently, the attacker would be able to impersonate the device containing PUF and act as an authenticated device.

LR and SVM, Neural Networks (NNs), and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) are typically used for ML-modeling attacks [39, 190]. Those ML techniques, when trained with tens of thousands of CRPs, are capable of accurately predicting the response to unseen challenges. The attacks rely on the linear relation between challenges and responses in most of PUF designs [39].

3.3 Contributions

The contributions of this chapter are:

- CaPUF, a lightweight silicon-based PUF dynamically changing its response behavior to achieve full resilience against ML modeling.
- ANV-PUF, the first strong PUF that uses the iterative pulsing property of NVM PUFs to achieve full resilience against ML modeling.
- Studying the effects of implementing PUFs as run-time accelerators on FPGAs on the performance of the PUFs.
- Studying the endurance degradation of NVM-based PUFs and its effect on the reliability of the PUFs.

3.4 Previous ML-Resilient PUFs

With progressing ML attacks, creating ML-resilient PUFs is crucial for secure hardware. The following is a review of efforts to design ML-resilient PUFs, focusing on two widely used technologies for PUFs,silicon-based and NVM-based PUFs, to understand their strengths, weaknesses, and challenges in achieving robust ML resistance with practical design overhead.

3.4.1 Silicon-based

The earliest ML-resilient design is the XORPUF [177], where the output of several PUFs is xored, creating a more complex model. However, it is not fully secure as it resembles a combination of linear models [39]. With fewer xored PUFs, one might dominate the output, making attacks easier [146, 180]. Thus, a high number of PUFs is needed, leading to significant overhead. The CT PUF [199] combines three PUF designs. Depending on the count of 1*s* in even and odd challenge positions, the response comes from an RO-PUF, APUF, or BiPUF, making it difficult to model the PUF. CT PUF also introduces an optional security layer by xoring the output of two PUFs, similar to XORPUF but with different designs. NoPUF [187] introduces obfuscation by hiding a reliable PUF within a noisy one. It is based on the APUF design and is fine-tuned to be reliable only for specific challenges. If the challenge is outside this subset, the output is noisy, complicating PUF modeling due to unreliable responses. An attacker with NoPUF design knowledge can model it using reliability information [39, 99, 146].

3.4.2 NVM-Based

The first NVM PUF was proposed in [124], using PCM-based NVM where each cell is in an arbitrary intermediate state. The challenge is an address to the memory; the selected cell is compared to a reference. If the resistivity is higher than the reference, the output bit is 1; otherwise, it is 0. Occasionally, a short reconfiguration message is sent with the challenge, converted to an analog voltage V_{NVM} using a Digital to Analog Converter (DAC), applied to all memory cells to change their states (Figure 2.10). Reading the same address twice, before and after reconfiguration, gives different responses, making prediction difficult if reconfiguration is periodic. During enrollment, CRPs are collected under different reconfiguration messages. The verifier, tracking reconfiguration messages, knows the expected response to authenticate the device. This idea was extended to STTRAM, SOTRAM, and RRAM [30, 48, 112, 203]. The first improvement over [124] was made by [200], noting that logarithmic resistance changes in PUF reduce uniqueness. They used a logarithmic amplifier, increasing the uniqueness from 60.56% to 48.14%. Employing ECC maintained reliability at 99% instead of dropping to 90%. The Reed-Muller ECC requires minor overhead for storing helper data on the NVM chip. In contrast, [201] used an Imprecise Control Regulator (ICR) and

Hash function to boost uniqueness. ICR causes unpredicted offsets, increasing randomness. The hash function further randomizes the output, improving uniqueness from about 30% to 50%, with some area overhead.

The ideas in [124, 200, 201] focus on weak PUFs. Unique responses correlate linearly with memory addresses. To increase response range, [202] uses MLC property of NVM with a varying current reference as a challenge, converted to an analog signal by a DAC. Starting at HRS, the NVM cell's resistance changes iteratively using short pulses. After each pulse, cell current is compared to the analog signal; if lower, the counter increases; otherwise, counter value is returned as the response. The counter resets after reading.[89] follows [202], using RRAM instead of PCM to enhance ML resilience by xoring outputs of multiple cells. The responses were ML-resilient with 50% uniqueness. Despite using a counter, the response range is limited to 4-bit pulses.[30] expands the range by employing Arbiter design from APUF with RRAM, switching from LRS to HRS based on challenge bits, affecting delay due to different capacitance behaviors. However, this PUF is only 65% secure. To summarize, NVM-based ML-resilient PUFs are either weak as in [89, 200, 201] or not fully secure as in [30]. There is a need for a strong NVM-based ML-resilient PUF.

3.5 Proposed ML-Resilient PUFs

To address the limitations of existing ML-resilient PUFs, this chapter introduces two novel designs: CaPUF and ANV-PUF. They aim to improve PUF security with complex dynamic behaviors that are difficult for ML models to predict while being strong PUFs.

3.5.1 CaPUF Design

The first ML-resilient PUF in this chapter is CaPUF which is silicon-based. The main idea is to use several PUFs as building blocks to have a dynamically changing behavior from the PUF. With each new challenge, some of the properties of CaPUF change as if the challenge was given to a different PUF than with the previous one. CaPUF achieves the dynamic behavior through two aspects. The first aspect is the cascaded architecture. There are four stages of PUFs cascaded one after the other, the output of one stage is the input of the next stage. Hence, the response comes from an internal challenge that is different from the one sent initially. The PUFs used for the cascade are PLPUFs. They are used because they are very lightweight PUFs.

The second aspect is the frequency of collecting the outputs from PLPUF. Instead of using a constant frequency all the time for all PLPUFs, for each stage, the frequency of collecting each individual response bit is variable. It depends on the output of another PLPUF from the previous stage. This introduces an extra layer of protection, as for each challenge the frequency of collecting each bit is different and the model of the PLPUF changes dynamically.

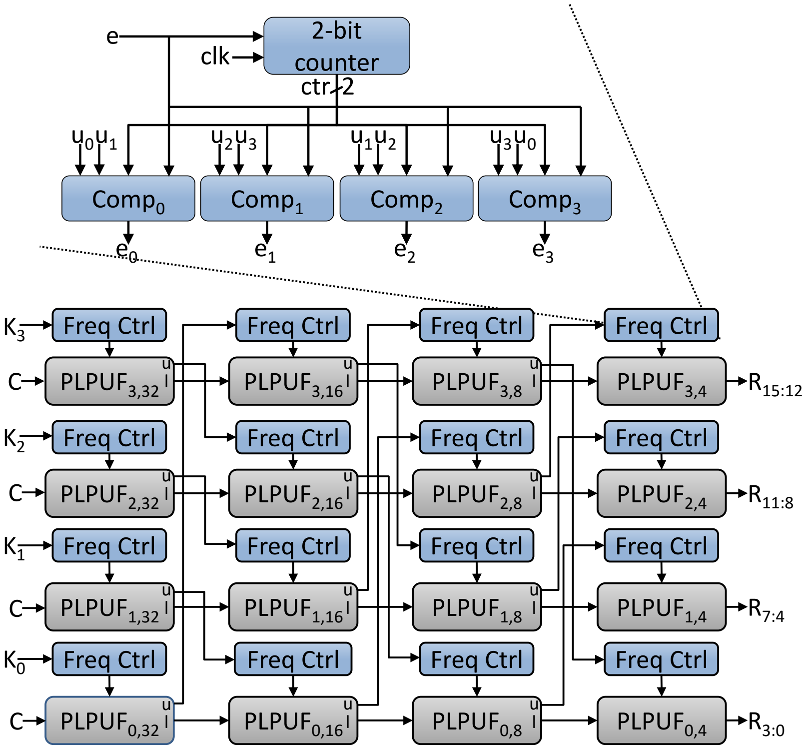


Figure 3.2: The Novel Cascaded PUF (CaPUF) containing 16 PLPUFs in a 4×4 grid. C is the 32 bit challenge sent to CaPUF, R is the 16 bit response (each 4 bits come from one PLPUF), and K_0 until K_3 are four secret keys.

Figure 3.2 shows the design of CaPUF. It has four stages (shown as columns in Figure 3.2) and each stage contains four PLPUFs. The challenges and responses of the PLPUFs of the first stage are 32 bit long (the same as the design in Figure 2.8), and in the second/third/fourth stage 16/8/4 bit, respectively. CaPUF has four chains (shown as rows in Figure 3.2) from $chain_0$ (bottom) to $chain_3$ (top). The output of one PLPUF in the chain is connected as an input to the next PLPUF in the chain. The 32 bit challenge is fed to all four PLPUFs of the first stage, and the outputs of each PLPUF of the fourth stage are concatenated to form the 16 bit response of CaPUF. Each stage produces double the amount of bits needed as input for the next stage. The bits are separated into two signals, upper and lower, for all stages except the final one. The lower part is used as an input for the next stage.

The upper part of the output is used to control the frequencies of the PLPUFs from the next stage. It does not control the same PLPUF that takes the lower part as input challenge. This choice is made so that there is no dependency between the PLPUF input and its frequency controller. The same chain controls the frequency of another chain only once, to break any possible dependency between the chains. The frequency control of the first stage must be initialized with a secret key (see K_i values in Figure 3.2). CaPUF uses a Physically Obfuscated Key (POK) to generate the K_i values. The POK can be any PUF with high reliability. For the case of CaPUF, it uses a PLPUF with an arbitrary constant challenge to keep it lightweight.

Figure 3.2 shows the details of the frequency controller of the last stage, the other controllers look the same but with more comparators as they have more bits. The PLPUF is enabled for four clock cycles. A counter is used to track the clock cycles. Two bits of its input (the upper signal from a PLPUF of the previous stage) control the frequency of collecting one response bit. If the value of the two bits is equal to the counter, the enable signal for the Flip-Flop is turned on to collect the response bit. It is collected after 1, 2, 3, or 4 clock cycles. The control is done by a simple 2 bit comparator. This makes each bit independent from the other.

Compared to the 32×16 APUF normally used to get the 16 bit response from a 32 bit challenge, CaPUF uses significantly less hardware and have ML-resilience. Therefore, even when CaPUF uses 16 PLPUFs, it is still lightweight.

3.5.2 ANV-PUF Design

The second PUF design in this chapter, ANV-PUF, is NVM-based. ANV-PUF's design is similar to an APUF, with 128 blocks, each receiving one challenge bit. However, instead of having delay propagated on switches, ANV-PUF relies on the MLC property. To profit from the MLC property, ANV-PUF uses an extra 8-bit value in conjunction with the 128-bit challenge. The 8-bit value specifies the voltage level that the NVM cell must reach by changing its internal resistance. The change in voltage is related to the change in resistance, which is not linear. Thus, the modeling of the PUF will be more difficult than that of a normal APUF.

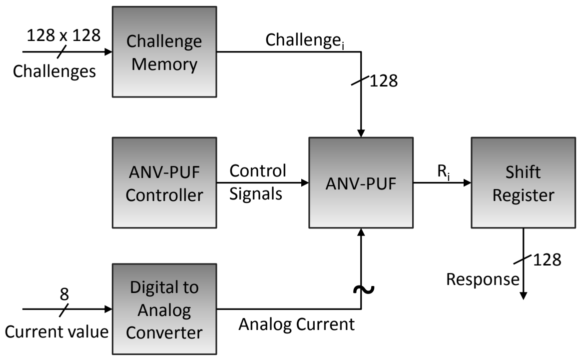
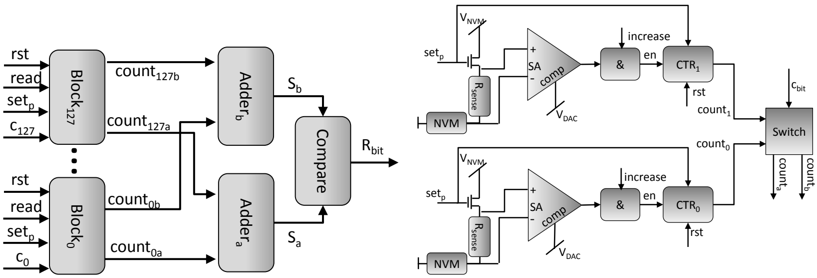


Figure 3.3: ANV-PUF System, challenges given sequentially to ANV-PUF to build a full bitwidth Response. Digital to Analog Converter used to generate a variable reference voltage V_{DAC}.

Similar to all APUF designs, ANV-PUF produces one unique response bit for one full 128-bit challenge. To generate a 128-bit response, 128 ANV-PUFs would have to be used in parallel. Another alternative is to give 128 challenges sequentially to a single ANV-PUF and collect the response. The sequential alternative is chosen because of it needs $\frac{1}{128}$ resources compared to the parallel one. For the sequential alternative, 2^7 different challenges are needed to obtain one 128-bit response, the number of possible different challenges reduces from 2^{128} (one challenge is given to 2^7 parallel PUFs) to 2^{121} (2^7 different challenges are given sequentially to one PUF), which is still a large enough number of challenges.

Alongside ANV-PUF itself, additional logic is needed to control the generation of the full response. Figure 3.3 shows the implementation of the complete system for using ANV-PUF. The 128 challenges are received and stored in a challenge memory. To convert the extra value used to generate the voltage reference, a DAC is used. For generating the full 128-bit response, ANV-PUF uses the same reference value. Finally, a controller is needed that supplies reset signals, writes, and reads pulses to the ANV-PUF. The control signals, the analog voltage, and the 128 challenges are given as input to ANV-PUF. The response is collected sequentially in a shift register to output the 128-bit full response.



(a) ANV-PUF Structure, 128 blocks are used to produce count$_a$ and count$_b$. Based on which sum is higher, the output bit is either ‘0’ or ‘1’

(b) One block of ANV-PUF: Two NVM cells are used to produce the count$_0$ and the count$_1$, respectively. Based on the challenge value, count$_a$ and count$_b$ are each assigned one of them.

Figure 3.4: Design of ANV-PUF

ANV-PUF itself consists of 128 blocks that implement the NVM cells, two adders (Adder$_a$ and Adder$_b$), and a comparator as shown in Figure 3.4a. Each block i receives a one-bit challenge c_i, the control signals *rst* and *read*, and sequential *set* pulses set$_p$ to produce two count values, count$_{ia}$ and count$_{ib}$. Both adders perform the addition over all 128 counts having the same respective subscript to calculate S_a and S_b. If S_a is higher, then the output of the comparator is ‘0’, otherwise it is ‘1’.

To produce both counts, all blocks include two NVM cells, as Figure 3.4b shows. Each NVM cell is connected to a transistor acting as a switch and a small shunt resistor (R_{sense}) to provide input to a Sense Amplifier (SA) with comparator functionality. If set$_p$ is high, the NVM cell is supplied by V_{NVM}, which is a variable voltage reference and supplies either read, set, or reset voltage level. When V_{NVM} is at read level, the voltage passing through R_{sense} is compared to the reference voltage of the DAC (V_{DAC}) using the SA. If the voltage through R_{sense} is lower, the counter increases. Once the voltage of both R_{sense} is higher than V_{DAC} both counts pass through the switch. The switch is controlled by the challenge bit; if the challenge bit is '0' $count_a$ is assigned $count_0$ and $count_b$ is assigned $count_1$. If the challenge bit is '1' $count_a$ is assigned $count_1$ and $count_b$ is assigned $count_0$. Hence, with each unique 128-bit challenge, a unique set of 128 counter values are summed by $Adder_a$, and another unique set of 128 counter values are summed by $Adder_b$. Consequently, this leads to a unique comparison between S_a and S_b for each unique 128-bit challenge.

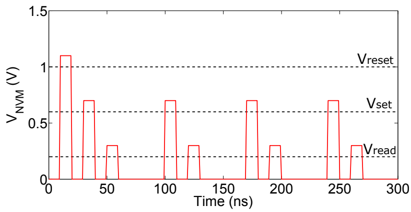


Figure 3.5: Time Plot of ANV-PUF Operation based on PCM parameters for pulse width and amplitude. One reset signal is followed by several set and read signals to perform the counting.

To operate the full system, the ANV-PUF controller has to generate various signals. The most important signal is the V_{NVM} power supply signal as it controls the iterative increase of counter values. It has to change between the three voltage levels for *set*, *read*, and *reset*. Figure 3.5 shows how the signal works for ANV-PUF when PCM is used as the NVM cell. To generate one count, first, a 'reset' pulse is generated to bring the PCM cell to the fully amorphous state. This is followed by a short 'set' pulse to go into a gradual state between amorphous and crystalline. Consequently, a 'read' pulse is generated to evaluate whether the V_{DAC} is matched or not. This pattern of a short 'set' pulse and then a 'read' pulse is iterated until the desired level is reached. The same sequence with one 'reset' signal followed by iterations of short 'set' and 'read' pulses is used as well for RRAM when used as the NVM cell. However, the amplitudes and widths of the pulses are changed to the levels needed for RRAM.

In addition to V_{NVM}, the controller generates several other signals. It generates the 'set pulse' (set_p) that controls the switch behavior of the transistor and

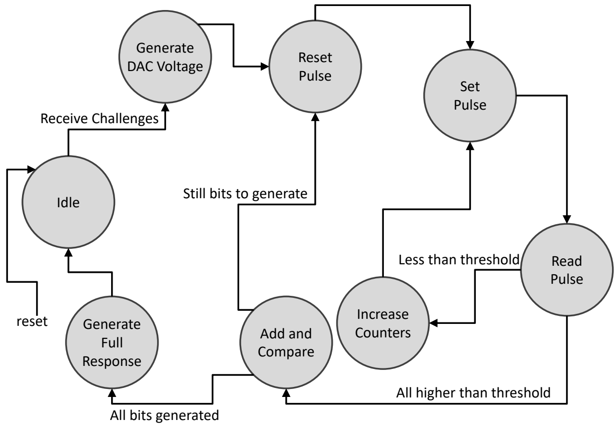


Figure 3.6: Operation of ANV-PUF controller: Through the finite state machine, the controller is able to collect the response bits sequentially from all 128 challenges.

acts as a clock signal for the counter. It also generates the 'increase' signal that enables the counter and the 'reset' signal that resets the counter after the result of a challenge is collected. Moreover, it controls the collection of the response bit by bit in the shift register.

Figure 3.6 shows the operation of the ANV-PUF controller. Initially, it is idle until the challenges are received. Then it generates the voltage using the DAC based on the 8-bit value received with the challenges. To ensure that ANV-PUF starts from a full high resistivity state (HRS), a reset pulse is given to all the NVM cells. Set pulses are then repeatedly generated, followed by read pulses and an increase of the counters. Once a cell can match the voltage, the SA stops sending the enable signal to its counter. If all cells reach the reference level, the controller switches to the next state to perform the additions and then compares S_a and S_b to produce the response bit. The whole process is then repeated. It starts by resetting all cells to HRS and resetting counters to zero, then generating the next response bits. Once all 128 bits are generated, the full response is generated and delivered, and the controller returns to the idle state.

3.5.3 Complexity of Modeling the novel PUFs

ML models can predict the output of PUFs based on the simplicity of their delay models [2]. To further understand why it would be hard to model the novel PUFs in the scope of a successful attack, the following sections show an approximate mathematical model for their delay. The goal is to show that the mathematical model is more complex than the other PUFs related to them, namely APUF and PLPUF, and thus it will be harder to build an accurate model that predicts responses correctly. Note that it is not an exact model, but an approximated model for simplicity. The exact model would be significantly more complicated and even harder to be built through ML. Note that the mathematical model is not used exclusively, but also state-of-the-art attacks are run against CaPUF and ANV-PUF.

3.5.3.1 Mathematical Model of PLPUF

The model for PLPUF is based on the design from Figure 2.8 with a bit width of 32 bits. The assumption is that all its L_i elements and the XOR-gate have the same delay D, which makes this an approximate model, as in practice the different L_i elements will have slightly different delays. Based on the period T of the clock signal ('clk' in Figure 2.8), each bit c_i of a new input challenge will traverse through a constant number n of L_i elements.

$$n = \frac{T}{D}. \tag{3.1}$$

This means that PLPUF will calculate n responses (all 32 r_i signals in Figure 2.8 correspond to one response) for every input challenge c, but only the n^{th} response will be sampled by the output register. In the following, r^j will be used to differentiate the n responses, and r_i^j will be used to denote bit i of the j^{th} response. The first response r^0 corresponds to the inversion of the challenge. All the following responses r^j can be calculated as shown in Eq. (3.2). The first bit r_0^j of the j^{th} response comes from the xor gate, and the other bits come from the previous bit position of the $j-1^{\text{th}}$ response.

$$
\begin{aligned}
r_i^0 &= \overline{c_i} \quad i \in [0, 31] \\
r_0^j &= \overline{r_0^{j-1} \oplus r_1^{j-1} \oplus r_{21}^{j-1} \oplus r_{31}^{j-1}} \quad j \in [1, n-1] \\
r_i^j &= \overline{r_{i-1}^{j-1}} \quad i \in [1, 31], j \in [1, n-1]
\end{aligned}
\tag{3.2}
$$

With enough CRPs, the model can deduce n and the number of transformations that occur through the PLPUF. The final model of a PLPUF that creates a response r can be expressed as an arbitrary function f based on n and the challenge c,

$$r = f(n, c). \tag{3.3}$$

f calculates the above-described shift and xor operations as shown in Eq. (3.2). This is a simplified version of the model. The real model would be a function of the delays of all L_i elements and the xor gate, which would all have different values. However, it is still a simple additive model that can be modeled by an ML-based attack after enough CRPs are collected.

3.5.3.2 Mathematical Model of CaPUF

Based on the approximate model for PLPUF, the approximate model of CaPUF is built. For illustration, the explanation focuses only on chain_0 from Figure 3.2, but the same concept holds for all chains. The explanation goes step by step from the first stage to the final stage. Thus, showing how each stage in combination with the frequency controllers contributes to the complexity of the model, which in turn makes CaPUF harder to model. The explanation uses the same subscripts as Figure 3.2 which are two numbers separated by a comma. The first number is the chain identifier, and the second number is the bit width used for the challenges and responses of the respective PLPUF on the chain. Starting by the first stage, the output from $\text{PLPUF}_{0,32}$ will be $r_{0,32} = f(n_{0,32}, c)$. $n_{0,32}$ can be computed as

$$n_{0,32} = \frac{K_{0,32} T}{D_{0,32}}, \tag{3.4}$$

where $K_{0,32}$ comes from the POK that stores the initial value for the first stage PUF, $D_{0,32}$ is the delay of one element in $\text{PLPUF}_{0,32}$, and T is the clock period. Here, the first effect of the design can be noticed. The response is collected after multiple clock cycles based on the secret value $K_{0,32}$. However, this alone would not be sufficient to prevent ML-based attacks, as $K_{0,32}$ is constant and the model will be able to learn it. That is why the other stages are added. The final output $r_{0,32}$ for the initial challenge c is

$$r_{0,32} = f(\frac{K_{0,32}T}{D_{0,32}}, c). \tag{3.5}$$

Going to the next stage in the chain, the input challenge for $\text{PLPUF}_{0,16}$ is the response from the previous stage $c_{0,16} = r_{0,32}$. $n_{0,16}$ is calculated as

$$n_{0,16} = \frac{r_{1,32}T}{D_{0,16}}, \tag{3.6}$$

where $r_{1,32}$ is the response from $\text{PLPUF}_{1,32}$ from Figure 3.2. $r_{1,32}$ is variable and changes with each new input challenge. Hence, $n_{0,16}$ also changes with each challenge, which achieves the dynamic behavior that is desired for the design.

By substituting the values of $c_{0,16}$ and $n_{0,16}$ in Eq. (3.3), the final output from the PUF can be expressed as

$$r_{0,16} = f(\frac{f(n_{1,32}, c_{1,32})T}{D_{0,16}}, f(n_{0,32}, c)). \tag{3.7}$$

Similarly, going to the third stage of the chain, the challenge $c_{0,8}$ is the output of Eq. (3.7). The value $n_{0,8}$ is evaluated as

$$n_{0,8} = \frac{r_{2,16}T}{D_{0,8}}. \tag{3.8}$$

$r_{2,16}$ is the response of $\text{PLPUF}_{2,16}$. Hence, $n_{0,8}$ is variable and gets a different value based on each new challenge.

By substituting $c_{0,8}$ and $n_{0,8}$ in Eq. (3.3) the response $r_{0,8}$ from this stage is based on

$$r_{0,8} = f(\frac{f(n_{2,16}, c_{2,16})T}{D_{0,8}}, f(\frac{f(n_{1,32}, c_{1,32})T}{D_{0,16}}, f(n_{0,32}, c))). \tag{3.9}$$

At the final stage, $n_{0,4}$ depends on $r_{3,8}$ and is formulated as

$$n_{0,4} = \frac{r_{3,8}T}{D_{0,4}}. \tag{3.10}$$

$r_{3,8}$ is based on a complex path similar to $r_{0,8}$. This makes $n_{0,4}$ variable and unpredictable as each PUF in the path of $r_{3,8}$ will affect its value. The final output from this chain is then based on

$$r_{0,4} = f(\frac{r_{3,8}T}{D_{0,4}}, f(\frac{r_{2,16}T}{D_{0,8}}, f(\frac{f(n_{1,32}, c_{1,32})T}{D_{0,16}}, f(n_{0,32}, c)))). \tag{3.11}$$

It can be seen that the output is based on several outputs of PUFs from the cascade. The relation between the different PUFs is not a simple additive one but more complicated with multiplications.

To model the full CaPUF, the attacker would need to model several different PUFs, where their behaviors affect each other in multiplicative ways and transformations of inputs to outputs. Note that Eq. (3.9) and (3.11) are not fully expanded for brevity. The underlying model is even more complex. Moreover, this is just an approximate model. The real model will have a more complex structure. For once, the delay of all elements within one PUF is not equal. Thus, the calculation of n, the intermediate responses r^j, and consequently r will be more complex. Modeling all these PUFs and their relations will require an extremely high number of CRPs which might not be feasible to collect.

3.5.3.3 Mathematical Model of Arbiter PUF

Next, the vulnerability ofAPUF to ML-modeling attacks is explained. APUFs can generally be modeled using a linear additive model [39, 180, 190]. Looking back at Figure 2.9, the output bit is based on which path has a longer delay. This can be modeled mathematically as a sign equation using Eq. (3.12), where R_{APUF} is the output response bit, and T_{D_1} and T_{D_0} are the delays of the upper and lower paths, respectively.

$$R_{\text{APUF}} = \text{sign}(T_{D_1} - T_{D_0}) \tag{3.12}$$

The delay of one path itself is the sum of the delay of all the stages that build APUF. The focus is on T_{D_1} for simplicity, but the same holds for T_{D_0}. The delay can be represented as shown in Eq. (3.13), where n is the number of stages and $w_{i_1}(c_i)$ is the delay of the i-th stage of the arbiter path as a function of c_i, which is the challenge bit controlling the i-th stage of the arbiter.

$$T_{D_1} = \sum_{i=0}^{n-1} w_{i_1}(c_i) \tag{3.13}$$

$w_i(c_i)$ is calculated following Eq. (3.14), where δ_i^{cross} is the delay of the stage when the path through the stage is crossed and $\delta_{i_1}^{\text{direct}}$ is the delay of the stage when the path through the stage is direct.

$$\begin{aligned} w_i(c_i) &= \delta_{i_1}^{\text{cross}}, \text{ if } c_i = 1 \\ w_i(c_i) &= \delta_{i_1}^{\text{direct}}, \text{ if } c_i = 0 \end{aligned} \tag{3.14}$$

Substituting Eq. (3.13) and (3.14) in Eq. (3.12) results in the final model in Eq. (3.15), where $\delta_i^{(c_i)}$ denotes the dependency between which the value δ is used based on c_i.

$$R_{\text{APUF}} = \text{sign}(\sum_{i=0}^{n-1} \delta_{i_1}^{(c_i)} - \sum_{i=0}^{n-1} \delta_{i_0}^{(c_i)}) \tag{3.15}$$

Hence, the model at the end is a linear additive model, where some constants are summed. An attacker would only need to be capable of recording enough CRPs for the model to be able to evaluate all the different possible $\delta_i^{(c_i)}$ values. For an APUF with 128 stages, it will need to be able to calculate 512 values, which usually takes around 10K-50K CRP for the model to evaluate it [190].

3.5.3.4 Mathematical Model of ANV-PUF

Compared to APUF, ANV-PUF adds an additional layer of security using V_{DAC} that leads to a minimal but significant change. Previous works, e.g., [147, 180, 190] show that a minimal change of adding a random shift or substituting a single bit of the challenge would increase the complexity of modeling the PUF quadratically. This makes the task of building the model practically impossible, as the attacker has to collect an enormous number of CRPs, which might not be feasible. To showcase the change that is caused by adding V_{DAC}, the mathematical model of ANV-PUF is built. Starting with a very similar model to APUF, the response bit $R_{\text{ANV-PUF}}$ is calculated as Eq. (3.16), where S_b and S_a are both sum values of Figure 3.4a.

$$R_{\text{ANV-PUF}} = \text{sign}(S_b - S_a) \tag{3.16}$$

Both S_a and S_b are the result of the summation of the different counters $count_{ia}$ from Figure 3.4a and follow Eq. (3.17). The focus is on S_a for simplicity, but the same holds for S_b.

$$S_a = \sum_{i=0}^{n-1} count_{ia}(c_i) \tag{3.17}$$

$count_{ia}(c_i)$ switches based on the value of c_i between $count_0$ and $count_1$ (see Figure 3.4b) and follows Eq. (3.18).

$$\begin{aligned} count_{ia}(c_i) &= count_{i1}, \text{ if } c_i = 1 \\ count_{ia}(c_i) &= count_{i0}, \text{ if } c_i = 0 \end{aligned} \tag{3.18}$$

Both $count_0$ and $count_1$ are functions of $log(V_{\text{DAC}})$ [200] with V_{DAC} itself being a variable and not a constant. Substituting Eq. (3.17) in Eq. (3.16) results in Eq. (3.19)

$$R_{\text{ANV-PUF}} = \text{sign}\left(\sum_{i=0}^{n-1} count_{ia}(c_i, log(V_{\text{DAC}})) - \sum_{i=0}^{n-1} count_{ib}(c_i, log(V_{\text{DAC}}))\right) \tag{3.19}$$

By comparing Eq. (3.19) and Eq. (3.15), the attacker must no longer try to capture constant delay values δ, but rather capture a variable value which is more challenging. The model will have to decode the logarithmic relationship between V_{DAC} and $count_0/count_1$ for each NVM cell used in each of the 128 blocks.

3.5.4 PUFs Implementation and Simulation

3.5.4.1 CaPUF Implementation on FPGA

After designing CaPUF, it is tested on real hardware. First, CaPUF was implemented as synthesizable VHDL code. The target is the Xilinx VC707 board containing a Virtex-7 FPGA. Moreover, to ensure that the PUF is not affected by modifications to the experimental setup framework, it was placed in a Pblock (definition of a sub-area on an FPGA) with constraints that manually place its elements on selected resources.

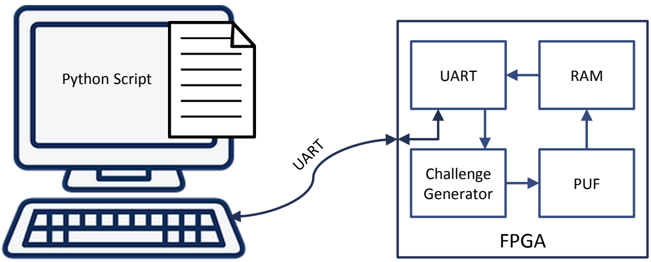


Figure 3.7: The experimental setup for logging the challenge response pairs

Four different VC707 boards are used to test CaPUF. Figure 3.7 shows the entire framework for conducting the experiments. In addition to the PUF, other components are needed to be able to log the results and communicate them to a computer for further analysis. A challenge generator is used to send the challenges to the PUF and collect a large set of CRPs. The challenges should be random and not in sequence to cover as many possibilities as possible. An LFSR with a constant seed generated randomly is used as a Pseudo Random Number Generator (PRNG) to generate the challenges. The responses along the challenges are stored on the FPGA in block RAM. For each PUF implemented on one of the four boards, 65,536 CRPs are generated. Each CRP is generated 1000 times to evaluate the reliability. The CRPs are communicated via UART to the computer to perform the analysis.

In addition to the CaPUF, PLPUF [98], APUF, 3-XORPUF, and 5-XORPUF [177] are implemented. Both XORPUFs are based on the implemented APUF. The difference between them is the number of used PUFs for the xor (3 or 5). This comes with a trade-off between security and resource usage. In addition, the performance of PLPUF is recorded on the four boards. This is because CaPUF is compared to PLPUF, which is used as its building block to see if any of the metrics (uniqueness, reliability, and uniformity) degrades. APUF, 3-XORPUF, and 5-XORPUF were only evaluated on one of the boards, since they are only used to assess whether attacks work or not and this can be measured based on one board.

3.5.4.2 Simulating ANV-PUF

There was no possibility of taping out an NVM IC for ANV-PUF. Moreover, commercial NVM ICs do not allow exploiting the MLC property. Therefore, a Matlab simulation environment is built to evaluate ANV-PUF. For the target NVM cell, both RRAM and PCM are used. They are differentiated by calling the RRAM-based PUF ANV-PUF$_{\mathrm{RRAM}}$ and the PCM-based ANV-PUF$_{\mathrm{PCM}}$. The matlab simulation environment solves differential equations to get the behavior of the NVM cells either PCM or RRAM. It use the mathematical model parameters shown in Table 3.1 from [46, 151] for PCM and from [52, 196] for RRAM. The exact mathematical model for PCM is taken from [201] and the mathematical model for RRAM is taken from [145]. The material simulated for PCM is GST and for RRAM it is HfO_X. ρ_L and ρ_H of PCM are higher than those of RRAM. HfO_X is a unipolar RRAM and, therefore, all voltages are in the positive range. V_{read} for RRAM is in the middle between the set voltage and the reset voltage, unlike for PCM, which has V_{read} of a very low value.

Table 3.1: System parameters used in Matlab simulation. PCM parameters are based on [46, 151]. For RRAM the parameters are based on [52, 196]

NVM	$\rho_{L\Omega/m}$	$\rho_{H\Omega/m}$	L/W	V_{read}	V_{set}	V_{reset}	t
PCM	416μ	100	5.7	0.1 V	0.6 V	1 V	10 ns
RRAM	40μ	27	0.2	1.2 V	1.8 V	0.8 V	5 ns

The NVM cells and the circuits are not always identical to the ideal design. To reflect this in the simulation, several sources of offset and noise are identified. The offsets are the main source of the PUF behavior, while the noise degrades the reliability. The following offset sources were identified: (i) length and width of the active region of the NVM cell, (ii) amplitude and width of V_{NVM} pulses, (iii) ρ_H and ρ_L of the active region, (iv) output from the DAC, and (v) input and output offsets for the SA. The offsets are in the range of 1%

of their ideal values and are randomly selected for each device. Two ANV-PUF_{RRAM} and two ANV-PUF_{PCM} are built, each PUF has random offsets that help to evaluate the uniqueness of ANV-PUF for both technologies.

For noise, three main sources are considered, (i) the ambient temperature, (ii) pulse amplitude of V_{NVM}, and (iii) pulse width of V_{NVM}. The three noise sources cause changes up to 25% of their ideal value randomly for each device in different runs.

To make the noise evaluation more realistic, in addition to the mathematical evaluation, a SPICE noise simulation is performed. The SPICE models for PCM and RRAM are from [72, 90] respectively. The DAC and SA are openly available part models from Analog Devices. Part **LTC2688** is the DAC and part **LT6118** is the SA with comparator functionality. The noise contribution until the output of the SA of each component of one block of ANV-PUF is evaluated in the SPICE simulation.

Based on the Matlab simulation environment, 1,048,576 response bits from each PUF are collected in the form of 8192 responses, each with a bit width of 128 bits. Each 128-bit response uses unique 128 challenges, each composed of 128 bits and one random 8-bit value to generate the reference voltage V_{DAC}. The same experiment is repeated 1000 times under random changes to calculate confidence intervals and standard deviation for all the calculated metrics.

3.6 Evaluation of the novel PUFs

3.6.1 CaPUF Quality and Performance

The quality and performance of CaPUF are evaluated on the collected CRPs. This is an important step before trying to evaluate its performance against ML-modeling attacks. If it has a bad uniqueness, then an attacker can model it based on a reference PUF. If it has a biased uniformity, then an attacker can predict large portions of the response. If it has low reliability, then it is noise-dominated and cannot be used to authenticate a device.

For uniqueness, Figure 3.8 shows in orange the results for calculating the Hamming distance between CaPUFs on the different boards. It has a Gaussian distribution, which is expected for such a random relation. It has a peak at

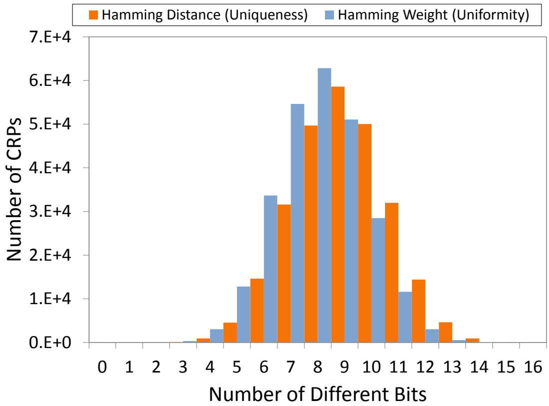


Figure 3.8: Hamming weight representing uniformity and Hamming distance representing uniqueness based on all the values collected from the FPGA boards showing the desired normal distribution behavior of CaPUF

9, which is only 1 bit away from the ideal 8. This is an acceptable deviation, as it is still close to 50%, which would be desirable. Overall, calculating the average on this Hamming distance was 8.9 and the standard deviation was 1.76.

As for uniformity, Figure 3.8 shows in blue the Hamming weight of the responses from the four boards. Here the distribution the expected Gaussian shape. In addition, the peak is exactly at 8 which is the ideal value. This means that for uniformity, CaPUF has an almost ideal performance. Overall, it has 8.01 as an average and a standard deviation of 1.63.

Table 3.2: Performance of PLPUF and CaPUF

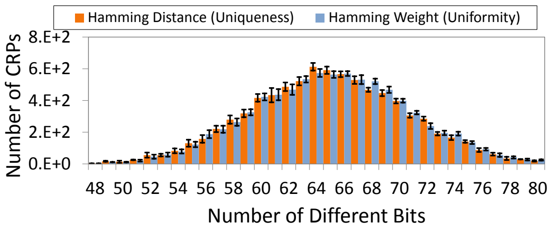
PUF	Uniqueness	Uniformity	Reliability
PLPUF	49.60%	46.12%	98.47%
CaPUF	55.63%	50.06%	92.54%

Moreover, Table 3.2 shows the results of reliability, uniformity, and uniqueness in the four boards. The metrics are calculated for PLPUF (the building block of CaPUF) and CaPUF. The uniqueness of PLPUF is slightly better than the uniqueness of CaPUF as it is closer to the value of 50%, both are still under

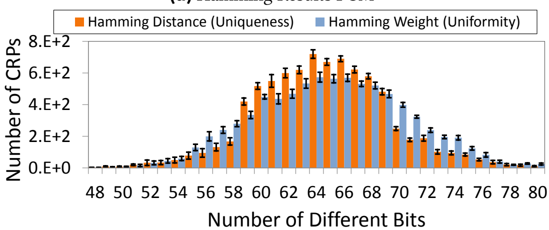
60%. However, CaPUF has an improvement for the uniformity over PLPUF as it is almost the ideal value 50%. In terms of reliability, PLPUF was able to achieve a remarkably high value of 98.47%. CaPUF has a lower reliability, which can be understood as it is based on some chains of PUFs, which reduces overall reliability. This does not affect the performance of CaPUF, as it is still in the range where fuzzy extractors can extract the response on the verifier side [118].

3.6.2 ANV-PUF Quality and Performance

In a similar fashion, the uniformity and uniqueness are calculated for the generated CRPs for ANV-PUF$_{RRAM}$ and ANV-PUF$_{PCM}$. They are in a very good range for both NVM technologies. For ANV-PUF$_{PCM}$, uniqueness and uniformity have a value of 50.88% and 50.41%, and for ANV-PUF$_{RRAM}$, 50.34% and 49.27%, respectively.



(a) Hamming Results PCM



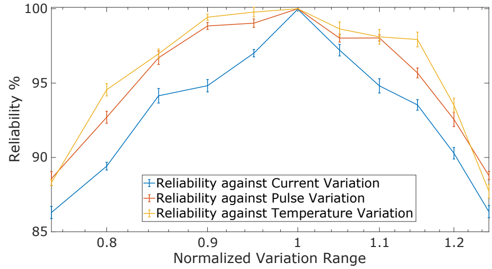
(b) Hamming Results RRAM

Figure 3.9: Hamming Results for (a) ANV-PUF$_{PCM}$ and (b) ANV-PUF$_{RRAM}$, both have the desired Gaussian distribution around the mid value 64 for Hamming distance between responses and Hamming weight of the responses. The error bars show the confidence interval for the 1000 experiments.

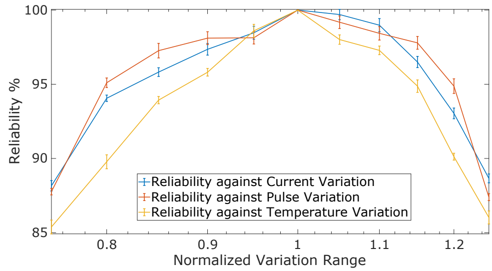
Figure 3.9 shows the histogram for uniqueness and uniformity of both ANV-PUF_{PCM} and ANV-PUF_{RRAM}. In general, both versions of ANV-PUF have a Gaussian distribution of around 64 with no offset. Hence, the uniformity and uniqueness of both versions are in the desired range. The main difference between both is in the standard deviation. For ANV-PUF_{PCM}, the standard deviation of uniformity and uniqueness is 4.37% and 4.35%, respectively. As for ANV-PUF_{RRAM}, it has a lower standard deviation of uniqueness at 3.71% and a slightly lower standard deviation of uniformity at 4.24%.

Based on the three noise sources, the system is simulated and the CRPs are calculated. Then the Hamming distances of the responses under noise simulation vs. the noise-free golden reference are calculated. The reliability under noise effects gets degraded as expected. However, as Figure 3.10 shows, the reliability of both ANV-PUF_{PCM} and ANV-PUF_{RRAM} does not drop below 85%, even with noise reaching 25% of the normal value of pulse width, pulse amplitude, and ambient temperature. ANV-PUF_{RRAM} is affected most by variation in temperature and least by pulse amplitude noise. In contrast, ANV-PUF_{PCM} is affected least by noise in the pulse width and most by the pulse amplitude noise. Note that this reliability degradation occurs, as no ECC techniques are used, in case a technique such as the technique used by [200] or by [117] would be used, then the reliability would stay higher. Moreover, the reliability does not get worse than the theoretical limits, where the PUF output is no longer usable, as stated by [117, 118], i.e. the PUFs are usable without ECC, even though their reliability under noise can be improved even further when needed.

The Matlab reliability simulation is not used solely. Rather, a SPICE simulation environment is used to inspect the noise sources and their effect on the SA output. The noise effects are shown in Figure 3.11, and they are similar for both ANV-PUF_{PCM} and ANV-PUF_{RRAM}. The noise contribution of the NVM cell is generally the lowest for both ANV-PUF variations. However, the noise contribution of the PCM cell is greater than the noise contribution of the RRAM cell. As for the contribution of all the other circuit components, the contribution of the DAC is the highest. This is expected as it is the first circuit component and any noise it contributes will be further propagated across the circuit. Moreover, since it generates the reference voltage V_{DAC}, any noise from it will affect the comparison by the SA. The second highest is the noise contribution of V_{NVM}. Any change in the amplitude or width of the pulse will affect the reset, partial sets, and read signals, which consequently will affect the input voltage to the SA and reflect on its output. The noise



(a) Reliability against noise for ANV-PUF_{PCM}



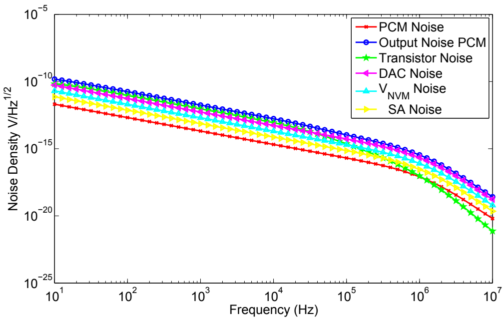
(b) Reliability against noise for ANV-PUF_{RRAM}

Figure 3.10: Reliability against noise in voltage amplitude, pulse width, and ambient temperature variation. Overall with extreme noise levels and extreme high and low temperatures, the reliability does not drop below 85%. The error bars show the confidence interval for the 1000 experiments.

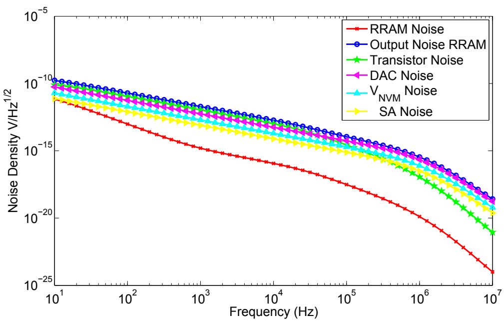
contribution of both the SA and the transistor are similar. The SA comes as the final component, and hence, its noise does not get propagated any further. As for the transistor, it acts as a mere switch and has no strong impact on the functionality of the circuit. These results show that lowering the noise of each component to a level comparable to the noise of the NVM cell would reduce total noise and enhance reliability.

3.6.3 CaPUF's Resilience against ML-based attack

The collected CRPs are not only used to evaluate the CaPUF performance metrics but also to evaluate whether or not the ML-attacks would be successful against the CaPUF. The state-of-the-art open-source LR and SVM modeling



(a) Contribution of noise for ANV-PUF$_{PCM}$



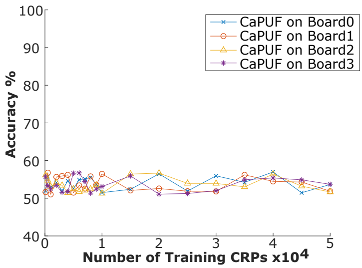
(b) Contribution of noise for ANV-PUF$_{RRAM}$

Figure 3.11: Noise contribution of each circuit component for the final output of the SA for (a) ANV-PUF$_{PCM}$ and (b) ANV-PUF$_{RRAM}$

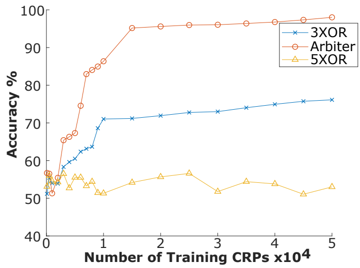
attacks of [199] are trained based on the collected CRPs. The attack works as follows: It uses a separate model for each response bit with two classes representing both binary values. The challenge value is the single feature used for training. It performs 1000 iterations on the classifier's parameters (LR or SVM) to fine-tune the model.

The attacks attacks are used against all implemented CaPUFs. Up to 50,000 CRPs were used to train the models, the rest are used to test and obtain the prediction accuracy.

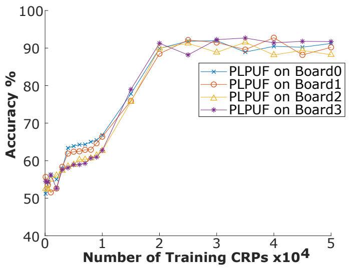
The results for both attacks are similar as shown in Figure 3.12. APUF was easily breakable by both attacks. It required around 10,000 CRPs to be almost



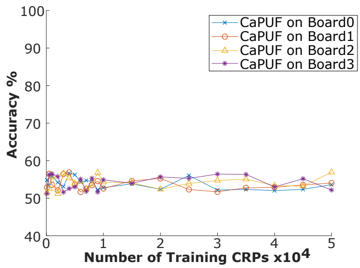
(a) LR on CaPUF



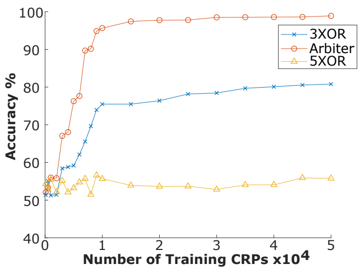
(b) LR on Arbiter-based PUFs



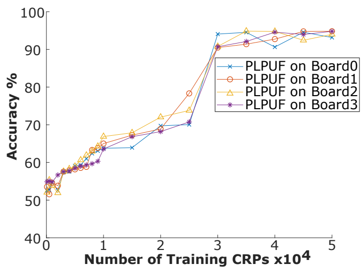
(c) LR on PLPUF



(d) SVM on CaPUF



(e) SVM on Arbiter-based PUFs



(f) SVM on PLPUF

Figure 3.12: SVM and LR attacks on CaPUF, PLPUF, APUF, 3-XORPUF, and 5-XORPUF. The attacks are only as good as flipping a coin on CaPUF and 5-XORPUF, but they have a better performance against 3-XORPUF, and they are successful against APUF and PLPUF. Board0 to Board3 denote the four different boards used to implement the PUFs.

completely predictable and became partially predictable using even less CRPs. This is in line with many of the previous works [39, 99, 199]. PLPUF was also broken by both attacks. It showed to be harder to break than APUF, and it was easier to break it by using the LR attack.

3-XORPUF was partially broken. With enough CRPs the bits were predictable up to 75% accuracy. But neither of the attacks was able to fully model it. Only 5-XORPUF and CaPUF were not broken by the attacks. The accuracy of the prediction did not reach higher than 57% for CaPUF and 59% for 5-XORPUF. Note that the prediction accuracy is per individual bit of the response, that is, the chance of correctly predicting every bit. This is why the ideal value is 50% (similar to flipping a coin on each bit). If the accuracy is too high or too low, then modeling the PUF worked well. For low accuracy, the attacker only needs to invert the values, and then the bits will be of mostly correct values.

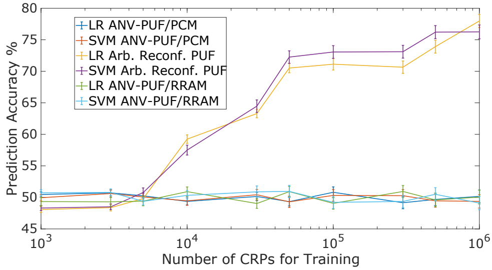
3.6.4 ANV-PUF's Resilience against ML-based attack

Similar to CaPUF, the generated CRPs (including the 8-bit reference voltage value) from the experimental setup are used to evaluate the success of ML-modeling attacks against ANV-PUF. As the simulation of ANV-PUF produced significantly more CRPs than running CaPUF on hardware, the evaluation of ANV-PUF is not limited to SVM and LR but extends to the use of NN and CMA-ES. For SVM and LR, the attacks are the same open-source attacks used for CaPUF. For NNs and CMA-ES, there are no open-source attacks, so the attacks are replicated using the same parameters detailed in [190]. Table 3.3 shows the parameters for each ML model. SVM uses a fourth-degree polynomial as its kernel to be able to catch nonlinear behavior. LR uses the Newton-Cholesky solver, which is recommended for a problem where the number of samples is significantly higher than the number of features and the best matching parameters for this solver. The NN of a 4-layer dense network (based on [190]), with each layer having 32 nodes with relu activation. CMA-ES uses a standard deviation of 0.5 and allows for up to two training restarts from the best point reached.

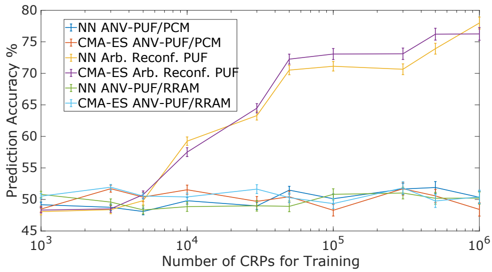
The attacks are run on both variations of ANV-PUF and on the Arbiter Reconfigurable PUF from [30]. The Arbiter Reconfigurable PUF uses RRAM and switches between LRS and HRS instead of using switches as in conventional APUF. Its model is built by using the same RRAM model used for

Table 3.3: Model parameters of the ML-models used for attacks

ML-Model	Parameters
SVM	Kernel: poly, gamma: auto, degree: 4
LR	penalty: l2, dual: false, solver: Newton-Cholesky
NN	Layers: 4, nodes per layer: 32, activation: relu
CMA-ES	sigma: 0.5, restart: from best, restarts: 2



(a) Results of LR and SVM



(b) Results of NN and CMA-ES

Figure 3.13: Resilience of ANV-PUF against ML modeling attacks, the ideal prediction accuracy should be 50% which is as good as flipping a coin. ANV-PUF prediction accuracy stays in the 50% range. The error bars show the confidence interval for the 1000 experiments.

ANV-PUF_{RRAM} and generated the same number of responses based on the same challenges. Figure 3.13 shows the results of predictability using the attacks. The attacks are run several times, gradually increasing the size of the training set from 1,000 CRPs to 1,000,000 CRPs. The predictability of ANV-PUF for both of its variations remains in the ideal range of around 50%. In contrast to ANV-PUF, the predictability of Arbiter Reconfigurable

PUF reaches 78% when 1,000,000 CRPs are used for training. This is higher than the predictability of 65% reported in [30] however they used a smaller training set. Note that when using a training size of 10,000 CRPs (as in [30]), the numbers match the range of 65%, as can be seen from Figure 3.13. In all cases, the predictability of Arbiter Reconfigurable PUF is significantly higher than the ideal range of around 50%. The reason for this higher predictability is that it does not use the MLC property of NVM. It only uses the switching between LRS and HRS to change the delay across the PUF elements.

3.6.4.1 Using Long-Short-Term-Memory Classifier

ANV-PUF incorporates a finite state machine within its structure. While its security does not depend on the finite state machine but rather on the difference between the used NVM cells and the accumulation of the sums. Next it is investigated if Neural Networks with memory, e.g., a Long Short-Term Memory (LSTM) classifier, would lead to better results than existing state-of-the-art attacks.

As ANV-PUF produces 128 response bits sequentially, they are used as the input time series for an LSTM classifier with multivariate multiple input series. The LSTM is trained using 1,000,000 CRPs (same as for the state-of-the-art attacks) and tried four different configurations as follows

1. Activation Function: Relu, Optimizer: Adam, epochs = 50, steps = 16
2. Activation Function: Relu, Optimizer: Adam, epochs = 100, steps = 32
3. Activation Function: Relu, Optimizer: Adam, epochs = 150, steps = 48
4. Activation Function: Relu, Optimizer: Adam, epochs = 300, steps = 64

For all four configurations, the prediction accuracy does not exceed the range of 50%. Note that these results are not conclusive and do not show that no LSTM-based ML model would be capable of attacking ANV-PUF. However, no state-of-the-art attack is capable of attacking ANV-PUF. And also the examined LSTM-based attacks were not.

Table 3.4: CaPUF Comparison to the related works

PUF	ML-Resil.	Locking	LUTs	Memory	Time
CRC	yes	no	5120	0	N.A.
PLPUF	no	no	32	0	1 cycle
CaPUF	yes	no	329	0	16 cycles
Arbiter	no	no	1024	0	1 cycle
3-XOR	partial	no	3072	0	1 cycle
5-XOR	yes	no	5120	0	1 cycle
Slender	yes	no	1168	68 KB	N.A.
CT	yes	no	3072	0	1 cycle
LPN	yes	no	49K	7 KB	N.A.
FSM	yes	yes	1082	68 KB	2508 cycles
IPA	yes	yes	1060	2 KB	18764 cycles
NoPUF	partial	obfusc.	1024	0	1 cycle

3.6.5 Comparing CaPUF to the State-of-the-Art

CaPUF is to the state of the art silicon-based PUFs. The comparison is done for all PUFs assuming that they have 32 bit challenges and 16 bit responses, which are the same bit widths as those of CaPUF. The numbers for area and latency are based on implementing them on FPGA, based on their description of their respective works, or the numbers reported by the work itself. Hence, the numbers are in LUTs and memory is in Kilobytes.

CaPUF is the smallest ML-resilient PUF. As a matter of fact, only PLPUF is smaller than CaPUF, even APUF requires more hardware than CaPUF. The closest in terms of area to the implementation from the ML-resilient PUFs is CT, which requires almost 10× more LUTs than CaPUF. Slender requires fewer LUTs than CT; however, it uses 68 KBs of memory, which would require more area.

It can be seen that all cryptography-based PUFs have significant resource usage. For example, the stream cipher dominates the resource usage of CRC. LPN has the highest resource usage; it is the only one that exceeds 10 K LUTs. This is because of its on-chip implementation of the error correction code. FSM and IPA both use significantly less hardware than LPN. However, both require thousands of clock cycles to complete their computation. The reason for this is that they use a lightweight serial hash function which is slow. Additionally, for IPA, it uses voting rounds instead of error correction, which adds a significant delay overhead. These delay overheads are very problematic in the enrollment phase, as collecting the CRPs requires a three to four orders of magnitude longer time for each individual IC. The delay of the three CaPUF modes is also slightly higher than the other PUFs. However, it is similar to the serial implementations of weak PUFs when they try to output several bits, so it is in the acceptable ranges [34].

3.6.6 ANV-PUF's Comparison to the State-of-the-Art

NVM-based PUFs, compared to ANV-PUF, are mostly weak Table 3.5 and lack the CRP range of ANV-PUF, so they report results directly. The comparison includes other PUF metrics beyond ML resilience. ANV-PUF is both strong and ML-resilient, unlike current PUFs: [89, 202] which are ML-resilient but weak, and [30] which is strong but not fully ML-resilient. PUF in [30] achieves full ML resilience using the xor method [177] across four parallel PUF instances, increasing area and energy overhead. Some attacks target this xor method [39] and are not covered in [30].

PUFs are compared for uniqueness and uniformity. PUFs of [89, 202] don't assess uniformity as they are weak. PUF of [202] has low uniqueness, unlike the ideal uniqueness in [89]. For the PUF of [30] and both ANV-PUF variations, both uniqueness and uniformity are ideal. Energy consumption per response bit is considered. ANV-PUF consumes the most energy due to its use of multiple NVM cells with iterative set pulses. The Reconfigurable Arbiter PUF of [30] uses one set/reset pulse per cell for one bit. This comparison excludes the energy overhead from xoring four parallel PUFs. Weak PUFs of [89, 202] use sequential sets/resets with fewer cells, lowering energy consumption. The focus is on PUF implementation, without considering energy for hash functions that enhance uniqueness.

3.6.6.1 Comparison to Strong PUF of Ref. [30]

Now the focus is on a comparison between ANV-PUF and the PUF of [30] as it is the closest to ANV-PUF. Both the PUF of [30] and ANV-PUF are based on the famous Arbiter PUF design [177]. This is the first design of a PUF and is still widely used as the base for novel PUFs of different types [180,

Table 3.5: Comparison to the related works, PUF of [30] is strong but not fully secure. PUFs of [89, 202] are secure but not strong. ANV-PUF is both secure and strong.

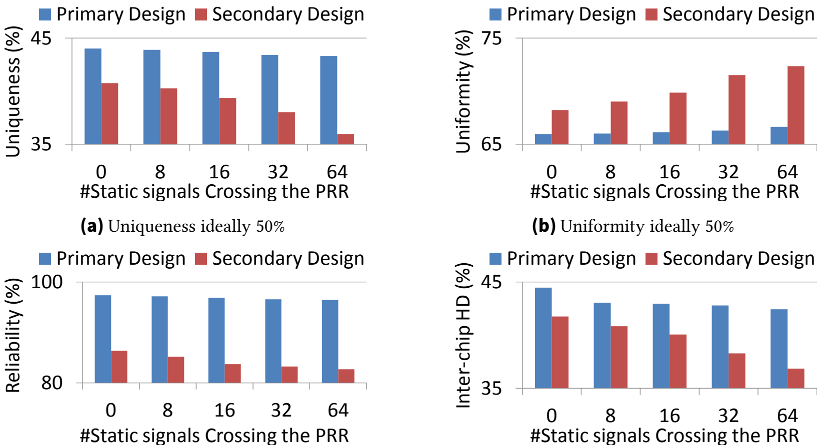
PUF	ML-Resil.	Strong	Unique	Uniform	Energy
ANV-PUF$_{PCM}$	yes	yes	50.88 %	50.41 %	6.5 μj
ANV-PUF$_{RRAM}$	yes	yes	50.34 %	49.27 %	7.2 μj
PUF of [30]	partially	yes	50.21 %	$\sim$ 50 %	575 nj
PUF of [202]	yes	no	30 %	N.A.	1.9 μj
PUF of [89]	yes	no	49.69 %	N.A.	150 nj

190]. There are two similarities with the PUF of [30]. First: Using the Arbiter PUF as the base of the design, and second: Using NVM as a building block. However, both PUFs are significantly different.

The PUF of [30] uses the propagation delay of signals that traverse the NVM cell, as it differs significantly depending on whether the NVM cell is in high resistance state (HRS) or low resistance state (LRS). Hence, it does not benefit from the MLC character of NVM cells. In contrast, ANV-PUF focuses on leveraging the MLC character of NVM cells. ANV-PUF does not rely on the propagation delay of signals through NVM cells. It rather obtain the unclonability and ML-resilience from the number of electric pulses needed for each cell to reach an arbitrary resistance level.

3.6.7 CaPUF's Reliability on FPGAs

CaPUF can be implemented on FPGAs as a reconfigurable accelerator. The impact of Dynamic Partial Reconfiguration (DPR) on CaPUF's performance is assessed. Since CaPUF uses PLPUFs, their performance is studied; if they degrade, CaPUF will degrade as well. Each PRR includes static routing from neighboring areas via a communication channel through the PRR. This communication is likely routed partially through the PRR. Trials change the communication channel bitwidth from 0 to 64 bits. PLPUF is implemented as a primary design (PRR is fine-tuned) and secondary design (PRR just fits the PUF). Figure 3.14 shows the results. Increased signals crossing the PRR degrade PUF performance in both primary and secondary designs. In the primary design, degradation remains within 1%. In the secondary design, performance is always worse and can degrade up to 5%.



(a) Uniqueness ideally 50%

(b) Uniformity ideally 50%

(c) Reliability ideally 100%

(d) inter-chip HD ideally 50%

Figure 3.14: The performance of PLPUF degrades significantly when used as a secondary design, especially with the increase of static routing crossing the PRR

3.6.8 ANV-PUF Endurance

NVM cells degrade with repetitive usage [14, 15, 22, 24] and have a limited lifespan. An endurance analysis methodology, as shown in Figure 3.15, utilizes a Markov chain to determine the probability of ANV-PUF failing after N challenges. The analysis primarily examines ANV-PUF_{PCM} but generally applies to ANV-PUF_{RRAM} as well.

3.6.8.1 Analyzing States

The Markov chain evolves state probabilities using the transition matrix. Each state's probability is a vector, updated by applying the transition matrix to the current vector. All transitions start from the "Receive Challenge" state, so the initial state is a one-hot vector with a probability of one for the "Receive Challenge" state.

The proposed state machine, which includes a terminal state with zero transition probability to other states, will converge to this terminal state. The evolution process stops when the probability of reaching the termination

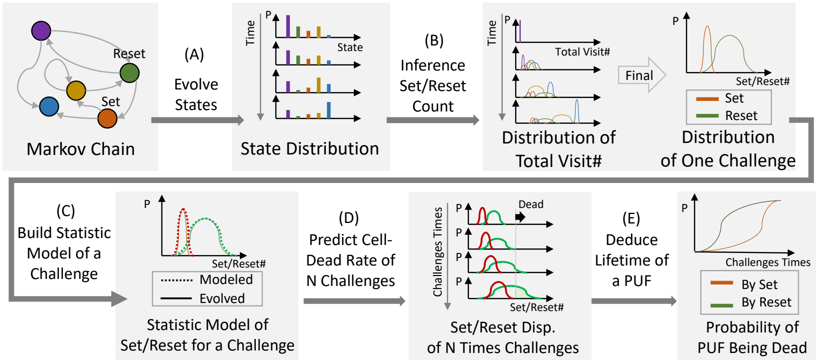


Figure 3.15: Flow to analyze the endurance of the PUF

state is 1 - 10^{-5}. Iterative updates of the state vector determine the probability of each state at each iteration. Each state is denoted by $P_m(t, s)$, representing the probability that state s is visited at iteration t. These records calculate the distribution of accumulated visit time for each state.

3.6.8.2 Inference Set/Reset Count

To assess the PUF system's endurance, it's crucial to calculate accumulated set/reset operations. First, determine the distribution of total state visits per challenge. Next, derive the final distribution for set and reset operations. The total visit count is obtained by converting the recorded transition state probabilities using the following equations.

$$P_c(N|t, s) = P_c(N-1|t-1, s)P_m(t-1, s) + KEEP(t, s) \tag{3.20}$$

$$KEEP(t, s) = P_c(N|t-1, s)(1 - P_m(t-1, s)) \tag{3.21}$$

$$P_c(0|t, s) = \begin{cases} 1, & \text{if } t = 0 \\ KEEP(t, s), & \text{otherwise} \end{cases} \tag{3.22}$$

In Eq. (3.20), $P_c(N|t, s)$ denotes the probability of state s being visited N times by iteration t. This probability comes from two parts: the probability that the visit count just reached N at iteration $t-1$, from Eq. (3.20), and the probability that the visit count reached N before iteration $t-1$ with no further updates, from Eq. (3.21). The initial condition is in Eq. (3.22), where the probability of not visiting any state is set to one initially, and the probability of staying in the non visiting state is given by $KEEP(s,t)^t$. As the system transitions to the ending state, the set and reset operation states become unvisited and converge to a stable distribution, defining the set and reset distribution for a given challenge.

3.6.8.3 Modeling of Set/Reset Distribution

The model considers the set and reset counts of a challenge as random variables that follow the distribution introduced in Section 3.6.8.2. Based on this distribution, the model deduces the set and reset counts after N challenges. The distribution of the total number of set and reset operations after N challenges is derived by summing N independent random variables, each following the set and reset operation distribution of an individual challenge. Finally, the model obtains the time-variant distribution of the set and reset operations, enabling the deduction of the lifetime of the system.

3.6.8.4 Deduce Lifetime

The model sets an endurance limit for a cell's set and reset operations, beyond which the cell is considered dead. A PUF with M cells is dead if 15% of its cells are dead. The goal is to find the PUF failure probability based on this and the distribution of set and reset operations. For each challenge t, the probability of a cell being dead is the chance its set and reset operations exceed the endurance limit Eq. (3.23). Since cells operate independently, the number of dead cells among M follows a binomial distribution Eq. (3.24). The PUF failure probability is the sum of probabilities for having $k > 0.15M$ to Eq. (3.25) dead cells. The 15% limit is a standard assumption [117, 118], and the lifetime threshold is taken from the mid-range value in [46] for PCM cells.

$$P_{cell}(dead|t) = P(set\ or\ reset\ ops. > limitation)|_t \tag{3.23}$$

$$P(k\ dead|t) = C_k^M P_{cell}^k (1 - P_{cell})^{M-k}|_t \tag{3.24}$$

$$P(PUF\ dead|t) = \Sigma_{k>0.15M} P(k\ dead)|_t \tag{3.25}$$

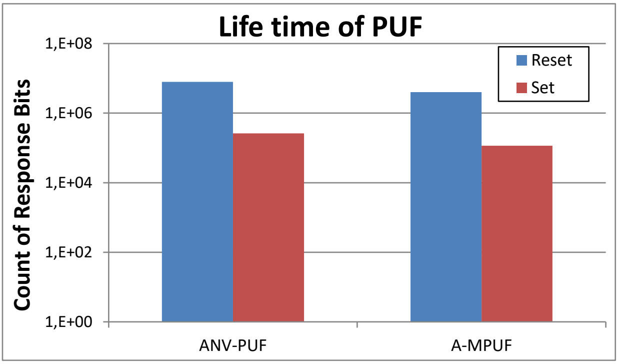


Figure 3.16: Lifetime of both ANV-PUF [1] and PUF from [30]

Based on the analysis above, both ANV-PUF [1] and the PUF from [30] are run through the lifetime analyzer. As Figure 3.16 shows, ANV-PUF has a slightly better lifetime.

3.7 Summary

This chapter tackles the issue of authenticating communication between clients and servers using ML-resilient PUFs. It introduced two novel PUF designs: CaPUF and ANV-PUF, aiming for strong resilience against ML-based modeling attacks while remaining lightweight for resource-constrained devices. CaPUF uses a cascaded architecture of PLPUFs, dynamically changing its response to complicate modeling attacks. Its modular structure and dynamic frequency control enhance its unpredictability, making it harder to model than traditional PUFs. On the other hand, ANV-PUF uses the MLC property of NVM cells for non-linearity in response generation, improving resilience to ML attacks. Using a variable voltage reference with challenge bits, ANV-PUF increases modeling difficulty. Both CaPUF and ANV-PUF showed strong metrics of uniqueness, uniformity, and reliability. Evaluations against state-of-the-art ML attacks confirmed robustness, with prediction accuracy around 50%, indicating resistance. Additionally, the chapter provides analysis the reliability of PLPUF the basic block of CaPUF as a runtime reconfigurable accelerator and the lifetime of ANV-PUF.

4 Identifying and Mitigating Covert Channels on FPGA-Accelerated Cloud Systems

While the data is transmitted from the client to the server in ciphertext form securely, the processing itself usually happens on the plaintext. Therefore, covert-channel attacks pose a significant threat as they exploit the shared nature of multi-tenant FPGA resources, allowing one tenant to infer sensitive information [51, 73]. This chapter explores two such covert channel attacks, one based on power consumption and the other on thermal emissions, and propose countermeasures to mitigate these threats.

Power-based covert channels leverage fluctuations in the FPGA's PDN to transmit information between colluding malicious tenants [73]. By carefully modulating their power consumption, these tenants can communicate without directly interacting with each other, bypassing traditional security measures. Similarly, thermal-based covert channels use the heat generated by FPGA circuits to encode and transmit data [85]. Variations in temperature, which are easily detectable by thermal sensors, provide a covert means of communication that is difficult to detect and counteract.

Both types of attacks pose significant risks when FaaS is offered in accelerated cloud systems, where multiple tenants share the same physical FPGA resources. If left unaddressed, these covert channels could lead to data breaches and other security violations. Therefore, the development of effective countermeasures is crucial to ensuring the security and reliability of FPGA-accelerated cloud systems [204].

This chapter is based on contributions from [4–7].

4.1 Motivational Example

As a motivational example, consider a scenario in a secure environment where a colluding application from a dishonest vendor operates within the secure world of a TEE-enhanced FPGA-MPSoC. This application uses a benign accelerator in the PL of the FPGA to modulate the temperature of the PL (1). Meanwhile, a malicious receiver, running in the normal world, accesses the temperature sensor of the PL (2), which is available in the normal world, to decode the messages transmitted.

Figure 4.1 illustrates this scenario, where the temperature of the PL in the FPGA-MPSoC is exploited as a means of covert communication. This setup allows for information transfer between applications in different security contexts, effectively creating a covert channel within the FPGA infrastructure of an accelerated cloud system.

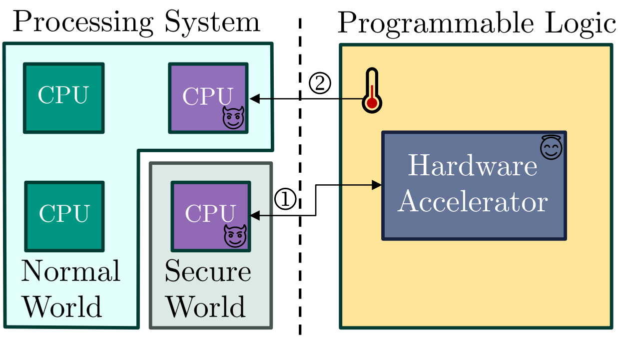


Figure 4.1: Overview of a thermal covert channel on a TEE-enhanced FPGA-MPSoC.

4.2 Problem Statement

In accelerated clouds, the usage of MPSoC with integrated FPGA, i.e., FPGA-MPSoCs is increasing [110]. These FPGA-MPSoCs could allow multiple tenants, in software or hardware, to share the same physical infrastructure. This shared infrastructure, while efficient, introduces significant security risks, particularly through the use of covert channels for intra-chip communication. These channels enable malicious tenants to bypass traditional security mechanisms, leading to threats such as data leakage or the coordination of more complex attacks on the system.

As illustrated in Figure 4.2, the ability of tenants to communicate covertly within the same FPGA-MPSoC presents a serious vulnerability. They can use software and hardware components on the FPGA-MPSoC to engage in covert communication, potentially leading to unauthorized data transfer or the synchronization of malicious activities across different tenants. This poses a critical challenge in maintaining the security and integrity of FPGA-accelerated cloud systems, where isolation between tenants is presumed but not always guaranteed.

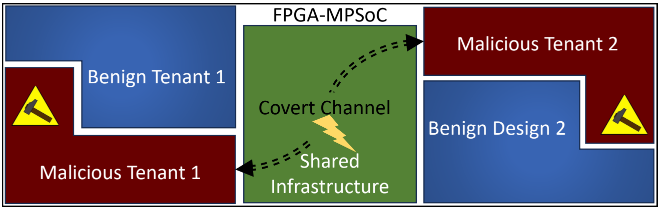


Figure 4.2: Problem Statement: Multiple tenants on an FPGA-MPSoC whether in software or hardware use covert channel intra-chip communication. This leads to threats of data leakage or possibility to coordinate attacks on the system.

4.3 Contributions

The key contributions of this chapter are as follows:

- It introduces a thermal-based covert-channel attack, where temperature variations in the FPGA are used to encode and transmit sensitive information between colluding tenants and breaking the TEE isolation.
- It proposes a novel power-based covert-channel attack that allows multiple malicious tenants to coordinate and synchronize fully duplex communication.
- It proposes countermeasures for covert channels, both on the software and hardware level.

4.4 Previous FPGA-based Covert Channels

Covert channels have been vastly demonstrated for FPGA-based systems over the recent years. The PDN has been the main mechanism exploited to implement channels in multi-tenant FPGAs and FPGA-MPSoCs. In this context, the attacks have targeted the device's voltage [78, 82, 87] and frequency [42, 68] as the medium to modulate the power, obtaining fast transmission speeds and low error rates. Other approaches for covert channels use non conventional means to modulate different resources on FPGA such as PCIe usage for cloud systems [79] or internal wiring [77] in multi-tenant FPGA systems.

4.5 Novel Covert Channel Attacks

This chapter shows that the threat of covert channels is more complex than previously shown in the literature. It exploits one vulnerability in FPGA-MPSoCs to break the isolation in TEEs. Moreover, it shows how covert communication can be established on FPGA-MPSoCs in a multiparty manner.

4.5.1 Through-Fabric: Cross-world attack on FPGA-MPSoC

The first covert channel attack is a thermal covert channel on TEE-enhanced FPGA-MPSoCs. On FPGA-MPSoCs, the usage of hardware accelerators by Trusted Applications (TAs) introduces a vulnerability exposed by Through-Fabric. As isolation focuses on separation between applications on the PS side, an attacker can still use the PL shared medium, even if the accelerators are separated, to leak data.

4.5.1.1 Threat Model and Assumptions

The basic threat model for the thermal covert channel follows the same principle as other covert channels [144], where a malware and a spy application communicate with each other in an illegitimate way.

The spy or *receiver* application, runs in the normal world as a Customer Application (CA). On the other hand, a malware *transmitter*, which is a malicious or colluding application that gets triggered by an unaware innocent application to perform a security-critical operation on its behalf, is assumed to function following the same model as other TEE-focused covert channels [55, 144]. In the context of TEEs, this colluding program is a Trusted Application (TA) that executes in isolation in the secure world through the OP-TEE framework, while the triggering program is a Customer Application (CA) in the normal world. In a practical scenario, this colluding application could be provided by a dishonest vendor, or a vulnerable TA could be exploited by an attacker. The spy or *receiver* application runs in the normal world as a CA belonging to the same or even potentially a different tenant. Neither the *transmitter* nor the *receiver* require any privileged permissions to perform any of the operations involved in the attack. A real example of this type of colluding application is the recent *xz utils* vulnerability [148], where through an elaborate supply chain attack, an adversary inserted a backdoor into *xz utils*, a widely used open source library, and almost succeeded in merging malicious updates into major Linux distributions before detection.

Differently from other FPGA-based approaches in the state-of-the-art, Through-Fabric uses a completely benign hardware accelerator to modulate the temperature of the PL. Moreover, neither the transmitter nor the receiver modules of the attacker require to implement any extra hardware on the FPGA logic.

4.5.1.2 System Overview

To get Through-Fabric to work, several system components are involved. They can be divided into two main parts: software and hardware. The software part establishes the communication cross-worlds. As for the hardware part, the transmitter software uses it as the heating mechanism, illustrating the feasibility of the attack without the need for any custom modifications to the hardware.

Software: A simplified overview of the software components of the system is depicted as a flow in Figure 4.3. In the normal world, the innocent CA intends to perform hardware-accelerated AES decryption on a ciphertext through the decryption TA, which resides in the secure world. To do so, the innocent CA uses the OP-TEE client API (1) to invoke the AES decryption TA, unaware of its malicious nature. In turn, the OP-TEE client API routes the request to the OP-TEE driver (2) in the Linux kernel. On the Linux kernel side

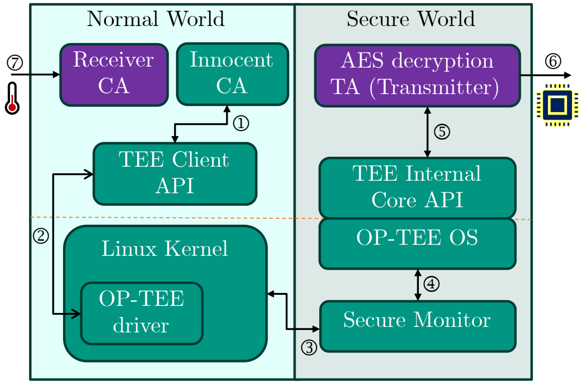


Figure 4.3: Overview of the software components of Through-Fabric

of the normal world, the OP-TEE driver then directs the request to the secure monitor (3) on the secure world, which itself handles the communication between worlds by routing the request to the OP-TEE Trusted Operating System (4). Through the internal API, the OP-TEE OS framework determines the malicious AES decryption TA as the one invoked and passes control to it to handle the request (5). Finally, the TA uses the hardware accelerator API (6) to perform the decryption. In a normal (benign) operation, at this point, the execution control would return in a reversed path back to the CA with the ciphertext decrypted. However, due to its malicious nature, before returning control, the TA leverages the hardware accelerator **again** (6) to encode and leak secret data (for example, key or plaintext) by modulating the temperature of the programmable logic on the FPGA.

Notably, the TA is configured to keep the execution context (instance) after the sessions ends, using the `TA_FLAG_INSTANCE_KEEP_ALIVE` flag [144]. This allows the TA to store and further leak private data after the transaction has finished.

In the normal world, another malicious application (i.e., the receiver CA, potentially owned by a different tenant) continuously reads the temperature sensor of the FPGA (7) to detect the beginning of the transmission and decode the secret being sent by the TA, hence establishing an illegitimate communication channel between the secure and normal world.

Hardware: To demonstrate the attack, an existing hardware accelerator is used as the heating mechanism. The processor residing in the trusted world is connected to one accelerator on the PL. The connection is done via an AXI crossbar. The benign AES 128 bit decryption engine from [135] is the accelerator used to heat the chip. It is available as open source and is highly parallelized. The AES engine receives the cipher text and decryption key as input from the processor and returns the plain text in one clock cycle.

Notably, while a custom and more power hungry hardware accelerator (e.g., ring oscillators) would benefit the transmission by heating faster, depending on the attacker to compromise the hardware or implement their own logic could be either impractical or easy to detect [10, 11]. The decision to employ a completely benign module is to match a realistic use case. The selected AES engine performs a security-related operation, which justifies its use from the secure world, while also being a tested accelerator provided by an honest vendor.

4.5.1.3 Transmitter

The transmitter module of the attack is implemented within the malicious decryption TA. Algorithm 1 shows the implementation logic for the transmission. Upon being invoked ((5) in Figure 4.3), the TA performs the normal (benign) decryption of the ciphertext. However, as a malicious application, the TA proceeds to leak the newly decrypted plaintext by modulating the temperature of the FPGA. To do so, it first computes the number of decryptions needed to encode a bit of ‘1’, and the time it requires to wait to encode a bit of ‘0’, using the desired bit rate and the latency accelerator. Then for each bit in the secret, the TA performs the extra decryptions for each bit value that is a ‘1’, or waits for the corresponding time for each bit of ‘0’. Finally, the application returns the plaintext normally to the calling CA.

In the case of a long secret, in order to avoid suspiciously long decryption delays, the transmitter module can leverage the `TA_FLAG_INSTANCE_KEEP_ALIVE` flag, to save it, while only leaking a few bytes at a time per call, especially with short ciphertext decryptions. On further calls, the attacker can obfuscate the transmission of the rest of the message by leveraging the decryption of longer ciphertexts.

Algorithm 1: TA transmitter for the TCC

```
Input: ciphertext: encrypted text from innocent CA
Result: plaintext: decrypted text
plaintext ← HwAESDecrypt(ciphertext);          /* Performs normal
  decryption */
N ← 1/(bit_rate * latency_acc) ; /* Calculate the required number
  of decryptions to heat up enough to encode a '1' */
tdown ← 1/(2 * bit_rate);
for bit in secret do
    if bit is 1 then
        for i = 0 to N-1 do
            HwAESDecrypt(randominput) ;          /* Perform extra
              decryptions to increase temperature */
        end
    else
        TEE_Wait(tdown) ; /* Sleeps to cool down the hardware */
    end
end
return plaintext;
```

4.5.1.4 Receiver

The receiver is implemented as an application that runs in the normal world. It performs three simple steps which are shown as the three loops in algorithm 2. The first step (line 2 to line 6) is to continuously collect the data from the PL temperature sensor and store it in an array of size $total\_samples$. This step involves only reading data from a register. Once it collected the samples, it filters out the data to the desired frequency of communication (line 7 to line 11). The receiver performs this by calculating the Fast Fourier Transform (FFT) over a moving window and keeping only the bins that correspond to the frequency of communication.

The final step is to decode the filtered data into the corresponding sent bits (line 12 to line 27). To achieve this, the receiver applies two criteria: an absolute value and a gradient. During the moving window corresponding to each bit, if a value higher than a precomputed high threshold (ρ_1) is achieved, then a bit value '1' is interpreted. Similarly, if the maximum value is lower

Algorithm 2: Normal world receiver for the TCC

```
Result: msg: demodulated message
s ← 0;
while s < total_samples do
    temps[s] ← readTemp() ;    /* Get new temperature reading */
    sleep(sampling_time);
    s++;
end
N ← total_samples − win_size ;    /* Compute number of windows */
for i in N do
    X ← FFT(temps[i : i + win_size]);    /* Computes FFT */
    filtered[i] ← X[ω_k] ;    /* Filter the data at ω_k */
end
for j in N do
    bit_w ← filtered[(j ∗ win_size : j ∗ win_size + win_size] ;    /* get
     the demodulated bit window */
    if max(bit_w) > ρ_1 then
        msg[j] ← 1 ;    /* if high absolute change bit is ‘1’ */
    else
        if max(bit_w) < ρ_2 then
            msg[j] ← 0 ;    /* if low absolute change bit is ‘0’ */
        else
            if |slope(bit_w)| > δ_H then
                msg[j] ← not(msg[j − 1]) ;    /* bit flipped */
            else
                msg[j] ← msg[j − 1] ;    /* bit stayed the same */
            end
        end
    end
end
return msg;
```

than the pre-computed low threshold (ρ_2) then a bit value ‘0’ is interpreted. However, when multiple bits of the same value are sent sequentially, the temperature saturates to a value between. To solve this, the gradient between two consecutive samples is computed as a moving slope. If the slope of the readings from the moving window is greater than the precompute threshold for a high slope δ_H, it means that a rapid change of temperature occurred. Consequently, a bit with value opposite to the previously received bit is transmitted, and the value is toggled based on the last received bit. Otherwise, if the slope is low, it means that the same value is received and no toggling occurs.

To set the thresholds ρ_1, ρ_2, and δ_H a subset of the sent data is analyzed. Each bit is sent over a period T and an interval of temperature samples t is recorded. The temperature recorded for the '1' bits in the dataset t_1. Similarly, the temperatures for '0' bits are collected in the data set t_0. The high threshold (ρ_1) is set as $\rho_1 = \mu(max(t_1)) - \sigma(max(t_1))$ with μ being the average and σ being standard deviation. The low threshold (ρ_2), is set as $\rho_2 = \mu(max(t_0)) + \sigma(max(t_0))$. Finally, for the slope threshold δ_H, t_g is collected, which is the dataset of any bits that are of opposite value to the previous bit. Then δ_H is calculated as $\delta_H = \mu(max(t_g) - min(t_g))/T$.

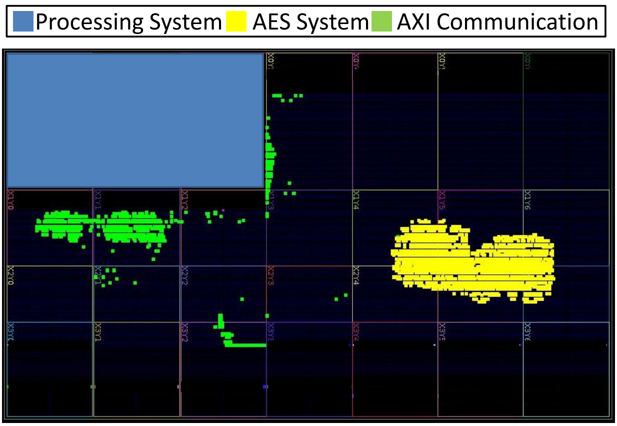


Figure 4.4: Floor plan of the implemented system on the ZCU102 UltraScale+ FPGA evaluation board

4.5.1.5 Evaluation Platform

Through-Fabric is run on a ZCU102 FPGA-MPSoC [197] which is supported by OP-TEE. The design is implemented using Vivado 2018. The MPSoC contains a quad-core ARM CORTEX A-53 processor and a PL with 200k LUTs. The AES accelerator is open source from [135]. It uses 11k LUTs and the PS to PL AXI interface uses 3.1k LUTs, at a clock frequency of 100MHz. The normal world operating system is a custom Linux distribution built using the PetaLinux SDK from Xilinx. Figure 4.4 shows the implemented system on the FPGA.

4.5.2 Covert-Hammer: Synchronizing Covert Communication from Multiple Tenants

In the usual case of covert communication, one party transmits, while other parties, usually only one, receive [82, 144]. In contrast, this chapter proposes a covert channel synchronization protocol for multiple tenants in hardware that goes beyond the traditional case where one is a transmitter and the other is a receiver. Rather, there are several malicious tenants, all of them are transmitting and receiving via the synchronization protocol.

4.5.2.1 System Overview, Assumptions and Goals

The system targeted by Covert-Hammer is illustrated in Figure 4.5. It is fully operating on the PL side with no involvement from the PS. It is a multi-tenant setup where at least two tenants are malicious and other tenants are benign. The setup also includes a static region that is designed by the CSP. Among others, the static region serves as a clock source for the different tenants and their communication interface with the outside world. This communication is essential for the tenants so they can get input data or communicate intermediate/final results for the customers renting the area on the FPGA-MPSoC. The static design also ensures that there is no intra-FPGA communication between the tenants.

Covert-Hammer assumes that each malicious tenant knows how many other malicious tenants are going to participate in the communication. However, other than having the same clock, the malicious tenants do not have any other means of synchronization nor can they know whether or not they reside on the same FPGA-MPSoC with other malicious tenants or the number of the malicious tenants on the FPGA-MPSoC. Therefore, the aim is to establish a synchronization protocol to coordinate the communication and allow them

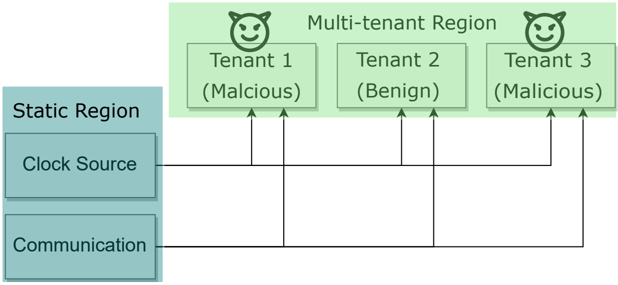


Figure 4.5: Expanded structural overview for the system targeted by Covert-Hammer. At least two malicious tenants coordinate their covert channel communication. The static region provides the needed infrastructure to all tenants.

to communicate reliably. The transmitting tenants uses seemingly-benign power wasters, same as those used for power-hammering, which makes it hard for the CSP to detect that they are malicious [16].

Moreover, the goal is to have a simple and robust synchronization protocol. The aim is to avoid using any sophisticated error correction or modulation schemes to ensure that the receiver is as lightweight as possible.

4.5.2.2 Covert Synchronization Protocol

The synchronization protocol is simple. It has n communication slots for a communication between n tenants with one slot per tenant. Each slot consists of 150 cycles at 100 MHz, the sending tenant generates a short voltage peak of 10 cycles, waits for 40 cycles, and repeats it once again. This is followed by staying silent for 50 cycles. Then a final voltage peak of 10 cycles is generated at the middle of the last 40 cycles.

This design is simple yet effective. Generating two voltage peaks (which are sensed by the listening malicious tenants) acts as synchronization. If the other malicious tenants do not exactly detect two peaks, they understand that this is not real communication but rather random noise, and no synchronization occurs. The silence for 50 cycles serves to increase the resilience to noise. It ensures that the two previous peaks are not just part of some ongoing noisy pattern detected mistakenly. This is important as the existence of several benign tenants may coincidentally produce such a pattern.

4.5.2.3 Design of the Malicious Tenants

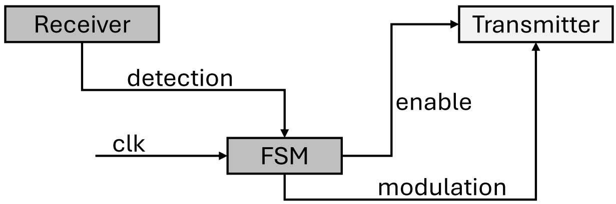


Figure 4.6: Figures of the design inside the two communication tenants with the connections to the state machine

The malicious tenant is able to: (i) send covert messages and (ii) receive covert messages. To perform these tasks, it uses the design from Figure 4.6. As mentioned above, the attacker sends peaks via the covert channel. Therefore, the attacker uses "power wasters" to send the peaks. Unlike previous works, the sender uses seemingly benign "power wasters" and successfully modulates them. By continuously enabling and then disabling the power wasters, the attacker causes sufficient power disturbances for communication, but not enough for power-hammering, i.e., no faults are injected to neighboring tenants or crash happens to the chip.

For receiving, it is necessary to detect the generated pattern reliably from the other tenants. As the tenants are using a covert channel for communication, there is no direct way to receive this pattern. Therefore, the tenants have to implement the receiver to detect the pattern generated by malicious tenants. This pattern causes a power disturbance that affects the timing of the circuits implemented on the FPGA. Therefore, the power disturbance can be detected by measuring the speed of a circuit. In order to detect these changes, the tenants implement Time to Digital Converters (TDCs) as a cascade of buffers based on the TDC design from [80]. Each buffer has a delay, and in case power disturbances exist, the delay of each buffer increases. When a a pulse is sent to the TDC and measure how long it needs to go through all buffers. When a high-power disturbance exists, the signal takes longer than expected to traverse all buffers. Hence, by monitoring the speed of signals traversing the TDC, tenants can reliably detect the peaks. In total, seven TDCs are used to enable reliable detection via majority voting. The TDCs are designed to be lightweight and cause no significant overhead.

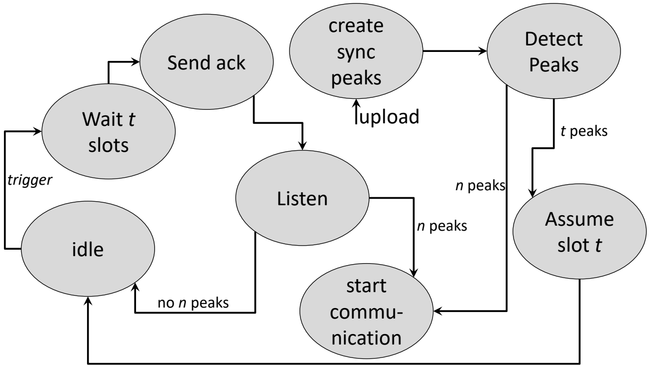


Figure 4.7: State machine controlling the malicious tenant.

Creating the covert/synchronization message, figuring out the proper communication slot, and receiving is controlled via the state machine shown in Figure 4.7. When a malicious tenant is uploaded, it does not know if other malicious tenants reside on the FPGA or not. Therefore, it directly starts sending the covert message. If it receives the correct number n messages from the n tenants, this means that all the needed tenants are there and the attack can start right away. However, if it receives less messages from number t it knows that not all the necessary tenants are there and waits for their upload. Based on the number t, it assigns itself the appropriate slot in upcoming communications. For example, if it is the first malicious tenant to be uploaded, it will receive $t = 0$ messages. Therefore, it will self-assign slot 0. Then when the next tenant comes, it will send its request for synchronization and will get only $t = 1$ messages and it will assume slot 1. This stacks nicely and in a conflict-free manner.

If the communication does not start right away, the finite state machine stays in the idle phase and continuously checks for the power disturbance via the TDCs. Once it detects the first peak, meaning that a new malicious tenant is uploaded, it waits for t slots and then sends its message in its correct slot. If all n messages are received, the communication is started. The communication is done in a cyclic manner, each malicious tenant sending covert messages during its slot. The malicious tenants can use this communication to perform any coordinated attacks needed, e.g., coordinated power hammering [4], inject faults in NNs [43], or use the TDCs to perform analysis on the benign tenants [122].

4.5.2.4 Validation on Hardware

Implementation on the ZCU102 Board: Similar to Through-Fabric the implementation of the Covert-Hammer system is on a Xilinx ZCU102 FPGA development board containing an UltraScale+ Zynq MPSoC. Synthesis, placement, and routing are performed using Vivado 2022.2 running on an Intel i9-12900 system with 64 GiB of memory. A complete run starting from synthesis to bitstream generation takes between 30 and 45 minutes, depending on the system and the configuration.

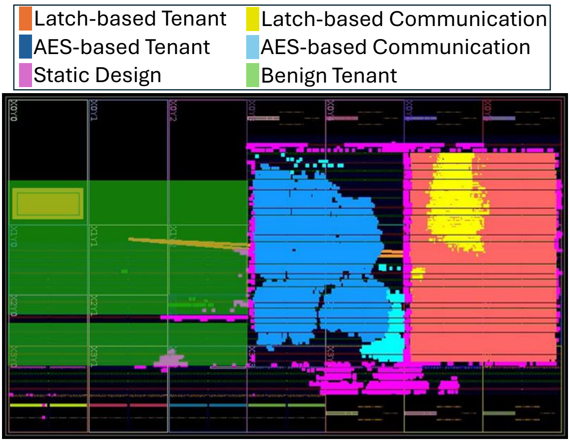


Figure 4.8: Floor-plan of the implemented system with the malicious tenants on the ZCU102 FPGA development board. In this floorplan, a latch-based transmitter and an AES-based transmitter are used for the malicious tenants.

The implemented system has three tenant slots alongside the static design as Figure 4.8 shows. Two slots are for malicious tenants, and the final one is for a benign tenant. The sizes for each tenant slot are given in Table 4.2. For the benign tenant,they are three different designs based on circuits from three benchmarks, ISCAS [44], Groundhog [104], and Berkeley [157]. The circuits used from each benchmark are stated in Table 4.1

Table 4.1: Circuits used from each benchmark as background applications to evaluate the robustness of the covert communication.

ISCAS	Groundhog	Berkeley
s208_1	GH09.B.1	ucsb_152_tap_fir−0
s420_1	GH09.B.2	ava−1
s526n	GH09.B.4	top_rs_decode−3
s9234_1	GH09.B.6	uoft_raytracer−3

For malicious tenants, they explore four different designs as the transmitter part. They use self-oscillating latch-based attacks from [126], AES-based attacks from [159], DES-based, and SHA-based attacks from [10, 16]. All these attacks can bypass any offline checks performed by commercial service providers [125]. Moreover, since three of the attacks are based on benign circuits (AES, DES, and SHA), it will be even harder to detect them by most of the offline detection tools.

Table 4.2: Size of tenant slots in three tenant setup. The malicious tenant slots are given more resources to be able to fine-tune the transmission.

Tenant	malicious 1	malicious 2	Victim
Size (LUTs)	69520	72048	34152
% of LUTs	27.8%	28.8%	13.7%

For the receiver part, the malicious tenants implement it using TDCs. The tenants use the same design from [80] Each TDC is designed as a long delay chain and uses 8 carry chains and 77 registers. The 7 TDCs increase the robustness of the communication and this design allows to have them with a low overhead. The receiver including TDCs uses only 0.6% of the available FPGA resources which is negligible.

Implementation on HACC Cloud Setup: The evaluation extends to Heterogeneous Accelerated Compute Cluster (HACC) [97]. HACC is a cloud setup hosted at several universities and run by AMD. It contains servers accelerated with cloud FPGA boards from Xilinx, e.g., Alveo U200. It natively supports only single-tenant designs. However, it is possible to have the FPGA partitioned into reconfigurable regions to mimic the multi-tenant scenario. The target is the Alveo U200 FPGAs in their infrastructure. It is significantly bigger than the ZCU102 FPGA with almost 10× the resources. Therefore, a CSP can divide the FPGA into 15 tenant regions, each with 6% of the FPGA resources (roughly the same size as 60% of the ZCU102) while keeping 10% for the static design.

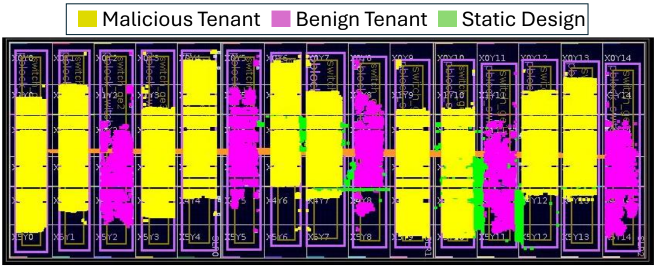


Figure 4.9: Floorplan of the implemented system on the Alveo U200 board on the HACC cloud. Up to ten malicious tenants residing on the same FPGA. All four transmitters were successfully uploaded to the FPGA.

Figure 4.9 shows the final design. Ten tenant regions are assigned to malicious tenants, while five are kept for victim tenants. For all four transmitter designs, they were able to be synthesized, generated the bitstream, and uploaded to the Alveo U200 within the HACC system. This proves that the offline checks performed by AMD in the cloud setup does not catch these stealthy attacks. Moreover, the AMD HACC shows its traffic (how many single-tenants are active at any moment) to its users. This traffic data is used to emulate a multi-tenant scenario where five tenants have to share an FPGA to evaluate whether having an attack from multiple tenants is possible or not.

4.6 Performance of the Covert Channels

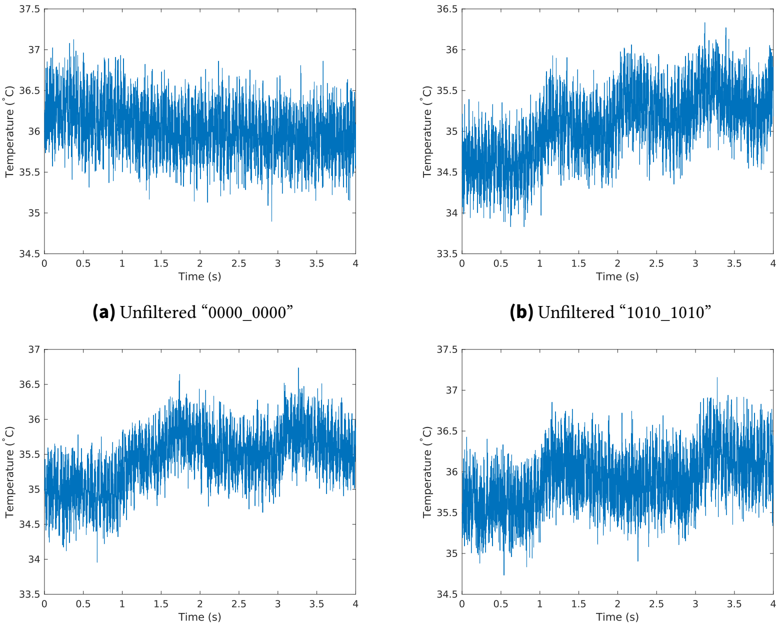
Based on the implemented systems, the performance of the proposed covert channels , Through-Fabric and Covert-Hammer, is evaluated under realistic conditions. The performance analysis covers both the transmitter and receiver efficiency, as well as the robustness of the channels in various scenarios. For

Through-Fabric, the evaluation id for the reliability of transmitting data using thermal modulation. For Covert-Hammer the effects of noise and number of communicating tenants on the robustness of the channel are evaluated.

4.6.1 Performance of Through-Fabric

Performance of the Through-Fabric covert channel is done by analyzing both the transmitter and receiver. The focus is on the AES accelerator's modulation impact on PL temperature and data transmission and the receiver's noise filtering and decoding accuracy.

4.6.1.1 Transmitter & Receiver



(a) Unfiltered "0000_0000"

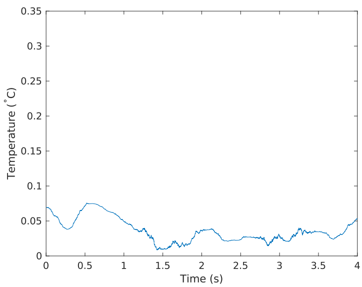
(b) Unfiltered "1010_1010"

(c) Unfiltered "1100_1000"

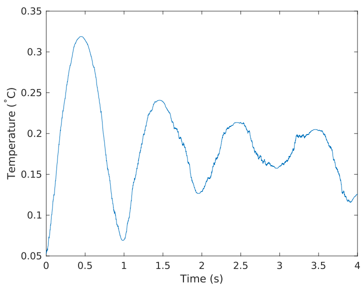
(d) Unfiltered "1000_1000"

Figure 4.10: Temperature of the PL based on sending the different 8-bit messages. Each message has a slightly different profile.

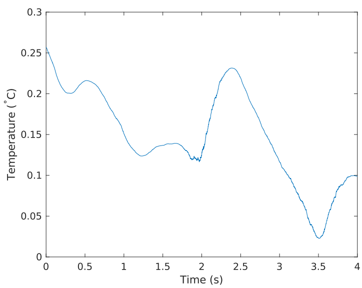
By modulating the usage of the AES accelerator, the transmitter creates changes in the temperature profile of the PL. This is apparent from Figure 4.10, as with sending each different packet of 8-bits the PL temperature profile is different. However, the temperature is affected not only by the usage of the accelerator. For example, the ambient temperature and different noise sources cause the temperature to fluctuate. Therefore, even with the different temperature profiles, it is necessary that the receiver applies further filtering.



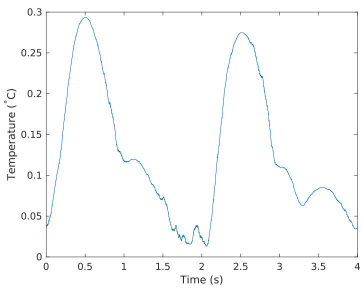
(a) Filtered "0000_0000"



(b) Filtered "1010_1010"



(c) Filtered "1100_1000"



(d) Filtered "1000_1000"

Figure 4.11: Data received by the receiver after applying filtering from algorithm 2. It can be seen that '0' bits and '1' bits are differentiable.

On the receiver side, after applying algorithm 2, the receiver is able to filter out all the effects from other sources. Figure 4.11 shows the same four packets after applying the filtering at the transmit frequency. For sending all zeros, the temperature stays at a very low level. For alternating between '0' and '1', the peaks are very distinct. When mixing zeros and ones in a series, they are still differentiable using the latter part of algorithm 2.

4.6.1.2 Channel metrics

In order to evaluate the effectiveness of the Thermal Covert Channel (TCC), the compromised Trusted Application (TA) is run. It performs several consecutive descriptions using the AES accelerator, in order to send 8,000 bits encoded in 8-bit packets on the FPGA-MPSoC board. Table 4.3 shows the result metrics for the new cross-world thermal covert channel from this experiment including bit error rate (BER), packet error rate (PER), and transmission rate. As it can be seen, the channel is effective under the tested scenario, achieving a transmission rate of 2 bps, which is on par with similar TCCs on non-CPU devices [86, 101]. Moreover, the attack was able to produce very low error rates, in a range similar to other state-of-the-art approaches for thermal covert channels (1-11%) [142]. Notably, since the attack is performed within the OP-TEE environment, its effectiveness shows how the isolation and data confidentially principles of the TEE have been effectively broken by the attack.

Table 4.3: Thermal covert channel evaluation metrics

Bits	Packets	Transmission rate (bps)	BER (%)	PER (%)
8000	1000	2	1.9	4.3

4.6.1.3 Comparison to state of the art

Other works exploited covert channels on FPGAs before. However, as Table 4.4 shows, the contribution is distinct from them in several ways. First, the contribution is the first to exploit a temperature-based covert channel between CPUs using the FPGA. The second distinction is that the contribution neither requires special malicious hardware for the transmitter nor for the receiver. Finally, none of the related works showed that they were able to break TEE on FPGAs.

Table 4.4: Comparison to related works. The contribution does not need any malicious hardware on the transmitter or receiver side.

Work	Requires Mal. (HW) Transmitter	Requires Mal. (HW) Receiver	Covert Channel	Break TEE
Through-Fabric	✗	✗	temperature	✓
Ref. [82]	✓	✓	voltage	✗
Ref. [42]	✓	✓	frequency	✗
Ref. [87]	✗	✓	voltage	✗
Ref. [78]	✓	✓	voltage	✗
Ref. [68]	✗	✓	frequency	✗
Ref. [79]	✗	✗	PCIe	✗
Ref. [77]	✓	✓	inter. wiring	✗

4.6.2 Performance of Covert-Hammer

Subsequently, the focus shifts to evaluating the performance of Covert-Hammer across the two implemented systems. The emphasis is placed on the efficacy of the synchronization process rather than on the communication itself. This focus is based on the notion that once synchronization is achieved, each tenant involved in communication reliably secures their designated time slot. Consequently, even in the presence of message errors, the overall communication process remains operational.

4.6.2.1 Robustness of the Covert Channel on ZCU102

For coordinated covert communications, the feature evaluated is the robustness of the covert channel. This metric is of high importance because if the covert channel is noisy and not robust, then the attacking tenants will not be able to synchronize and the attack will fail. To evaluate the robustness the used metric is the correct packet rate.

Figure 4.12 shows the correct packet rate of the channel. The different benchmarks did not cause any significant change in the correct packet rate. Therefore, the detailed analysis for them is excluded from the subsequent results as it would be redundant. For the transmitter tenants themselves, not all of them

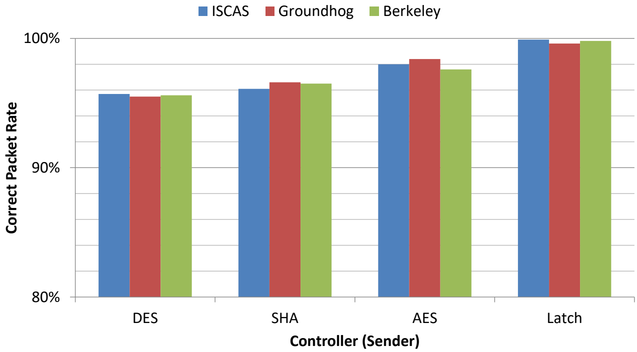


Figure 4.12: The correct packet rate of the covert channel. Using the different attackers as the transmitter with noise from benign benchmarks shows that the covert channel is robust. The correct packet rate never drops below 95%.

are as robust as each other. When acting as a transmitter, the latch-based attacker has a correct packet rate higher than 99.5%. AES and SHA-based attackers have correct packet rates of around 98% and 96% respectively. The lowest correct packet rate is the one from the DES-based attackers of 95%, which is still significantly robust.

4.6.2.2 Evaluation on the AMD HACC Cloud Setup

The evaluation is extended to the HACC setup from AMD. The evaluation focuses the chances of uploading several malicious tenants to the same FPGA and the success of the covert channel communication between several tenants.

Success of having Multiple Malicious co-Tenants: HACC does not offer multi-tenancy. However, based on the recorded traffic of the cloud setup, it is possible to emulate such a scenario. In one location, HACC has 50 FPGAs. The traffic is recorded and instead of assigning each tenant an FPGA, the assumption is that five tenants will share an FPGA. The assumption that the CSP will cluster tenants on the same FPGA whenever possible.

The average usage time of a user during the recorded period was one hour and the maximum allowed was five hours. Each malicious tenant reserves the maximum period. Moreover, the assumption is that the malicious tenants will be uploaded separated by 15 minutes. The attacker will flood a system with upload requests until the FPGAs are all full.

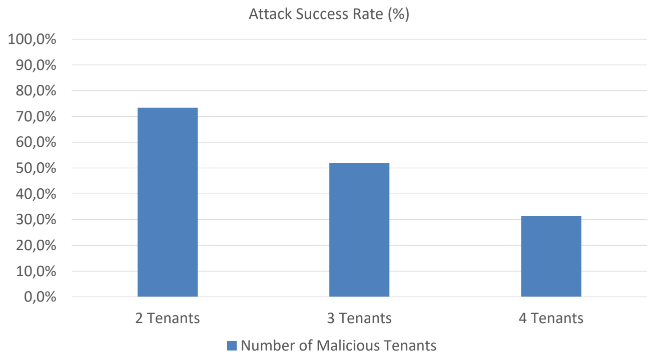


Figure 4.13: Attack success rate of coordinated malicious tenants on the Alveo U200 at the HACC cloud setup.

Figure 4.13 shows the success rate of the attack. An attack is successful if the minimum number of tenants needed is uploaded to an FPGA while having at least one victim residing with them. The success rate gets lower as the number of malicious tenants increases. If the attack needs two tenants to be successful, it has a chance of success 70%, while for four tenants, the chance of success of uploading the tenants together is around 30%. Note that these results are highly dependent on the clustering policy of the CSP. If a different policy is used, e.g., considering the tenant's period of usage, or differently sized tenants are needed, the results would change.

Success of Establishing Covert Communication: Next, the covert communication in on the Alveo board is evaluated using the HACC setup shown in Figure 4.9. The number of communicating tenants range from two to ten. For simplicity, all the malicious tenants use the same type of power wasters. Communication degrades significantly when the number of tenants communicating increases. This makes sense as if only one of the tenants fails in the communication even once and the whole system fails because the slot assignment fails. Moreover, based on the type of the power-waster that launches the communication, the failures increase. For DES-based, the communications fail most significantly, while for latch-based, the communication is more stable and successful.

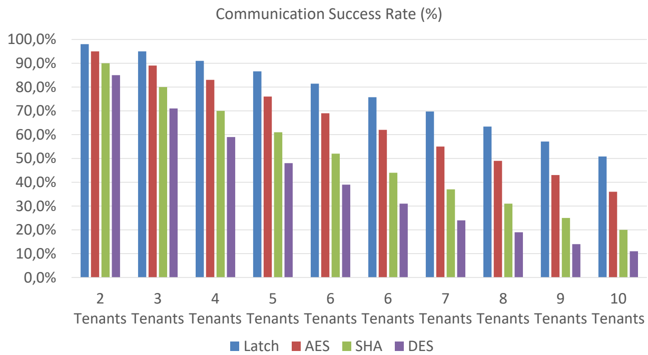


Figure 4.14: Communication success rate on the Alveo U200, with each extra tenant the ability to establish the communication drops significantly.

4.7 Proposed Countermeasures

In addition to introducing novel covert channel attacks, this chapter also proposes countermeasures to mitigate them. The two covert channel attacks may include components within both hardware and software, or may be confined exclusively to hardware. Therefore, the chapter proposes countermeasures at both the hardware and software levels.

4.7.1 Hardware-based Countermeasure

The first proposal is to use Ring Oscillators (ROs) as a hardware-based countermeasure. Ring oscillators are implemented as a chain of Runtime-Configurable ROs (RCROs) as shown in Figure 4.15. The building blocks of the RCRO are the Configrubale Inverters (CIs). CIs have two types: Simple CIs (SCIs) and Complex CIs (CCIs). A chain of RCROs consists of 65 CIs (13 CCIs and 52 SCIs). This mix of CIs helps to generate random noise. For SCIs, if the *sel* bit

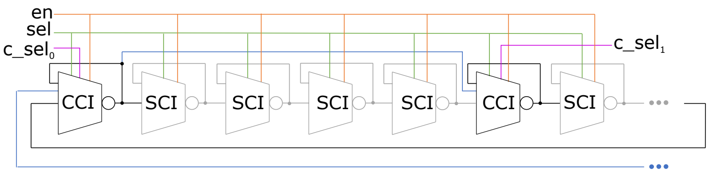


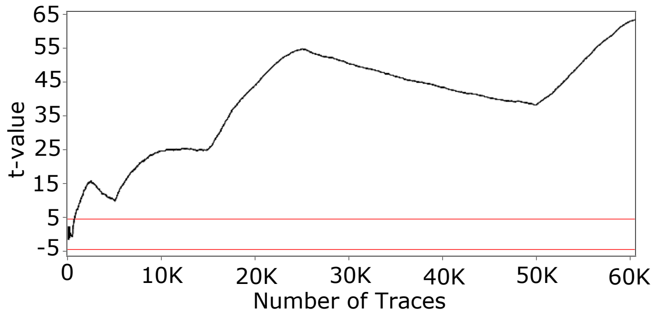
Figure 4.15: Runtime-Configurable Ring Oscillator.

is set, the output of the previous inverter in the chain is used as the input, rather than its own output being applied to the input as immediate feedback. For CCIs, not only the output signal of the previous SCI or its own feedback can be used as an input signal, but also the output signal of the previous CCI in the chain. For this additional case, a second *c_sel* control bit is required for each CCI. If the *c_sel* control bit is not set, the CCI behaves like an SCI. However, if both the *sel* control bit and the *c_sel* control bit are set, the output signal of the preceding CCI in the chain is selected as the input signal and the SCIs that are between the two CCIs are skipped to reduce the length of the chain by four inverters. Since the *c_sel* control bit can be set individually for each CCI, it is possible to change the chain length at any time by changing the value of *c_sel*. Depending on the configuration, the chain can have a variable length between 13 (all SCIs are skipped) and 65 CIs (all SCIs are used) with a step size of 4 CIs.

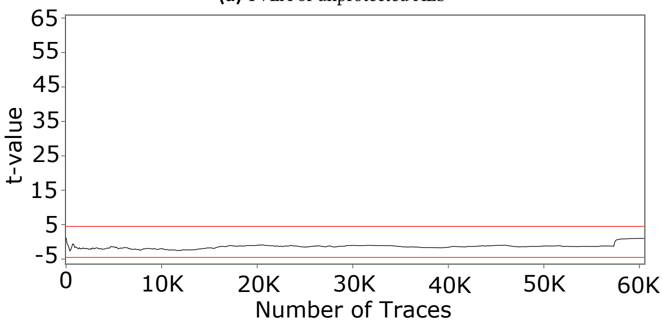
The behavior of *c_sel* and *sel* is controlled by a central control module. When the countermeasure is activated, the control module obtains a random number R from an Random Number Generator (RNG) and then enables all CIs for R clock cycles, and then disables them for R clock cycles. Then a new number R is generated and the process is repeated. This generated signal is forwarded by the control module to the ring oscillators as an enable signal so that these are always active or inactive for a random number of cycles. This generates random noise in the time domain. To generate random noise in the power domain, the *c_sel* signals for each CCI are also controlled by the RNG, Thus, a random number of SCIs is skipped at each run to vary both the power consumption and the frequency of the noise. Consequently, the noise does not contain any regular patterns that could be filtered out. This is in contrast to using all RCROs all the time, which would have a regular pattern.

The effectiveness of the countermeasure is evaluated using the Test Vector Leakage Assessment (TVLA) [36]. It uses two sets of power traces, the first set is generated by always choosing the same fixed input data. In the second set, the plain texts are chosen randomly. Then a Welch's t-test is applied to the two data sets to determine whether they differ significantly from each other. If the t-value stays in the range of $-4.5 < t < 4.5$ the system is considered secure, otherwise leakage exists [36].

The TVLA method is run on the system, once with the countermeasure activated and another with the countermeasure deactivated. For each, the evaluation is done by collecting 120K traces, 60K are with fixed input, and 60K with random input. Based on the traces, the t-value is calculated. Figure 4.16 shows the TVLA results. For the unprotected case (shown in Figure 4.16a) the t-value rapidly grows out of the secure range.



(a) TVLA of unprotected AES



(b) TVLA of protected AES

Figure 4.16: Test vector leakage assessment (TVLA) results

4.7.2 Software-based countermeasure

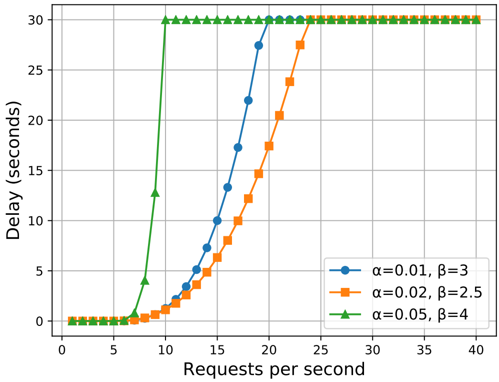


Figure 4.17: Delay added to the acceleration request for different number of requests per second from the same application.

For a software-based countermeasure, the idea is to implement an application-specific delay countermeasure. Similar to the classical use for memory contention on spinning processors [35], an increasing delay penalty can be applied to a TA using the accelerator with an abnormal frequency of requests. In this case, after the application has invoked the API method to use the accelerator, before queuing the request, the method checks the number of times this particular application has requested the accelerator in a period of time. If the number of times is greater than the threshold for normal use, then a delay is added to the request. Since the transmitter requires to use the accelerator multiple times to encode the bits as temperature variations, each use would be increasingly longer, disturbing the timings of the channel. A simple yet flexible function to obtain the delay for a number of requests per second (n) can be described as follows:

$$\text{delay}(n) = \min\left(\alpha \cdot (n - \text{threshold})^{\beta},\ \text{max\_delay}\right) \tag{4.1}$$

where α is a scaling factor that determines the severity of the penalty, β determines how sharply the penalty increases, and the threshold is the number of requests considered as normal use. *max_delay* limit is as an upper boundary for the delay. Figure 4.17 shows the delay for different configurations of α

and β, with a threshold of 5 requests per second and a max_delay of 30 seconds. As it can seen, this function can be used to enforce different degrees of delay penalty depending on the frequency of the requests to the particular accelerator.

4.8 Summary

Covert channel attacks pose a significant threat as they exploit the shared nature of multi-tenant FPGA-MPSoC resources, allowing colluding tenants to leak data. This chapter explores two such covert channel attacks, one based on power consumption and the other on thermal emissions, and propose countermeasures to mitigate these threats. It demonstrates that the threat of covert channels is more complex than what has been shown previously in the literature and can break TEE on FPGA-MPSoCs via the use of benign accelerators. Moreover, it demonstrates how a covert communication can be established in a multi-party manner on FPGA-MPSoCs. It also proposes a noise-generating countermeasure with random, variable frequencies using ring oscillator chains of different lengths on the hardware level and a timing-based countermeasure on the software level.

5 Data Leakage Mitigation in Cloud Systems Using FPGA-Accelerated Homomorphic Encryption

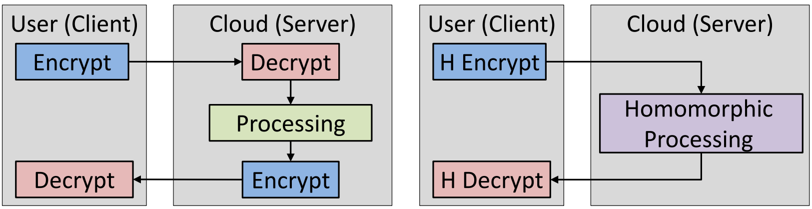
This chapter tries to fully eliminate the threat of data leakage from accelerated cloud systems. The landscape of computing infrastructure is changing with the adoption of cloud computing. It allows organizations to use on-demand services, freeing them from the burden of owning and maintaining their computing infrastructure. However, this transformation comes with challenges, mainly the escalating concerns over privacy and security. These concerns stem from the fact that data processing has to be done in plaintext within the infrastructure of the CSP. However, if the CSP has a vulnerable security, the users' data can be breached [191] and such attacks are not uncommon [107, 129].

In response to these concerns, HE emerges as a robust solution. It allows processing directly on encrypted data, i.e., without even providing the cloud servers the keys to decrypt the data first. HE ensures that sensitive information remains confidential throughout computational processes. HE applications span various domains, e.g., the healthcare sector, artificial intelligence, electronic voting systems, financial data processing, and encrypted search engines [21, 103, 109, 155]. Despite the promise of HE applications, practical implementation encounters obstacles, most notably the substantial computational and memory overhead of homomorphic operations. On the algorithmic side, there are mathematical optimizations. The most prominent of them is Fast Fully Homomorphic Encryption over the Torus (TFHE), which is post-quantum secure [54].

This chapter is based on contributions from [8, 9].

5.1 Motivational Example

Consider a scenario in which sensitive financial data must be processed within a public cloud system. Using classical approaches, this data would be sent to the server in plaintext, exposing it to potential breaches. Figures 5.1a and 5.1b compare this traditional method with a homomorphic approach. In the classical case, as shown in Figure 5.1a, the plaintext data is vulnerable once it reaches the server. In contrast, HE allows the data to remain encrypted throughout the computation process, as depicted in Figure 5.1b, ensuring that even if the server is compromised, the data remains secure. This example underscores the necessity of efficient HE implementations, particularly for applications where data confidentiality is crucial.



(a) Classical case **(b)** Homomorphic case

Figure 5.1: Client/Server computation flow. In the classical case, plaintext data is processed on the server side which might lead to data breach. In contrast, When Homomorphic Encryption is used, data is processed on the server side while encrypted. Thus, no data breach is possible.

5.2 Problem Statement

HE is both computationally and memory bound [192] as one step of HE typically involves the processing of MiBs of data which limits its usage in public cloud systems. HBM, as integrated in newer generations of FPGAs [184], can be used to resolve such memory bottlenecks when FaaS is provided by the CSP. Previous works tackled using HBM-enabled FPGAs to accelerate approximate versions of HE [192, 193]. However, such accelerators are mainly suitable for ML computations, but not suitable for accuracy-critical applications. This chapter implements HBMorphic an accurate accelerator for TFHE on an HBM-enabled FPGA to speed up its operation.

5.3 Contributions

This chapter contributions are summarized as follows:

- HBMorphic is the first accelerator to address the memory bottlenecks of fully accurate TFHE using HBM.
- HBMorphic carefully analyzes the data access pattern and maps the independently accessed data across memory channels to fully utilize the HBM bandwidth.
- To speed up computations in TFHE, HBMorphic uses a fast, parameterizable, recursive multiplier (using the Karatsuba algorithm) that can easily scale to various TFHE accelerator implementations and even offer a trade-off between resource utilization and performance.

5.4 Previous Homomorphic Encryption (HE) accelerators

Previous HE accelerators exist and can be grouped into three groups. The first group contains accelerators targeting TFHE [45, 74, 194]. They do not focus on solving the memory bottleneck, but work under the assumption that the data will be available when needed. They apply Fast Fourier Transform (FFT) or Number Transform Theory (NTT), which lower the multiplications overhead at the cost of reduced accuracy due to numerical errors [170], i.e., they are not suitable for applications requiring full accuracy.

The second group contains HE accelerators that use Karatsuba-based multipliers [69, 113, 140, 141, 179]. They either target SHE or implement a generic multiplier that can be used for HE. In both cases, the multiplier is less complex than the one implemented in this chapter and is not suitable for TFHE acceleration.

The final group contains HE accelerators that use HBM [192, 193]. Both accelerators target the Cheon-Kim-Kim-Song (CKKS) scheme that can work in an FHE-like mode. However, CKKS uses approximate arithmetic and therefore cannot be used for processing that requires the highest accuracy like health-related algorithms, electronic voting, and financial data processing.

5.5 HBMorphic's Design & Implementation

The aim is to design HBMorphic as an accelerator that will speed up the bottleneck of PBS. HBMorphic uses accurate multipliers, so that it can be used for all applications. Moreover, it uses HBM to reap the parallelism offered by 3D memories and reduce memory contention.

5.5.1 System Overview

HBMorphic is built as a full system on the FPGA to accelerate the PBS step of TFHE and evaluate the benefit of using HBM. Figure 5.2 shows the components of the system. For accelerating PBS a Karatsuba-based accelerator is implemented. The full system contains a MicroBlaze softcore to (i) communicate with the outer world, e.g., loading the data over Ethernet to memory, (ii) give the Karatsuba-based accelerator the addresses of the data, and (iii) communicate back the results. The system has an HBM interface to read and write the data, it is accessible both via MicroBlaze (to initialize the data) and via the accelerator to use the data. Similarly, the system has interfaces for two off-chip DRAMs, which is used for comparison purposes between data access via HBM and DRAM.

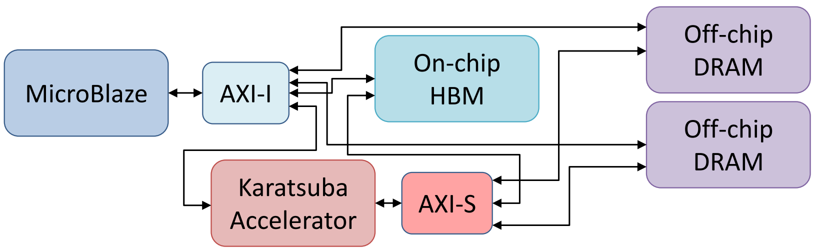


Figure 5.2: System overview, MicroBlaze initializes the data, and the Karatsuba-based accelerator accesses the data from HBM and accelerates the PBS. The off-chip DRAM interfaces are used to evaluate the benefit of HBM.

For communication between the components of the system uses Advanced eXtensible Interface (AXI). The different AXI connection types allow optimization of the connections for different purposes. The AXI Smartconnects (AXI-S in Figure 5.2) are optimized for speed, which is important to not slow down the AXI bandwidth. The accelerator use them to communicate with the different memory components to get the highest throughput. In contrast, MicroBlaze uses the AXI Interconnect (AXI-I in Figure 5.2) for communication. It cannot be optimized for speed, but this is not a problem as the MicroBlaze will not even saturate a slow AXI connection. This allows the AXI Interconnect to be much smaller than the AXI Smartconnects.

5.5.2 Custom HBM Interface

The accelerator is implemented on a VCU128 board containing an UltraScale+ FPGA including an HBM [184]. The HBM has 32 Pseudo Channels (PCs), each of size 256 MiB. The chip includes a generic interface containing an ASIC interconnect between the PCs and the FPGA. It can be bypassed by implementing a custom interface to get higher throughput. The design would work the same on other FPGAs or FPGA boards that include HBM with similar specifications.

HBMorphic implements its own custom HBM interface to obtain the highest throughput possible. Figure 5.3 shows the interface for the TFHE-777 accelerator that performs the external product step of PBS. For one PBS, a total of 25 MiB is read. The data needed for the external product is packed in 777 3D arrays, each of the size $\{4, 4, 512\}$ of 32 bit words. The interface distributes the 32 PCs evenly across the 3D arrays to minimize memory contention. Each PC is used to read only 256 words, distributing the load symmetrically.

To access each of the PCs independently, HBMorphic creates 32 AXI memory interfaces, each of which is capable of managing the read and write requests for one PC. The Karatsuba-based accelerator is contained in an AXI wrapper with 32 independent ports. Each PC has a data output width of 64 bits.

The frequency of the HBM is higher than the frequency of the logic implemented on the FPGA. Therefore, each port has a bitwidth of 512 bit to pack four words from each PC together. The 256 words are read sequentially over 16 read operations. To amortize the latency as much as possible HBMorphic uses double buffering. Therefore, the data is already available immediately when it is needed. Note that the memory interface does not focus on only having parallel instances of the AXI memory modules but rather on partitioning the memory in channel granularity. This granularity helps in distributing the data reading load equally across all channels and reaching the highest possible bandwidth.

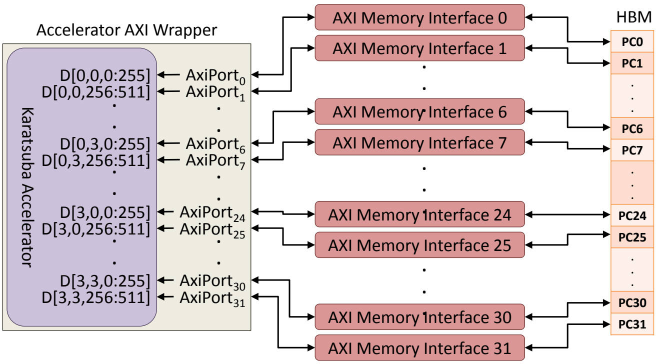


Figure 5.3: The HBM interface for TFHE-777. The 32 Pseudo Channels are divided equally to get the data packed over a 3D array. For each of the 777 multiplications, 256 32 bit words are read.

5.5.3 Accelerating the External Product of PBS

The external product allows a ciphertext multiplication whose result is an encryption of the product of plaintexts. Note that the mathematics of TFHE is performed over polynomials of size n. It is called 'external' because it combines the homomorphic ciphertext with the external new bootstrapping key in a homomorphic ciphertext form. It is defined as follows

$$A_i = (BK_i \cdot ((A_{i-1} * C_i) - A_{i-1})) + A_{i-1} \tag{5.1}$$

The computation goes from $i = 0$ to $i = n$. A_i is the ith coefficient of the new ciphertext, BK is the bootstrapping key in ciphertext form, C is the homomorphic ciphertext, A_{-1} is initialized to V, which is the optional lookup table that can be performed during the PBS.

A naive polynomial multiplication requires n^2 scalar multiplications. The Karatsuba multiplier [113] reduces the needed scalar multiplications to $n^{1.58}$, which makes a significant reduction for large values of n. For example, for $n = 512$, the number of scalar multiplications reduces from 262,144 to 19,683.

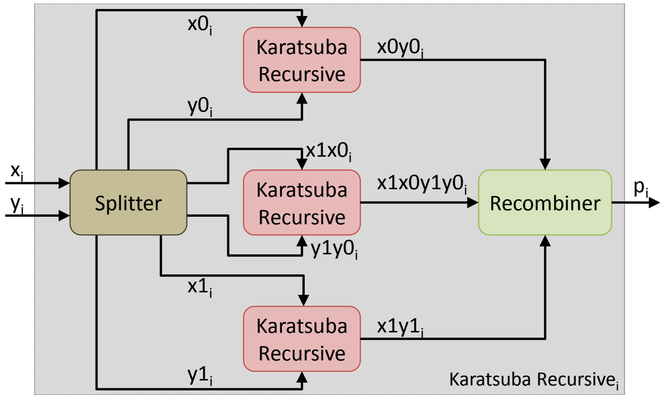


Figure 5.4: Recursive Karatsuba Multiplier. At level i, each recursive unit contains three sub-recursive units alongside a splitter to generate the sub-signals to each recursive unit and a recombiner that produces the final product of the level. The 0 denotes the lower half of the polynomial and the 1 denotes the upper half of the polynomial.

The Karatsuba multiplier is implemented using SystemVerilog which supports recursive calls of HDL designs. Figure 5.4 shows the design of the Karatsuba multiplier. Each unit includes a splitter, recombiner, and three recursive instances of the Karatsuba multiplier. The number of polynomials given to each recursive unit, e.g., $x1_i$, equals half the number of polynomials of the main input x_i. The accelerator is parameterizable. It takes two main parameters for input; the first is the polynomial size and the second is the stopping size. By default, the Karatsuba algorithm requires implementing log_2(polynomial size) levels of the Karatsuba recursive unit. In the final stage, it will implement the actual multipliers. Hence, it gets a fully parallel implementation. However, as the polynomial can be of high value, the 'stopping size' as a second parameter is needed. It switches from a parallel implementation to a sequential implementation at the configured level. In the end, the number of levels implemented would be log_2(polynomial size/stopping size). This makes the accelerator very easy to fine-tune, based on the resource constraints of any FPGA on which it would be implemented.

Algorithm 3 shows how the recursive Karatsuba works as implemented in SystemVerilog. It starts by splitting the input numbers, x and y, into two halves each: $x0$, $x1$, $y0$, and $y1$. Then it recursively computes the products $x0 \times y0$, $(x0+x1) \times (y0+y1)$, and $x1 \times y1$. These products are combined to obtain

Algorithm 3: The Recursive Hardware Multiplication

Input: x_i, y_i: Input arrays of scalars
Output: $product_o$: Output array of products
procedure KaratsubaMultiplication(x_i, y_i, $product_o$)
initialize $x0\_y0$, $x1x0\_y1y0$, $x1\_y1$
split x_i and y_i into halves $x0$, $x1$, $y0$, $y1$
if InputPolynomialSize is StoppingSize
    perform Karatsuba multiplication for the 3 halves
    $x0\_y0 \leftarrow$ KaratsubaMul($x0$, $y0$)
    $x1x0\_y1y0 \leftarrow$ KaratsubaMul($x1 + x0$, $y1 + y0$)
    $x1\_y1 \leftarrow$ KaratsubaMul($x1$, $y1$)
else
    recursively call KaratsubaMultiplication for each half
    $x0\_y0 \leftarrow$ KaratsubaMultiplication($x0$, $y0$)
    $x1x0\_y1y0 \leftarrow$ KaratsubaMultiplication($x1 + x0$, $y1 + y0$)
    $x1\_y1 \leftarrow$ KaratsubaMultiplication($x1$, $y1$)
recombine the results of the recursive calls
$product_o \leftarrow$ Recombine($x0\_y0$, $x1x0\_y1y0$, $x1\_y1$)
return $product_o$
end procedure
procedure KaratsubaMul(a, b)
initialize p
if a or b is a single scalar
    return $a \times b$
else
    split a and b into halves $a0$, $a1$, $b0$, $b1$
    recursively call KaratsubaMul for each pair of halves
    $p \leftarrow$ KaratsubaMul($a0$, $b0$)
    $p \leftarrow p+$ KaratsubaMul($a1$, $b1$)
    $p \leftarrow p+$ (($KaratsubaMul$(a0 + a1,b0 + b1) - p)
    -$KaratsubaMul$(a0,b0)−KaratsubaMul($a1$, $b1$))
    return p
end procedure

the final result. When the input size reaches the stopping size, the algorithm switches to simple multiplication instead of further recursion. The Karatsuba Multiplication procedure takes two input arrays, xi and yi, and computes their product, storing the result in the output array product. Initializes the arrays to hold intermediate results and splits the input arrays into halves. Depending on the input size, it performs either simple multiplication or calls itself recursively for each half. The KaratsubaMul procedure is a helper function that performs the actual multiplication of two arrays. Recursively splits the input arrays and computes the products of smaller halves. These products are then combined to obtain the final result.

5.5.4 Accelerator Implementation

Table 5.1: TFHE Parameters used for the accelerator implementation. Two variants are implemented, TFHE-777 and TFHE-50 based on Rust code from [53].

Parameter	TFHE-777	TFHE-50
***Learning with error* dimension**	777	50
***Generalized learning with error* dimension**	3	2
Polynomial size	512	64
Bootstrapping base log	18	18
Scalar size	32	32
Bootstrapping level	1	1

For the accelerator, two different variants of TFHE are implemented: TFHE-777 and TFHE-50. Both versions are equivalent on the algorithmic level. The main difference is the volume of the processed data, and subsequently, the overhead from processing each of them. The key size of TFHE-50 is smaller than TFHE-777 and therefore, TFHE-50 is not considered fully secure [53].

The complete parameters for each TFHE variant are shown in Table 5.1. The data for the accelerators are packed in 4D arrays of dimensions [*learning with error* dimension, *generalized learning with error* dimension+1, *generalized learning with error* dimension+1, Polynomial size] of 32 bit words. The multiplication is done *learning with error* dimension times over the sub-3D arrays of the data.

The accelerator is parameterized to build log_2(polynomial size/stopping size) levels of the Karatsuba algorithm. Ideally, the stopping size should to be equal

Algorithm 4: TFHE-777 Accelerator Operation

```
Input: A: 4D arrays containing cipher text and key
Result: R: new cipher text
init(HBM) ; /* MicroBlaze loads arrays to HBM */
start(load(A[0])) ; /* start loading first array using the 32 PCs of HBM */
for i in 777 do
    Wait for load(A[i]) ; /* Wait until the data is loaded */
    if i < 776 then
        start(load(A[i+1])) ; /* start loading the data of the next array from the 32 PCs */
    end
    [x, y] ← first_split(A_i, R_{i−1}) ; /* Calculate the first two values for the multiplier */
    init(Karatsuba_r(x,y,polynomial_size)) ; /* Load the data to the recursive Karatsuba multiplier */
    Run algorithm 3
    R[i] ← recombine(p_r) ; /* Calculate the ciphertext component over all the recursively calculated products */
end
return R;
```

to 1, i.e., everything is parallel. The FPGA from the VCU128 board has 6840 Digital Signal Processing (DSP) slices, each of which can be used as a multiplier. For TFHE-50, the polynomial size is 64. This means that there are 6 levels of the Karatsuba recursive unit with an overall of 729 multipliers. This is fine as they can all be mapped to the DSP slices. However, for the TFHE-777, with a polynomial size of 512, it would need to build 9 levels that would use 19,683 multipliers. This is much more than the available DSP slices. Therefore, it uses a stopping size of 2, which reduces the number of needed DSP slices to 6,561. This fits on the FPGA, leaving a couple of hundred DSP slices for any other computations needed. The downside is that the multiplier is now 3× slower than its maximum theoretical throughput if all levels were parallel. Algorithm 4 shows the steps needed to run the implemented accelerator for TFHE-777. First, the MicroBlaze initializes and makes sure that the data is written to HBM. Then the accelerator starts loading the first of the 777 3D arrays, and once all data is there, the accelerator starts executing. In parallel to accelerator execution, the next array is loaded from HBM to amortize the delays. This double buffering is done into Block RAM (BRAM) on the FPGA. Moreover, the intermediate result between one step and the next is also written to BRAM because it is used to calculate the next value.

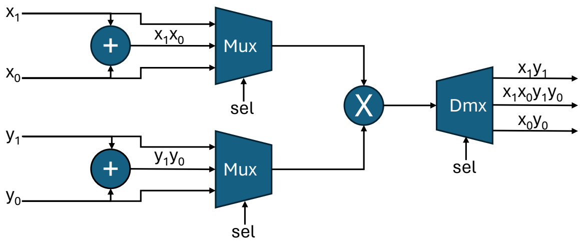


Figure 5.5: Design of the final multiplier in the recursive Karatsuba for TFHE-777. Not enough DSPs are available, therefore, 3 multiplications share the same multiplier.

Figure 5.5 shows the design of the final stage of the multiplier when accelerating TFHE-777. The last three multiplications use the same multiplier. Two adders that are implemented in LUTs prepare the x_1x_0 and the y_1y_0 terms. Then, via muxes and demuxes, the products are produced. The selection line of the muxes is controlled via a finite state machine. It has 4 values, '0' the muxes are turned off, they get constant 0 as input and the equivalent output from the demux is disconnected. Then the values '1', '2', and '3' control the output of the multiplications of the terms x_0y_0, $x_1x_0y_1y_0$, and x_1y_1 respectively.

5.6 Performance of HBMorphic

HBMorphic is evaluated on a VCU128 board [184], using Vivado 2022.2. The maximum frequency achieved by the design is 75 MHz. To validate the results, the ciphertexts and bootstrapping keys are generated using the rust code from [53].

To be able to evaluate several metrics, the implementation of each of the TFHE-50 and TFHE-777 use once the Karatsuba Multiplier and once using a normal multiplier. Moreover, for the memory interface, three different variations are

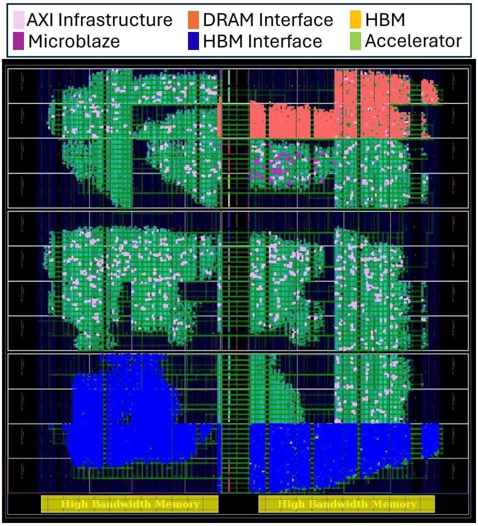
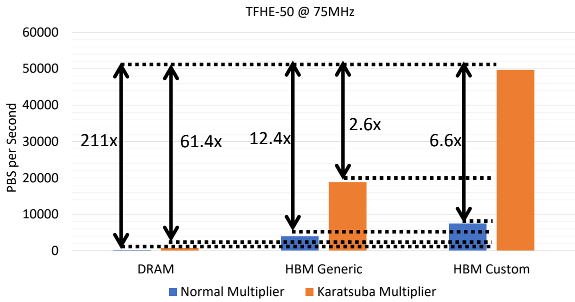


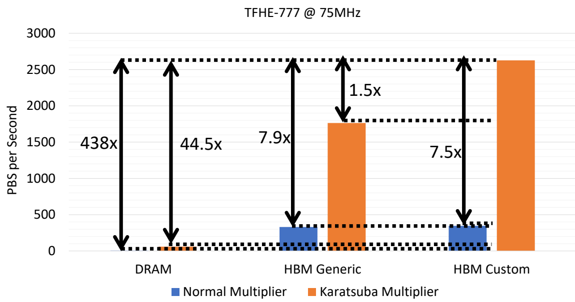
Figure 5.6: HBMorphic floorplan on the VCU128 board taken from the synthesized design on Vivado 2022.2. The accelerator implemented accelerates TFHE-50. The vertical stacking of HBM is not visible in this 2D representation.

used. The first one uses the off-chip DRAM to have a baseline without any HBM usage. The second one uses the on-chip HBM with the generic interface from Xilinx [186]. The final one uses the proposed custom HBM interface. The same optimization is done for both the custom and generic HBM interfaces. The HBM interface enables request and coherency in reordering, look ahead pre-charge and activate, and data is set to be accessed in a linear mode. For the generic interface, the address space considers the whole memory as one address space and let the storage of the data be done over all channels. For the custom interface, each channel is considered as a separate memory to be read and written independently from each other.

Figure 5.6 shows the synthesis and implementation results for HBMorphic when accelerating TFHE-50 using the Karatsuba multiplier and the custom HBM interface. The exact resource utilization is listed in Table 5.2 but the figure gives an initial idea. As expected, the Karatsuba multiplier takes up the majority of the design, with AXI bus components scattered over the system. The HBM interface is of significant size as well and is implemented very close



(a) Maximum programmable bootstrappings per second achievable using the accelerator for TFHE-50



(b) Maximum programmable bootstrappings per second achievable using the accelerator for TFHE-777

Figure 5.7: Maximum programmable bootstrappings per second for both variants of TFHE using the different multipliers and memory interfaces.

to the HBM to handle the data right away. The DRAM interface is also of a significant size, while the MicroBlaze is almost invisible.

5.6.1 Performance of the Accelerator

The first metric evaluated is how many PBSs per second can be performed using the accelerator. To be able to evaluate the benefit of the Karatsuba multiplier and the custom HBM interface, for each TFHE variant, the evaluation uses six different combinations between the multiplier and the memory interface with the Karatsuba multiplier and the custom HBM interface being the solution HBMorphic.

Figure 5.7 shows the performance of TFHE. For TFHE-50, using the off-chip DRAM achieves the lowest number of PBS as expected. The execution is dominated by the memory accesses. The Karatsuba multiplier is effective in comparison to the normal multiplier, reaching on average 4.9× improvement in PBS per second when using the same memory interface. Even using the Karatsuba multiplier with the generic HBM interface is performing 4.7× better than the normal multiplier using the custom HBM interface. The custom HBM interface in general outperforms the generic interface. Using HBMorphic, i.e., the custom interface and the Karatsuba multiplier can perform 49715 PBS per second in comparison to 18835 PBS per second using the generic interface. In comparison to the DRAM baseline, HBMorphic has a 211× speedup.

For the TFHE-777 the same trends generally hold with a few differences. First, since the data and computations are significantly more, the PBS per second achieved using the Karatsuba multiplier and the custom HBM interface (HBMorphic) is 2627. Second, the difference between using custom and generic interfaces is smaller as the Karatsuba multiplier achieves 1763 PBS using the generic interface. Third, using the custom or the generic interface makes little difference for the normal multiplier as it is now dominated by the computation of the multiplication, not the data loading. This makes sense as the number of multiplications grows from roughly 64^2 to roughly 512^2. This huge increase in the multiplications makes HBMorphic have a higher speedup in comparison to the DRAM baseline of 438×.

5.6.2 HBM Bandwidth Utilization

Next, the HBM bandwidth utilization of the accelerator is evaluated. For this evaluation, the combinations using DRAM are omitted as they have no HBM utilization. The focus is only on the TFHE-777 as it is the more secure and also the more data-intensive of the two variants implemented, leaving only with 4 combinations. The accelerator's memory access pattern of the data is tracked for 10 seconds using the HBM monitor from Xilinx [184].

The theoretical maximum bandwidth of the HBM on the VCU128 board is 460 GiB/s. However, based on the Xilinx documentation, practically the limit is 90% of this theoretical bandwidth [186]. For the normal multiplier, it

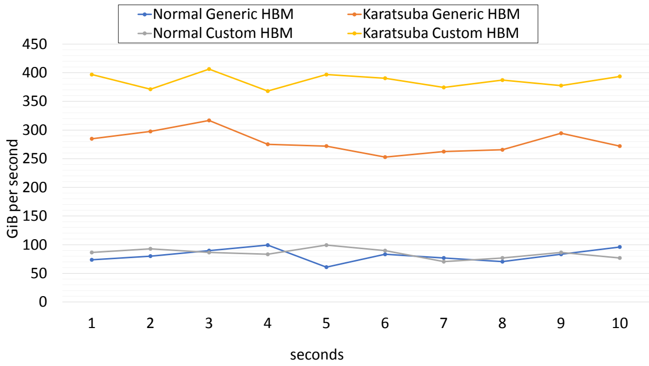


Figure 5.8: HBM Bandwidth utilization for the TFHE-777 different implementations. The custom interface with the Karatsuba multiplier reaches a peak of 406 GiB/s. The theoretical maximum bandwidth is 460 GiB/s

Table 5.2: Resource utilization on the VCU128 board. For TFHE-777 the FPGA is almost fully utilized.

Module	LUT	Register	BRAM	DSP	F7 Muxes	F8 Muxes	Carry8
TFHE-777	891,684	189,705	1,586	6,613	2,720	1,472	1,965
TFHE-50	332,242	31,843	254	1,348	936	489	876
Custom HBM interf.	113,574	76,257	151	0	0	0	138
Generic HBM interf.	38,648	39,194	105	0	0	0	56
MicroBlaze	1,192	810	0	0	34	0	19
DRAM controller	30,236	35,712	51	6	856	0	112
AXI bus	19,771	34,226	0	0	65	32	0

made no difference between the custom and generic interfaces. In general, it never broke the 100 GiB/s mark. Using HBMorphic, i.e., the Karatsuba multiplier in combination with the HBM custom interface, leads to the highest bandwidth utilization as Figure 5.8 shows. It reaches a maximum of 406 GiB/s and a minimum of 371 GiB/s. The highest bandwidth that the Karatsuba multiplier with the generic interface achieves is just 316 GiB/s, which is lower than the lowest bandwidth utilization provided by the custom interface. HBMorphic reaches 88% of the theoretical bandwidth, which is very close to the 90% mark that Xilinx mentions as the maximum practical bandwidth [186]. HBMorphic is able to achieve this as the Karatsuba multiplier requires only 1.04 microseconds to finish multiplication and needs 777 multiplications for one PBS. Fetching one 64-bit word from the HBM using the interface took an average of 132 nanoseconds. Leveraging the parallel nature of HBM all channels are used in a contention-less manner to utilize the full bandwidth, reaching 400GiB/s.

5.6.3 FPGA Resource Utilization

The accelerator is designed to use the most possible resources and make it as parallel as possible. Moreover, the HBM interface implemented has an increased cost. Table 5.2 shows the resource utilization for each implemented module. All the components reside simultaneously on the FPGA except for TFHE-777 and TFHE-50; only one of them exists on the FPGA at one time. The resource utilization for TFHE-777 is notably high, however, this is by design as the goal is to use all possible DSPs alongside the needed LUTs. The design of TFHE-777 is semi-sequential as the number of DSPs on the FPGA is not enough for a fully parallel version and therefore the difference between it and TFHE-50 is not higher than 3×. The custom HBM interface is notably large, however, this was expected based on [186]. It should be noted that it still fits alongside the TFHE-777 on the FPGA. Moreover, if the accelerator would be used in an ASIC design, then it would replace the generic HBM interface from Xilinx, reducing the overhead correspondingly.

5.6.4 Comparison to Related Work

Table 5.3: Comparison to other works constructing FHE accelerators

Accelerator	HBM	Algorithm	Accuracy
HBMorphic	✓	TFHE	full
Ref. [45]	✗	TFHE	partial
Ref. [192]	✓	CKKS	low
Ref. [194]	✗	TFHE	partial
Ref. [193]	✓	CKKS	low
Ref. [74]	✗	TFHE	partial

HBMorphic is compared to the state of the art in Table 5.3. This evaluation is limited to accelerators for FHE, excluding PHE and SHE accelerators. Moreover, the evaluation is qualitative rather than quantitative. The reason for this is that each accelerator targets a different board with different frequency specifications and the algorithms that are accelerated are not similar. Accelerators from [45, 74, 194] have a partial accuracy because they use NTT and FFT multipliers that are not accurate. Moreover, they work under the assumption that the data will be available, i.e., the do not consider any memory bandwidth or data latency. Accelerators from [192, 193] use HBM to resolve the memory bottleneck. However, they have even lower accuracy as they do not only use NTT and FFT multipliers but also target CKKS, which by its design supports only approximate calculations. Therefore, the accelerator is the only one that has full accuracy and is capable of effectively loading the data.

5.6.5 Discussion

To reach maximum speedup using HBMorphic, the memory access patterns of TFHE was analyzed. Based on these patterns, a custom HBM interface is designed and implemented to minimize the data contention. This is successful as HBMorphic increases the HBM bandwidth utilization from 69% (using the generic interface from Xilinx) to 88%. This higher bandwidth utilization leads to speedups of 2.6×, for TFHE-50, and 1.5×, for TFHE-777, to the performance of the accelerator.

While the custom HBM interface reduced the memory bottleneck significantly, the computation overhead was still significant. Using a Karatsuba multiplier greatly helps in getting more speedup without any accuracy loss. Another approach would have been to use FFT or NTT but it would have leads to accuracy losses. Designing the Karatsuba multiplier in a parameterizable way helps in adapting the accelerator to other boards with different specifications.

The memory interface is 3× larger than the generic interface, which leads to a speedup of 2.6× at best. However, comparing the overall overhead, the design becomes only 8% larger for TFHE-777 which is a reasonable overhead. However, the accelerator itself is quite large, as HE usually happens in cloud systems that often offer FaaS, such large FPGAs are usually available.

The ability to fine-tune both the HBM interface and the accelerator gives the edge to FPGAs over GPU. For example, Ref. [174] uses a GPU coupled with HBM to accelerate HE. The accelerator they build uses an NTT-based multiplier and achieves 2.9× speedup at best. This number is similar to HBMorphic's speedup but with a less accurate NTT-based multiplier instead of an accurate but slow multiplier, in concept, if HBMorphic uses an NTT-based multiplier it should be able to get an even higher speed up than the 2.6× because of the reduction in the number of multiplications.

5.7 Summary

This chapter tackles the threat of data leakage from cloud systems. It introduces HBMorphic, a fully homomorphic encryption accelerator on an FPGA with HBM when CSPs offer FaaS. HBMorphic accelerates the state-of-the-art TFHE algorithm which enables processing on encrypted data, and hence, if data is leaked it is in encrypted format using an accurate and fast Karatsuba multiplier. HBMorphic implements the Karatsuba multiplier recursively and in a parameterized way to adapt it to the resource requirements of the system. To load the data with high throughput, HBMorphic uses custom HBM interface. Using this interface and the Karatsuba algorithm HBMorphic has a speedup of 211× and 438× for both variants of TFHE: TFHE-777 and TFHE-50 respectively compared to a baseline implementation using DRAM and a normal multiplier. Compared to the state-of-the-art, HBMorphic is the only TFHE accelerator that supports accurate calculations along with fully benefiting from HBM bandwidth.

6 Eliminating Fault Injection Threats in Multi-tenant FPGAs

Although HE can stop data leakage, an attacker can still try to inject faults into computations and cause them to corrupt. This is relevant in a multi-tenant FPGA setup in accelerated cloud systems. In a multi-tenant setup, the reconfigurable fabric of the FPGA is partitioned into a static region and multiple PRRs. The static region handles supervisory tasks such as managing the reconfiguration of the PRRs, while the PRRs are used by the tenants for their specific designs. This capability allows CSPs to virtualize FPGA resources effectively, allowing multiple tenants to share a single FPGA without disrupting each other's operations.

However, this multi-tenant environment introduces significant security challenges. Research has demonstrated vulnerabilities in FPGAs that can be exploited through remote fault attacks [11, 64, 80, 119]. Such attacks have escalated to actual FaaS in accelerated cloud systems [4, 125], enabling large-scale DoS attacks that can cause significant financial losses for CSPs. These attacks typically target the FPGA's PDN, causing strong voltage fluctuations that lead to sudden shutdowns [80]. Despite existing countermeasures, such as design rule checks and bitstream verification [49, 121, 126], recent malicious designs that use seemingly benign circuits to induce faults or cause DoS remain hard to detect [16, 159].

6.1 Motivational Example

Consider a situation where a CSP provides FaaS to several tenants by letting them share the same FPGA. However, one of these tenants is malicious and

This chapter is based on contributions from [10, 11].

Figure 6.1: System model of a multi-tenant FPGA in a cloud environment. The FPGA is divided into a static partition responsible for management tasks and multiple tenant regions (PRRs), one of which is occupied by a malicious tenant attempting a power-hammering attack.

has designed a circuit intended to exploit the FPGA's PDN. This malicious design, which occupies one of the PRRs, utilizes a large number of oscillators or carefully crafted input patterns to create strong voltage fluctuations. These fluctuations can destabilize the FPGA, affecting the operations of neighboring tenants who are also utilizing their own PRRs.

As illustrated in Figure 6.1, while the other tenants are running legitimate operations within their allocated PRRs, the malicious tenant's design begins to induce significant voltage drops across the FPGA. This could lead to computational errors, data corruption, or even a complete DoS for the entire FPGA. The legitimate tenants, unaware of the malicious activity, may experience unexpected crashes or incorrect outputs, leading to potential data loss and significant downtime.

What makes this scenario particularly dangerous is that the malicious design can be crafted to appear benign during initial security checks. By using standard cryptographic modules or seemingly benign logic, the malicious tenant's design can bypass bitstream verification processes. However, once deployed, the design behaves differently, triggering the power-hammering attack.

The impact of such an attack is not limited to the malicious tenant's PRR; it can extend to the entire FPGA, disrupting all tenants' operations. For CSPs, this not only results in a loss of service, but can also lead to financial losses due to potential reputation damage.

This example illustrates the critical need for robust security measures that can detect and mitigate such attacks in real-time, ensuring that malicious tenants cannot disrupt the operations of others. The development of advanced countermeasures is essential to maintain the integrity and reliability of cloud-based FPGA services.

6.2 Threat Model

The threat model assumes a multi-tenant FPGA environment, where both victim and attacker have their own PRRs. The attacker seeks to exploit vulnerabilities within the FPGA's PDN to induce faults, disrupt the victim's computations, or cause a full system crash (i.e., DoS).

The attacker in this model is capable of deploying malicious designs that can evade traditional security checks. These designs may include circuits that appear benign during design rule checks, but are capable of causing significant harm once deployed. For example, a seemingly innocuous AES encryption module can be modified to induce strong voltage fluctuations when processing specific input patterns [159]. Additionally, the attacker may utilize multiple self-oscillating circuits that do not rely on external clocks, further complicating detection [126].

6.3 Contributions

The contributions of this chapter are:

- Development of the first online countermeasure against Power-Hammer attacks in multi-tenant FPGAs, capable of handling self-oscillating attacks that do not rely on a clock.
- Introduction of a novel reconfiguration-based approach that disables all interconnects in a malicious PRR, faster than existing methods, without affecting other tenants.
- Comprehensive evaluation of various Power-Hammering attacks on multi-tenant FPGAs.
- Proposal of an offline FPGA design classification system that identifies and extracts relevant features from the metadata of a tenant design, categorizing its risk level with better accuracy than state-of-the-art methods.

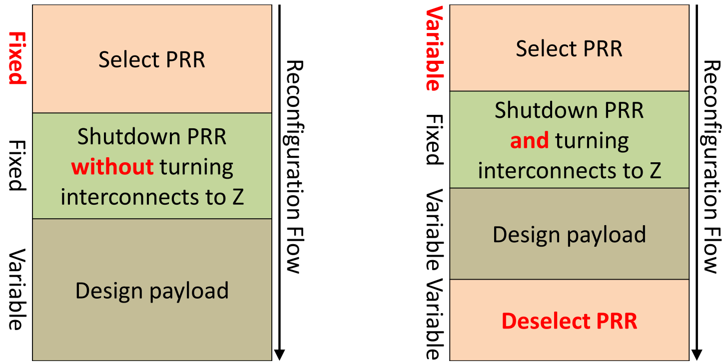
6.4 Tenant Design Analysis and Bitstream Reverse Engineering

To effectively counter Power-Hammering attacks, it is crucial to thoroughly understand the structure of the FPGA bitstream, which will later be pivotal in developing robust countermeasures. The process begins with an in-depth analysis of both malicious and benign tenant designs to identify features indicative of potential security threats.

A key focus of the analysis is on the power consumption estimates of the bitstreams for tenant designs. Although power consumption might seem a straightforward metric, the analysis shows that it is not reliable enough to detect malicious designs published earlier [138]. These designs often use highly regular structures, such as mux-based, latch-based, or glitch-amplification-based configurations. These repetitive structures, composed of small building blocks, make power estimation inaccurate. Therefore, detection cannot solely rely on power consumption; a deeper examination of bitstream metadata is required.

Repetition in bitstream elements is another critical factor. Although repetitive structures are an indicator of many malicious designs, they can also appear in simple benign designs with low resource usage. For instance, benign designs with minimal active logic and large unused areas often display high repetition in their bitstreams because the unused resources are configured similarly to each other. This can lead to false positives, where benign designs are mistakenly flagged as malicious. In contrast, complex benign designs like Bitcoin miners or clusters of diverse modules exhibit high power consumption and low repetition, making them easily distinguishable from simple malicious designs. However, complexity increases when malicious designs are based on benign modules, such as AES-based [159]. These designs mimic complex benign designs, showing both high power consumption and low repetition, which complicates detection.

The analysis goes deeper and reverse engineer the bitstream configuration for Xilinx FPGAs, particularly the 7-series and UltraScale+ architectures [26, 183] to reveal how these devices are programmed and reconfigured, which is vital to the design of countermeasures. Figure 6.2 shows the bitstream structure for both (a) the 7-series and (b) the UltraScale+ FPGAs.

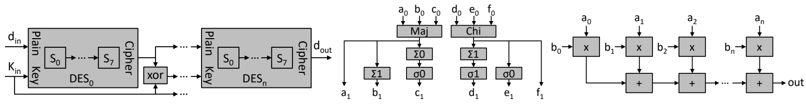


(a) 7-Series **(b)** UltraScale+

Figure 6.2: Partial bitstream structure of 7-Series and UltraScale+ FPGA (major differences shown in red bold font)

In the 7 series, the bitstream begins with a section that selects the PRR to be reconfigured by writing data to specific frames, the smallest addressable entities in the configuration data. This selection process is always a fixed size, regardless of the PRR size or design complexity. Next, the bitstream includes a shutdown command (`SHUTDOWN`) that disconnects the interface between the static logic and the PRR. The logic within the PRR continues running until the design payload, which has a variable size depending on the PRR, is overwritten. This structure means that if a detection mechanism detects an attack and tries to reconfigure the PRR with a benign blank bitstream (bitstream without any logic implemented) it will be slow, as it must overwrite a large portion of the PRR before stopping a potential attack.

The UltraScale+ architecture (Figure 6.2 (b)) introduces several key differences. Unlike the 7-series, the selection part of the bitstream is no longer fixed but scales with the size of the PRR. Additionally, a new, variable-sized deselection section appears after the design payload, which is not documented by Xilinx. The shutdown command has also changed from `SHUTDOWN` to `AGHIGH`, which puts all interconnects of the selected PRR into a high impedance state ('Z'). Hence, if a detection mechanism detects an attack and performs reconfiguration with a benign blank bitstream, it will effectively stop the attack before the design payload is reconfigured.



(a) DES-based attack **(b)** SHA-based attack **(c)** Reed-Solomon-based attack

Figure 6.3: Implemented attacks, derived from benign modules. With small modifications, removing sequential elements, and special toggling input patterns, they lead to successful attacks

Despite these improvements, there is still a risk that an attack could succeed before the countermeasure takes effect. If the malicious design is activated immediately after reconfiguration, even the rapid detection and response by voltage sensors might not be fast enough. The need for the deselection block of the malicious bitstream and the selection block of the benign blank bitstream to be reconfigured before executing the `AGHIGH` command introduces significant delays.

Although using bitstream compression [153] can speed up reconfiguration by reusing data across frames, the tests show that this approach, while making reconfiguration five times faster, is still insufficient to stop many attacks [11]. Thus, a faster and more effective method is required to prevent crashes.

This detailed understanding of the bitstream structure not only explains the shortcomings of existing countermeasures, but also provides the foundation for developing more effective solutions.

6.5 Extending the Seemingly-benign Power Hammering Attacks

To further show the threat from Power Hammering, malicious tenant designs similar to the AES malicious design from [159] are designed. These designs are more stealthy than the attacks from Figure 2.6. These malicious tenant designs are based on the Data Encryption Standard (DES), Secure Hash Algorithm (SHA), and Reed-Solomon, as depicted in Figure 6.3. The malicious DES-based design in Figure 6.3a utilizes unrolled DES S-boxes as the fundamental building block. Multiple blocks are interconnected in a chain with adjustable chain lengths to fit the size of the tenant region. The output of each block serves as the input for the subsequent block. The key for each block is computed by XORing the output of the preceding block with the original key. This process amplifies the toggling along the path, thereby increasing the power consumption.

The malicious SHA-based design also employs a chain of interconnected SHA sub-functions (shown in Figure 6.3b). Each sub-function receives six inputs, which are mixed to produce the various components of the SHA algorithm, resulting in six outputs. The output of one sub-function can be directly connected to the next's input, with the chain's length configurable as desired. Note that only the first input originates from the registers and that no combinational loops are present in the design.

As the Reed-Solomon encoder inherently comprises a chain of multiply-accumulate operations, the registers between the adder stages are simply removed to transform it into a malicious design (see Figure 6.3c). This modification results in a lengthy combinational path which can be configured as desired. The inputs originate from tenant-internal registers initialized by constants and subsequently inverted in every cycle to enhance toggling.

Furthermore, to improve detection difficulty, the concept of hiding these malicious designs among benign ones to avoid detection by current state-of-the-art solutions is explored by integrating malicious designs alongside a cluster of ISCAS sequential circuits [44]. Consequently, a bitstream scanner would identify slightly modified benign designs and encounter additional circuits introducing randomness to the structural design. This combined setup presents a more complicated functionality resembling a standard design, performing tasks beyond solely cryptographic operations or encoding.

6.6 Meta-Scanner: Identifying Malicious FPGA Designs

The main goal is to develop an offline scanner that allows the CSP to analyze tenant designs before uploading them to an FPGA. This should be done without a time-consuming and extensive netlist analysis. The idea is to classify tenant designs into three categories: high risk (RED), mid risk (YELLOW), and low risk (GREEN), which removes the burden from runtime countermeasures to identify the malicious tenant before starting the countermeasure.

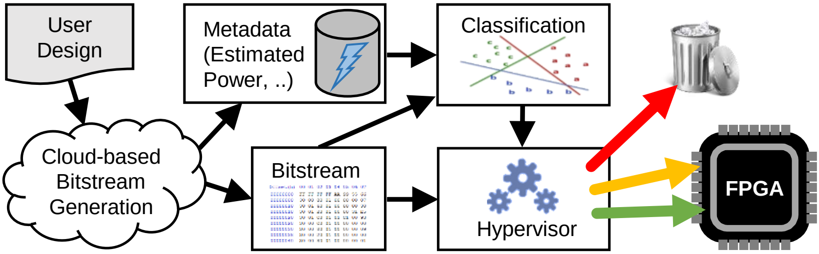


Figure 6.4: Basic principle of the proposed Meta-Scanner and loading flow. Bitstreams classified as being high-risk are not loaded. Other bitstreams are loaded but with careful placement.

6.6.1 System Overview

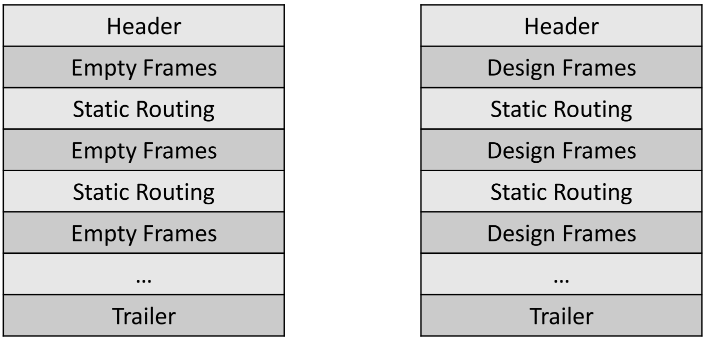
The focus is mainly on detecting malicious tenant designs. After correctly classifying the risk level of each tenant design, CSPs are able to decide whether to upload it or not. The assumption is that CSPs perform security checks or attestation of the FPGA design through a hypervisor as explained by previous works [198]. Moreover, CSPs can combine the risk classification with other data they might have. Usually, CSPs can have access to more information about their users, e.g., their history of previous tenancy on FPGAs. Hence, they may have some trust metric for users.

The steps for using Meta-Scanner are shown in Figure 6.4. Normally, a tenant would upload a design as an HDL code or as a netlist to the CSP. The CSP then generates the bitstream and extracts the features used by the scanner from the metadata. Then, based on the scanner, the CSP can correctly assess the risk category of the bitstream.

The hypervisor should never upload RED tenants (see Figure 6.4), as they are very likely to exhibit malicious behavior, whereas GREEN tenants can always be uploaded, as they are incapable of showing malicious behavior. YELLOW tenants can be uploaded to an FPGA, but special care must be taken. When ensuring that at most one YELLOW tenant is executing on an FPGA, then online countermeasures can target the potentially malicious tenant, allowing them to shut it down as soon as it measures any malicious activity. Instead, if two or more YELLOW tenants were on the same FPGA, it would no longer be known which of them started the malicious activity. Thus, the online countermeasures would no longer be able to localize and stop the activity fast enough before a crash occurs.

6.6.2 Metadata Extraction

The idea is to identify the area utilization of a tenant and its internal regularity by extracting corresponding properties directly from its bitstream. For every reconfigurable region, the synthesis tools for partially reconfigurable designs create a blank bitstream (shown in Figure 6.5a) that reconfigures the region into an empty state. A normal design bitstream for the same region can be seen in Figure 6.5b. It has the same structure as the blank bitstream. For unused regions, the frame data is identical to the frame data of the blank bitstream. Hence, any frame with data identical to the corresponding frame in the blank bitstream can be seen as empty.



(a) Blank Bitstream   **(b)** Design Bitstream

Figure 6.5: Bitstream Structure for blank bitstream and design bitstream. Both bitstreams will have the same static routing, but the design bitstream will have the content of the frames different.

Based on the bitstream structure, 5 features are extracted as follows. Note that the equations use the annotation from Table 6.1.

Table 6.1: Annotation of the mathematical explanation for the features

Variable	Explanation
$Bitstream_{Len}$	Number of frames per bitstream
$N_{UFrames}$	Number of unique frames
$N_{BFrames}$	Number of blank (empty) frames
$NonBFrames$	Non Blank Frames

- **Repetition:** The number of non-unique frames. If there are for instance 100 frames with identical data, that adds 100 to the Repetition. Nothing is added to the Repetition for a unique frame (i.e., no other frame has the same data). A higher Repetition indicates a higher risk of self-oscillating structures, as they normally consist of many repeated frames.

$$Repetition = Bitstream_{Len} - (N_{UFrames} + N_{BFrames})$$

- **Utilization:** The number of frames different from the frame data at the same position in the blank bitstream. This helps to identify complex designs that use a large degree of their resources.

$$Utilization = Bitstream_{Len} - N_{BFrames}$$

- **Average Frame Frequency (AvgFrameFreq):** is based on a histogram of all non-blank frames in the bitstream, i.e., of those frames that are different than the corresponding frame in the blank bitstream. The frequency of the histogram's bins denotes how many frames belong to that bin, i.e., how many frames have the same data. The AvgFrameFreq is equal to the average over the frequencies divided by the largest frequency. If the AvgFrameFreq is near one, it indicates a low degree of repetition, while if it is close to zero, it indicates a higher degree of repetition.

$$AvgFrameFreq = \frac{mean(hist(NonB_{Frames}))}{max(hist(NonB_{Frames}))}$$

- **Standard Deviation of the Frame Frequency (StdFrameFreq):** The metric calculates the standard deviation of the frame frequencies and then divides it by the largest frame frequency. This helps to identify how much repetition exists. A low deviation means that there is a high degree of repetition and a high deviation means that there is a low degree of repetition.

$$StdFrameFreq = \frac{std(hist(NonB_{Frames}))}{max(hist(NonB_{Frames}))}$$

- **Estimated Power:** This feature estimates the design's power consumption. It is the only feature not directly calculated from the bitstream but is reported by the synthesis tools after the design is placed and routed. Note that for the Amazon Cloud, the CSP has access to this information, as the place and route of a tenant design is performed under the control of Amazon.

$$EstimatedPower = VivadoPowerEstimation$$

Using these five features covers all the important aspects of high utilization, high power, regular structures, and regular structures hidden with some irregularities, which are essential for classifying the designs. Overall, they were effective enough to keep the accuracy, recall, and precision around 97%.

6.6.3 Proposed Classification

To demonstrate the feasibility of a machine learning approach, a set of 475 different tenant designs was first manually labeled. Then they were tested on a ZCU102 FPGA board according to the three risk classes and evaluating various classifiers on the set. The tenant designs are labeled according to the following principles:

RED (high risk): These tenant designs contain actual attack circuits, which are intentionally designed as malicious using different approaches both from the literature [29, 119, 120, 159, 176]. The hypervisor should never load them to tenant regions on the cloud FPGAs.

YELLOW (mid risk): If a circuit contains a lot of resources and may be used in combination with another similar design on the same FPGA to invoke crashes, id labeled as a YELLOW design. The hypervisor can permit these designs but requires consideration regarding the mapping into FPGA regions. Note that this definition includes completely benign but resource-intensive as well as intentional malicious designs. For instance, additional logic may be added to confuse offline bitstream checker and *hide* the attack, or attackers might use reduced variants of the RED designs based on multiple seemingly-benign modules. Multiple YELLOW-labeled tenants should not be present at any given time in the FPGA to prevent attacks. If at most a single YELLOW design is deployed per FPGA, runtime countermeasures will be fast enough to disable it in case of any detected malicious activity.

GREEN (low risk): Tenant designs from the GREEN category are considered harmless and can be arbitrarily placed into different FPGA regions by the hypervisor. They are neither resource-intensive nor contain known malicious structures such as self-oscillating circuits. Attacks are highly unlikely, even if combined with YELLOW designs on the same FPGA.

Based on the recommendations in [91], the evaluation uses 10-fold cross-validation for different classification methods. SVM, Multi Layer Perceptron (MLP), and Random Forest (RF) are tested. RF performed the best on the dataset and was used it in all further experiments. The scikit-learn python library [154] is used to implement the classifier and focus on optimizing the recall for classification of the RED bitstream class by setting the class weights to 200, 30 and 1 for RED, YELLOW and GREEN respectively. This approach prevents the misclassification of attack bitstreams into a lower-risk class. Thus, it maximizes the security at the cost of very few lower-risk bitstreams not being loaded to the FPGA.

6.6.4 Dataset Generation

To evaluate the effectiveness of Meta-Scanner in fulfilling its goal, the dataset of the bitstreams is generated. Table 6.2 summarizes the terminology used to describe the dataset generation.

Table 6.2: Terminology used for data generation.

Term	Explanation
Basic Design	HDL code of one module, e.g., DES or JPEG
Tenant Design	One basic design or several of them in a cluster
Tenant region	PRR on the FPGA assigned to one tenant
Floorplan	Partitioning the FPGA into different tenant regions
Bitstream	Tenant design in binary, uploaded on the FPGA

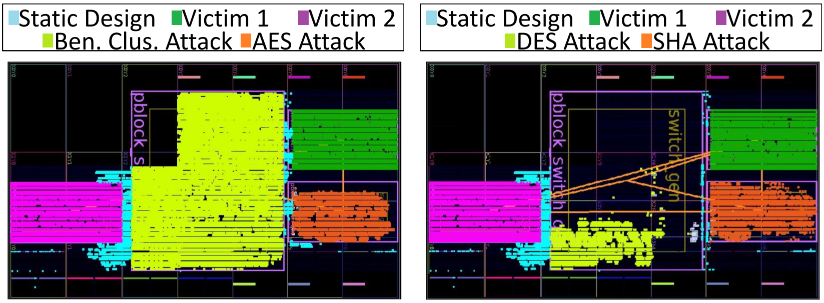
A set of bitstreams is built for metadata extraction and solution testing, based on 26 basic designs (9 malicious, 17 benign). These are configured into 475 tenant designs. The 9 malicious designs are state-of-the-art and three new designs. The 17 benign designs come from various benchmarks and some in-house designs like JPEG compression and RSA SHA. They are mixed to create tenant designs. Table 6.3 details the designs, sources, and usage frequency. Real tenant designs from CSPs are inaccessible. AWS Marketplace [32] cores are typically either simple designs for integration [62] or complete systems with indirect hardware access [100]. Thus, benchmarks were used for evaluation as done by [16, 50], covering applications including Neural Networks and Bitcoin mining.

Table 6.3: Basic Designs for Bitstream Generation

Basic Design	Benchmark	#Bitstreams
JPEG	Own Designs	61
RISCV	RISC-V River SoC	15
AVA decoder	Berkeley	42
RSA	Own Designs	46
Cluster of seq. circuits	ISCAS Sequential	64
Serial keyboard	Groundhog	24
PID Controller	Groundhog	45
FIR	Berkeley	28
FFT	Groundhog	40
Bitcoin miner	Opencores	22
AES Attack	Attack from [159]	59
Mux Attack	Attack from [126]	5
Shift register attack	Attack from [159]	20
RAM Attack	Attack from [29]	19
Reed-Solomon attack	Own Designs	19
DES*	Berkeley	25
SHA*	Own Designs	25
Glitch Attack	Attack from [138]	20
Latch Attack	Attack from [126]	20
Neural Network	Opencores	38
Ethernet	Opencores	38
CRC	Opencores	57
SPI	Opencores	57
Manchester encoder	Opencores	38
IIR	Opencores	38
DCT	HLS	25

* Used both maliciously and benignly

The bitstreams are generated for the ZCU102 FPGA board, utilizing its Xilinx UltraScale+ FPGA for measurements to establish labeling ground truths. These bitstreams are then loaded onto the FPGA board. The focus lies in detecting the success of attacks, which determines the labeling of the bitstreams. The same bitstreams can be used across multiple target FPGA boards, mirroring a cloud scenario from the user's perspective.



(a) AES and benign cluster coordinated tenant attacks **(b)** SHA and DES coordinated tenant attacks

Figure 6.6: Floor-planning of tenants where multiple tenants have different resource assignments and utilization on the same FPGA.

Various strategies are employed to create tenant regions. For example, Figure 6.6 demonstrates the implementation of coordinated attacks from multiple tenants. The FPGA's floor plan is divided into four regions, with two hosting malicious designs and the other two hosting benign ones. One region utilizes 50% of the resources, while the other three each utilize 15%, leaving 9% for the static design. In the example shown in Figure 6.6, the 50% region is positioned in the middle of the FPGA. However, for another floor plan, the 50% region can be placed at the top or bottom of the floor plan, not necessarily in the middle. This contributes to diversifying the bitstreams by avoiding constraining them into fixed regions but instead across several different regions.

A CSP typically maintains several floor plans to accommodate various types of users. For instance, the 50% tenant region from Figure 6.6 can be substituted with two smaller tenant regions, each utilizing 25% of the resources. Six different floor plans are the basis to generate 24 distinct tenant regions for placing tenant designs. The sizes of these regions vary, ranging from 50% of the FPGA resources to 15% of the FPGA resources.

Not all tenant designs were used in all the tenant regions as they might not fit into them, i.e., they need more resources than the region provides. Those tenant designs that did not fit were either modified, like changing the RISC-V dual core to a single core, or diversifying the designs further by the following modifications: (i) mixing them more, e.g., substituting a large FFT module with a smaller controller module and a Manchester encoder, (ii) increasing the repetition within the design, e.g., adding multiple JPEG compression instances after removing a large DES module. Moreover, hiding some malicious modules with benign modules makes the attacks stealthier, similar to [50]. The generated tenant designs are categorized into 153 GREEN ones, 120 RED ones, and 177 YELLOW ones.

6.7 LoopBreaker: Online Countermeasure against Power Hammering

As the YELLOW class exists and false negatives can occur, the offline countermeasure has to be complemented by an online countermeasure. Using the blank bitstream to deconfigure a malicious tenant is too slow. Therefore, The aim is to disable the interconnects as fast as possible. This is achieved by generating a carefully designed LoopBreaker bitstream. Other works have already studied the composition of the bitstream [178] and have even successfully generated bitstreams with a smaller size [166] by partitioning the design payload of regular bitstreams into multiple smaller bitstreams. That allowed them to fulfill latency constraints of the reconfiguration process in real-time scenarios.

This approach ignores the payload and uses the `AGHIGH` command to change all interconnects to the 'Z' state. Each part of the bitstream (i.e., select, shutdown, payload, and deselect, as shown in Figure 6.2) is separated into a custom bitstream. This allows individual configuration of selection, deselection, shutdown, and payload, facilitating precise control over a potentially malicious PRR. Splitting an existing bitstream into its parts doesn't result in valid bitstreams. Specific synchronization/desynchronization steps must be added to each part to validate the bitstream. Some desynchronization steps are required in the generated bitstreams for functionality, while others must be omitted to maintain the 'Z' state. For instance, the desynchronization includes the `GRESTORE` and `DGHIGH` commands, which revert the interconnects to normal. Additionally, `NOP` commands must be inserted at specific points. Following certain commands, `NOPs` must be added—too few cause errors, while too many cause delays.

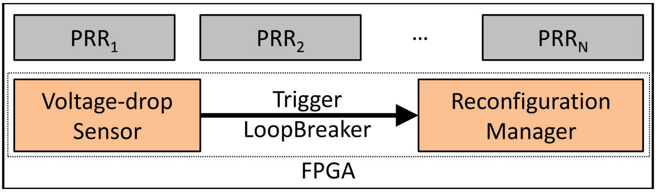


Figure 6.7: Multi-tenant FPGA with sensor and reconfiguration manager included

After correctly using the commands, the Cyclic Redundancy Check (CRC) checks of the bitstream data need to be treated properly. As the detailed CRC calculation rules are not documented, these checks need to be disabled when manipulating the bitstreams. However, simply disabling the CRC calculation does not work, because several commands require a specific CRC check. To identify these commands, a detailed analysis had to be performed. After identifying all these commands, the required CRC can be replaced by CRC reset commands. Simply removing the CRC check commands does not work, as the bitstream would no longer function correctly.

An analysis of the bitstream structure enabled the creation of different bitstreams (selection, shutdown, and deselection). The most crucial is the shutdown bitstream, known as LoopBreaker, which disables interconnects. There are two versions of this bitstream: one for the 7-series with 89 commands and one for the UltraScale+ with 310 commands. The length difference arises from the varying number of `NOPs` following each command. Additionally, the 7-series does not execute the `AGHIGH` command but instead performs the `SHUTDOWN` command, preventing further DPR after applying the LoopBreaker bitstream. Once all tenants have finished processing, the FPGA needs reconfiguring with the full bitstream. This prevents crashes, allowing tenants to continue work. In contrast, UltraScale+ FPGAs can normally perform subsequent DPRs.

Figure 6.7 shows the full multi-tenant system that is used with LoopBreaker. The connections to external components (e.g., RAM) are not shown as they are not needed for the following explanations. When a tenant is reconfigured into a PRR, reconfiguration of its deselect part is skipped, in order not to miss an attack in case it starts attacking immediately. In this way, the PRR is still selected, which allows us to disable it quickly if needed. The voltage drop-sensor monitors the system and, upon sensing an attack, notifies the reconfiguration manager, which then reconfigures the LoopBreaker bitstream to disable the PRR. In case that no attack was detected, during which another PRR reconfiguration shall be performed, then the reconfiguration manager first reconfigures the deselect bitstream, before reconfiguring the new bitstream. Based on hints from Meta-scanner (e.g., a YELLOW tenant exists on the FPGA) then the reconfiguration manager can also reconfigure the select bitstream for that specific PRR, in order to be prepared for an attack.

6.8 Performance of Attacks and Countermeasures

The tenant designs are implemented using Vivado 2019.1 to evaluate the proposed Meta-Scanner and LoopBreaker. The bitstreams were uploaded to a ZCU102 board. For the reconfiguration manager CoRQ [56] is used. Meta-Scanner is implemented in Python and tested on an AMD Ryzen 5 6-Core processor with 24 GiB main memory.

6.8.1 Ground Truth of Seemingly-benign Attacks

To label the novel malicious tenant designs they are run on a ZCU102 board to see if they crash the FPGA. Table 6.4 shows the results. The utilization (%) is based on the total LUTs available in the ZCU102 FPGA board. Any version of malicious designs having the size Table 6.4 or larger is labeled as RED.

Furthermore, smaller stealthy malicious designs are labeled as YELLOW due to their potential to coordinate attacks, substantiated by the findings presented in Table 6.4. Initially, when both tenants, SHA and DES, are malicious and deploy weakened versions of their attacks, a coordinated attack becomes feasible. Secondly, in scenarios where only one tenant (AES) is malicious but cannot execute an attack independently, it can exploit the presence of a resource-intensive benign tenant. When executed concurrently, the benign tenant inadvertently facilitates an attack, resulting in a system crash. Consequently, any benign large design capable of instigating an attack when combined with the small AES attack is classified as "YELLOW."

Table 6.4: Minimum time and utilization needed for achieving crashes using seemingly-benign attacks.

Attack based on	Crash speed	Crash FPGA utilization
AES* [159]	12 μs	18.5%
Reed-Solomon*	167 μs	38.7%
DES*	90 μs	27.0%
SHA*	34 μs	21.9%
SHA + DES+	60 μs	14.6% + 18.0%
AES + benign cluster+	>2 Min	13.9% + 34.0%

* attack from single tenant

\+ attack from multiple coordinated tenants

Table 6.5: Results of 10-Fold Cross Validation across 475 Total Bitstreams. Mean Accuracy: 0.979 ± 0.02. Mean accuracy of detecting newly introduced attacks: 0.968.

class	precision	recall	f1score	support	FPR	FNR
GREEN	0.990	0.979	0.984	17.8	0.007	0.020
YELLOW	0.969	0.978	0.972	17.7	0.015	0.016
RED	0.977	0.985	0.979	12.0	0.008	0.021
New RED	1.0	0.963	0.978	1.7	0.000	0.018

6.8.2 Performance of Meta-Scanner

The metadata from bitstream generation trains the random forest classifier. Data is split with 10% for testing using scikit-learn split method [154]. A

Table 6.6: Comparison with the state of the art. The numbers are based on the dataset, and tools are assumed to detect the mentioned attacks correctly. This conservative comparison ensures fairness. Only for the classifier, the mean accuracy ± standard deviation is presented.

Metric	M-S	Ref. [121]	Ref. [126]	Ref. [49]	Ref. [64]	Ref. [50]	Ref. [16]
Accuracy	0.979 ± 0.02	0.789	0.756	0.709	0.840	0.825	0.836
Hidden Attacks	✓	✗	✗	✗	✓	✗	✓
Partial Bitstreams	✓	✓	✓	✓	✗	✗	✓
Cryptographic Attacks	✓	✗	✗	✗	✗	✓	✓
Non-Cryptographic Attacks	✓	✗	✗	✗	✗	✗	✗
Short circuit Attacks	✓	✗	✗	✗	✗	✗	✗
Coordinated Attacks	✓	✗	✗	✗	✗	✗	✗

10-fold cross-validation on 475 bitstreams is shown in Table 6.5. The RED class has the highest recall and precision to avoid banning legitimate and uploading malicious designs, achieved by fine-tuning class weights. The GREEN and YELLOW classes also have high precision and recall, and the classifier's mean accuracy is 0.979. For novel attack designs, inference shows a mean accuracy of 0.95, precision of 1.0, and recall of 0.963. FPR and FNR are at most 0.021 as shown in Table 6.5, which is low. The FPR for YELLOW is about twice that of the other classes since errors in RED or GREEN often result in YELLOW. The FNR for YELLOW is slightly lower than the other classes but remains low overall.

Table 6.6 compares the scanner to the five state-of-the-art approaches [49, 50, 64, 121, 126]. As they can only classify into two classes (attack vs. no attack), the YELLOW and GREEN classes are considered as 'no attack', to give them an advantage and to have a conservative comparison. However, all state-of-the-art approaches have significantly lower accuracy compared to Meta-Scanner. Note that for the tools from [50, 64] the tool does not even support partial bitstreams in its current format. However, for a fair comparison, the conservative assumption is that they could be updated to support them. Meta-Scanner is the only tool that detects BRAM short circuit malicious designs and non-cryptographic seemingly-benign malicious designs (Reed-Solomon-based and shift-register-based).

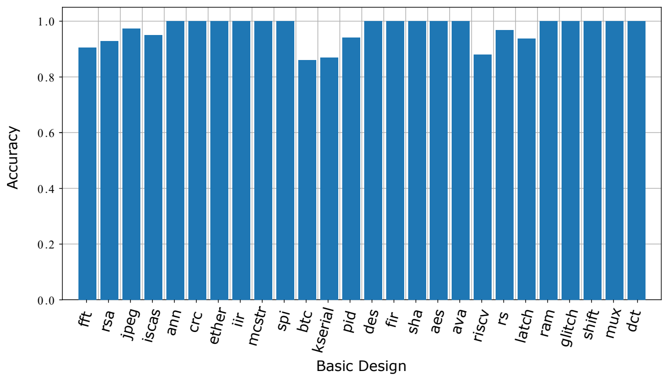


Figure 6.8: Mean detection accuracy depending on basic designs.

Moreover, Figure 6.8 shows the accuracy of classifying each basic design to the correct classes. The accuracy is defined as the number of samples correctly classified, divided by the total number of samples used for the inference. Many of the accuracy values are at 1.0, which means that no false positives nor false negatives occur for this basic design. Overall, all accuracy values are higher than 0.85. DES and SHA (which are both used as benign designs as well as malicious designs hidden using ISCAS circuits) have a high accuracy of 1.0. Hence, the scanner was able to correctly detect hidden malicious designs, and differentiate between using a module for an attack or using it as a true benign design. Moreover, the scanner can detect all the new malicious designs with high accuracy.

Additionally, the timing overhead is evaluated. The CSP performs Place & Route, feature extraction from the metadata, and scanning (inference of the classifier). Table 6.7 shows the results of running the scanner on the AMD Ryzen 5 6-Core processor with 24 GiB main memory. On average, Place & Route for one bitstream needed 27 minutes, while the feature extraction needs less than two seconds and the inference needs less than 10 milliseconds. Hence, the feature extraction and inference have negligible overhead. The feature extraction takes more time than the inference as it needs to parse the bitstream frame by frame. Moreover, the time needed for training is also measured, Meta-Scanner needs on average 2 minutes to train the decision tree.

Table 6.7: Timing overhead of the Classifier

	Training	Feature Extract	Inference	P&R
Time needed	2 min	1.6 s	5.0 ms	27 min

6.8.3 LoopBreaker's Worst Case Performance

To evaluate LoopBreaker the more aggressive self-oscillating attacks are used as they consume higher power and can lead to successful attacks faster than the seemingly-benign ones. LoopBreaker and Blanking require 1.56 µs and 200 µs, respectively, to successfully stop an attack. Note that bitstream compression is used for the Blanking solution, otherwise it would take longer (i.e., 1 ms). Figure 6.9 shows the latency until an attack leads to a crash or a timing fault. For each attack type and attacker size (i.e., Y-axis in Figure 6.9), they are tested with different toggling frequencies, the experiments are repeated 20 times and the fastest observed time until a fault/crash occurred is reported.

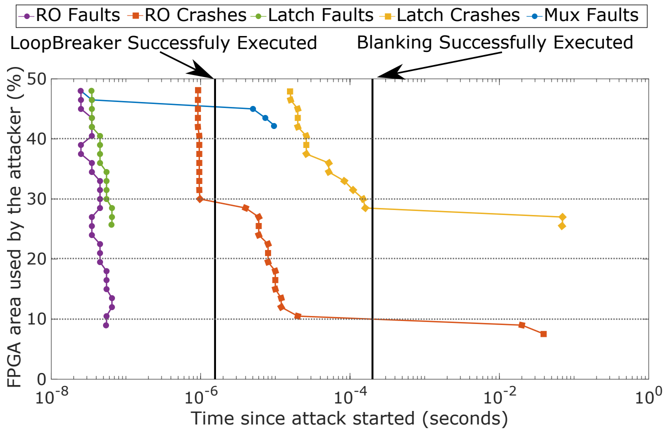


Figure 6.9: Observed attack latency leading to timing faults or crashes compared to LoopBreaker (1.5 µs) and Blanking Bitstream (200 µs) execution times. If execution time exceeds attack latency, faults/crashes occur. LoopBreaker prevents all crashes when the attacker uses up to 30 % of the FPGA area.

As a general trend, a larger attacker size typically needs a shorter time for the attack to be successful. The two vertical lines in Figure 6.9 (i.e., LoopBreaker and baseline Blanking) show the time needed from the start of the attack until the solution successfully stops it.

Attacks that are faster than the countermeasure, cannot be prevented. Note that this evaluation did not only *calculate* whether or not a countermeasure should theoretically prevent an attack (by comparing times), but they are experimentally tested that, by running the attack and the automated detection and prevention on the FPGA boards. Figure 6.9 shows that the Blanking solution could only prevent a small portion of the crash attacks, whereas LoopBreaker can stop most of them. Only crashes due to RO-based attacks that use attacker size larger than 30 % of the available FPGA area, could not be prevented by LoopBreaker. However, RO-based attacks can be easily detected by offline methods, i.e., before even reconfiguring the malicious tenant to the FPGA. The more realistic latch-based attacks would all be prevented by LoopBreaker, whereas Blanking was too slow for most of them. Note that attacker sizes larger than 50 % could lead to even faster attacks, however, that would not leave enough space for a same-sized second tenant and thus is irrelevant for multi-tenant systems.

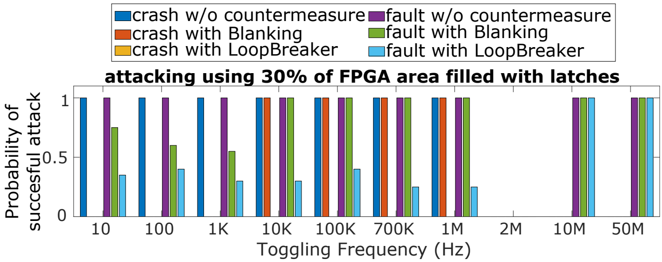


Figure 6.10: Probability of crashes and timing faults (bars) under latch-based attacks (see Figure 2.6(b)) from a 30 % FPGA area attacker at various toggling frequencies (X-axis). The LoopBreaker solution results in 0 % crashes and significantly reduces timing faults.

Most timing faults occur so fast after the start of the attack. Not even LoopBreaker could prevent them. However, no timing fault went undetected by the sensor used. Therefore, the malicious tenant could be stopped as fast as possible to prevent any additional faults in the other tenants' region. Additionally, the detection of the attack (and thus the increased likelihood of timing faults) is reported to the system manager, which can then inform the tenants to take appropriate measures (e.g., rollback in case they were not protected by redundancy measures like Triple Modular Redundancy (TMR)).

6.8.4 LoopBreaker's Average Case Performance

So far only the worst case attack scenario was considered. However, attackers might not have possession of an FPGA with the same setup as the one in the cloud environment. Therefore, they cannot perform a full characterization and thus do not know the most destructive toggling frequency. Figure 6.10 shows the effect of different toggling frequencies on the probability of an attack leading to a crash or timing fault. The attacks considered here use enhanced latches (from Figure 2.6(b)) and an attacker size of 30 % of the available FPGA area. Altogether, 10 different toggling frequencies are evaluated as shown in Figure 6.10. These 10 frequencies represent the decades from 10 Hz up to 100 MHz. Furthermore, the measurement results for 700 kHz and 2 MHz, had a distinctive behavior that is worth mentioning.

At 700 kHz, the attack primarily causes faults without crashes, though occasionally crashes occur post-attack. At 2 MHz, no crashes or timing faults were observed. The reasons are unclear, but high MHz frequencies seem less crash-inducing; no crashes were seen at 10 MHz and 50 MHz without countermeasures. Notably, LoopBreaker does not experience crashes in benchmarked scenarios (30% latch-based attacks) and significantly reduces timing fault probability compared to Blanking, from 10 Hz to 1 MHz.

Table 6.8: Probability of successful crash and fault, depending on the attack type, countermeasure type and attacker size

	Attack Type	Countermeasure	Attacker size for Ring Oscillator (RO)-based attacks					Attacker size for Latch-based attacks		
			7.5%	15%	22.5%	30%	45%	25.5%	30%	45%
Probability that the Attack leads to a Crash	Worst case	No countermeasure	40%	100%	100%	100%	100%	100%	100%	100%
		Blanking	0%	100%	100%	100%	100%	0%	100%	100%
		LoopBreaker	0%	**0%**	**0%**	100%	100%	0%	**0%**	**0%**
	Average case	No countermeasure	9.3%	70%	70%	100%	100%	19.5%	70%	70%
		Blanking	0%	40%	40%	70%	100%	0%	40%	40%
		LoopBreaker	0%	**0%**	**0%**	**40%**	100%	0%	**0%**	**0%**
Probability that the Attack leads to a Timing Fault	Worst case	No countermeasure	0%	100%	100%	100%	100%	100%	100%	100%
		Blanking	0%	100%	100%	100%	100%	100%	100%	100%
		LoopBreaker	0%	100%	100%	100%	100%	100%	100%	100%
	Average case	No countermeasure	0%	90%	90%	100%	100%	39.5%	90%	90%
		Blanking	0%	90%	90%	100%	100%	39.5%	79%	90%
		LoopBreaker	0%	90%	90%	100%	100%	**20%**	**42.5%**	90%

Table 6.8 shows the detailed results for different combinations of attack type, countermeasure type, attacker size and toggling frequencies. In addition to the latch-based attacks, at attacker sizes of 30 % (as shown in Figure 6.10), the results are of RO-based attacks and different attacker sizes, ranging from 7.5 % to 45 % (i.e., the biggest size that leaves enough space for a second same-sized tenant). Due to the limited success of Mux-based attacks (i.e., leading to no crashes and much less timing faults than RO-based or Latch-based attacks), they are excluded from this analysis for brevity. Each probability is calculated based on 20 runs of the specific combination. Table 6.8 shows two cases for each scenario: An *average case* where the attacker uses a random toggling frequency, and the *worst case* where the most-destructive toggling frequency is used.

By looking at the evaluation of crashes (in the upper half of Table 6.8), it is noticeable that LoopBreaker countermeasure is at least as good as the Blanking countermeasure, and most of the time is even better. For RO-based attacks, LoopBreaker can prevent all crashes up to an attacker size of 22.5 %, whereas Blanking is only able to prevent crashes up to an attacker size of 7.5 % (i.e., an attacker that uses 3 times less area). For latch-based attacks, LoopBreaker can even prevent crashes in all evaluated scenarios, whereas Blanking can only prevent crashes up to an attacker size of 25.5 %.

Fault evaluation (lower half of Table 6.8) shows that for RO-based attacks, Blanking and LoopBreaker perform no better than no countermeasure. Most timing faults occur too quickly for LoopBreaker to prevent, but the attack is detected and reported. For Latch-based attacks with random toggling frequency, LoopBreaker reduces the probability of timing faults more effectively than Blanking. Crucially, LoopBreaker significantly reduces crashes, keeping multi-tenant systems operational, even when an attacker uses more than 22.5 % of the available FPGA resources and knows the most-destructive toggling frequency.

6.9 Summary

This chapter addresses fault injection in multi-tenant FPGAs via power hammering. It proposes Meta-Scanner for offline detection of fault attacks in cloud FPGA instances. By analyzing bitstream metadata, Meta-Scanner implements a classifier for scanning. Using machine learning, Meta-Scanner categorizes client bitstreams into high-risk (blocked), low-risk (mapped arbitrarily), and mid-risk (allowed with restrictions) classes. The random forest classifier achieves 0.979 ± 0.02 accuracy on 475 bitstreams, demonstrating feasibility. Meta-Scanner has low overhead and adapts to new attacks. Complementing this, LoopBreaker provides online countermeasures, using partial reconfiguration and a voltage sensor to disable malicious tenants. The method stops attacks in 1.5 μs, making it the first effective online approach against Power-Hammering.

7 Conclusion

The advancements in accelerated cloud systems present both opportunities and challenges, particularly in securing sensitive data and computations. This dissertation presents a thorough work across multiple domains, addressing the critical issues of authentication, covert-channel mitigation, data leakage prevention, and fault injection threats in accelerated cloud systems. These topics remain vital in the quest for secure and efficient computational systems, especially as modern industries expand into the areas of AI, IoT, and real-time systems.

7.1 Summary of Key Contributions

The first key contribution of this dissertation lies in the exploration and development of Machine Learning-resilient Physical Unclonable Functions (PUFs) for client-server authentication with resource-constrained client devices. By leveraging the inherent unpredictability of PUFs, this approach significantly enhances the security of cloud-connected devices. The proposed approach, designed for client devices with limited computational and storage resources, ensures that the authentication protocol remains lightweight and efficient, without sacrificing security. It introduces two PUFs, an FPGA-based one (CaPUF) and an NVM-based one (ANV-PUF). Both PUFs are able to mitigate the modeling-based attacks with a modeling accuracy of around 50%, similar to flipping a coin. Moreover, the reliability challenges for both types of PUFs are analyzed. CaPUF cannot be used as a secondary accelerator when DPR is used. ANV-PUF can be used for generating 10^7 response bits before it suffers endurance degradation.

The second contribution addressed the critical challenge of covert-channel attacks in FPGA-MPSoCs. In FPGA-MPSoCs, malicious users can exploit these covert channels to leak sensitive information. In this dissertation, thermal and power-based covert channels were studied, illustrating their potential for misuse in real-world scenarios. The experimental results demonstrated the severity of these vulnerabilities when not properly mitigated. First, Through-Fabric shows how the usage of accelerators can be manipulated to break TEE and leak data via thermal covert channels. Second, Covert-Hammer shows how multiple malicious users can coordinate among themselves a communication protocol to exchange and lak data via power covert channels. Finally, to mitigate such attacks, countermeasures in hardware and software are presented. For hardware, it illustrates using ROs to induce noise and hinder covert channel communication. For software, adding increased delays when using the accelerator adds temporal noise to lower the success of the attacks.

The third major contribution focused on the use of Homomorphic Encryption (HE) to eliminate the threat of data leakage in cloud systems. By accelerating HE with FPGA and HBM integration, this contribution in the dissertation demonstrates that secure computations could be performed directly on encrypted data, significantly reducing the risk of exposing plaintext data during processing. This approach preserves user privacy while allowing for complex computations, a necessary feature in fields such as healthcare, finance, and government services where sensitive data is frequently processed. The proposed scheme HBMorphic is able to achieve up to 438× improvement of the bottleneck of the TFHE scheme.

Lastly, the dissertation tackled the issue of fault injection in multi-tenant FPGAs. In these environments, an attacker can deliberately introduce faults into computations, thereby compromising the integrity of the system. The research focused on enhancing the security of FPGA reconfigurable fabrics, which are particularly vulnerable in multi-tenant settings where power resources are shared among users. The proposed solutions involved a combined approach with offline and online defense mechanisms. Meta-Scanner is the offline defense mechanism. It uses a 3-class classifier to detect the level of the threat of a tenant design by analyzing its metadata. High-threat designs are banned, mid-threat designs are uploaded but targeted by the online mechanism in case they start an attack and low-threat designs are uploaded without any security measures. LoopBreaker the online mechanism turns all the interconnects of a suspicious tenant's area into high impedance in case an attack is detected. Meta-Scanner is capable of detecting the correct class of the designs with an accuracy of 98%. Moreover, LoopBreaker needs only 1.5 μs which is fast enough to stop all the mid-threat attacks.

7.2 Future Work

Building upon the findings of this dissertation, several directions for future research can be identified. As the technological landscape evolves, it is essential that security mechanisms adapt to new challenges while maintaining system efficiency.

1. Enhancing PUF-based Authentication: While the current PUF-based authentication methods have proven effective, future research can focus on improving their reliability and the ability to use them for key generation. Moreover, studying the possibility of using PUFs in mutual attestation and peer to peer authentication can be further explored. Finally, further investigation into the long-term reliability of PUFs in real-world conditions, such as temperature fluctuations, noisy environments, and aging effects, will be critical in ensuring their continued viability for security applications.

2. Improving Covert-Channel Mitigation Techniques: The mitigation strategies for covert-channel attacks presented in this dissertation provide a base, but they can be further optimized. Future research could explore the integration of machine learning techniques for the automatic detection of anomalous behavior indicative of a covert-channel attack. Additionally, real-time monitoring systems that adjust resource allocation dynamically in response to potential threats could enhance the security of multi-FPGA-MPSoCs. A promising direction would involve creating adaptive algorithms capable of learning and predicting attack vectors before they occur, thus offering preemptive protection against covert channel threats.

3. Optimizing FPGA-Accelerated Homomorphic Encryption: While this dissertation demonstrated the feasibility of FPGA-accelerated HE, further improvements in performance and energy efficiency are possible. Future research could focus on optimizing the hardware architecture to support more homomorphic operations, which would broaden the applicability of HE to more varied and demanding cloud applications. Additionally, efforts to reduce the power consumption of FPGA-accelerated HE systems will be important for making these solutions practical and sustainable.

4. Advanced Fault Injection Detection Mechanism: The current fault injection detection mechanisms can be expanded through the use of AI. Specially for online countermeasures, analyzing patterns of the power disturbance can detect anomalies and allocate the malicious tenant.

7.3 Final Remarks

This dissertation presents significant advances in improving the security of accelerated cloud systems. The results presented across four key contributions PUF-based authentication, covert-channel attack mitigation, homomorphic encryption, and fault injection prevention—demonstrate that these systems can be both secure and efficient. By addressing these critical challenges, this dissertation establishes a path to a more secure cloud systems capable of supporting the growing demands of modern computing applications.

Bibliography

[1] Hassan Nassar, Lars Bauer, and Jörg Henkel. "ANV-PUF: Machine-Learning-Resilient NVM-Based Arbiter PUF". In: *ACM Trans. Embed. Comput. Syst.* 22.5s (2023). ISSN: 1539-9087. DOI: 10.1145/3609388. URL: https://doi.org/10.1145/3609388.

[2] Hassan Nassar, Lars Bauer, and Jörg Henkel. "CaPUF: Cascaded PUF Structure for Machine Learning Resiliency". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.11 (2022), pp. 4349–4360. DOI: 10.1109/TCAD.2022.3197539.

[3] Hassan Nassar, Lars Bauer, and Jörg Henkel. "Effects of Runtime Reconfiguration on PUFs Implemented as FPGA-Based Accelerators". In: *IEEE Embedded Systems Letters* 15.4 (2023), pp. 174–177. DOI: 10.1109/LES.2023.3299214.

[4] Hassan Nassar, Philipp Machauer, Lars Bauer, Dennis Gnad, Mehdi Tahoori, and Jörg Henkel. "DoS-FPGA: Denial of Service on Cloud FPGAs via Coordinated Power Hammering". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. accepted. 2024.

[5] Hassan Nassar, Philipp Machauer, Dennis R. E. Gnad, Lars Bauer, Mehdi B. Tahoori, and Jörg Henkel. "Covert-Hammer: Coordinating Power-Hammering on Multi-tenant FPGAs via Covert Channels". In: *ACM/SIGDA ISFPGA*. Poster. 2024. DOI: 10.1145/3626202.3637613.

[6] Hassan Nassar, Simon Pankner, Lars Bauer, and Jörg Henkel. "Late Breaking Results: Configurable Ring Oscillators as a Side-Channel Countermeasure". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–2. DOI: 10.1109/DAC56929.2023.10247786.

[7] Hassan Nassar, Jeferson Gonzalez-Gomez, Varun Manjunath, Lars Bauer, and Jörg Henkel. "Through Fabric: A Cross-world Thermal Covert Channel on TEE-enhanced FPGA-MPSoC Systems". In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. accepted. 2025. DOI: 10.1145/3658617.3697767.

[8] Hassan Nassar, Lars Bauer, and Jörg Henkel. "HBMorphic: FHE Acceleration via HBM-Enabled Recursive Karatsuba Multiplier on FPGA". In: *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2024. DOI: 10.1109/FCCM60383.2024.00038.

[9] Hassan Nassar, Lars Bauer, and Jörg Henkel. "Turbo-FHE: Accelerating Fully Homomorphic Encryption with FPGA and HBM Integration". In: *IEEE Design and Test Magazine* (2025). accepted. DOI: 10.1109/MDAT.2025.3527368.

[10] Hassan Nassar, Jonas Krautter, Lars Bauer, Dennis Gnadd, Mehdi Tahoori, and Jörg Henkel. "Meta-Scanner: Detecting Fault Attacks via Scanning FPGA Designs Metadata". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024). DOI: 10.1109/TCAD.2024.3443769.

[11] Hassan Nassar, Hanna AlZughbi, Dennis Gnad, Lars Bauer, Mehdi Tahoori, and Jörg Henkel. "LoopBreaker: Disabling Interconnects to Mitigate Voltage-Based Attacks in Multi-Tenant FPGAs". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2021. DOI: 10.1109/ICCAD51958.2021.9643485.

[12] Hassan Nassar, Lars Bauer, and Jörg Henkel. "TiVaPRoMi: Time-Varying Probabilistic Row-Hammer Mitigation". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Virtual Event: IEEE, 2021, pp. 1711–1716. DOI: 10.23919/DATE51398.2021.9473912.

[13] Hassan Nassar, Rafik Youssef, Lars Bauer, and Jörg Henkel. "Supporting Dynamic Control-Flow Execution for Runtime Reconfigurable Processors". In: *2023 International Conference on Microelectronics (ICM)*. 2023, pp. 184–189. DOI: 10.1109/ICM60448.2023.10378905.

[14] Nils Hölscher, Christian Hakert, Hassan Nassar, Kuan-Hsun Chen, Lars Bauer, Jian-Jia Chen, and Jörg Henkel. "Memory Carousel: LLVM-Based Bitwise Wear-Leveling for Non-Volatile Main Memory". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.8 (2022), pp. 1–13. DOI: 10.1109/TCAD.2022.3228897.

[15] Lokesh Siddhu, Hassan Nassar, Lars Bauer, Christian Hakert, Nils Hölscher, Jian-Jia Chen, and Joerg Henkel. "Swift-CNN: Leveraging PCM Memory's Fast Write Mode to Accelerate CNNs". In: *IEEE Embedded Systems Letters* 15.4 (2023), pp. 234–237. DOI: 10.1109/LES.2023.3298742.

[16] Lilas Alrahis, Hassan Nassar, Jonas Krautter, Dennis Gnad, Lars Bauer, Jörg Henkel, and Mehdi Tahoori. "MaliGNNoma: GNN-Based Malicious Circuit Classifier for Secure Cloud FPGAs". In: *IEEE HOST*. 2024. DOI: 10.1109/HOST55342.2024.10545411.

[17] Jeferson Gonzalez-Gomez, Hassan Nassar, Lars Bauer, and Jörg Henkel. "LightFAt: Mitigating Control-Flow Explosion via Lightweight PMU-Based Control-Flow Attestation". In: *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2024, pp. 222–226. DOI: 10.1109/HOST55342.2024.10545348.

[18] Lars Bauer, Jürgen Becker, Jörg Henkel, Fabian Lesniak, and Hassan Nassar. "Adaptive Application-Specific Invasive Micro-Architectures". In: *Invasive Computing*. FAU University Pres, 2022.

[19] Jörg Henkel, Lokesh Siddhu, Hassan Nassar, Lars Bauer, Jian-Jia Chen, Christian Hakert, Tristan Seidl, Kuan-Hsun Chen, Xiaobo Sharon Hu, Mengyuan Li, Chia-Lin Yang, and Ming-Liang Wei. "(Invited Paper) Co-Designing NVM-based Systems for Machine Learning and In-memory Search Applications". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. accepted. 2024.

[20] Lars Bauer, Hassan Nassar, Nadir Khan, Jürgen Becker, and Jörg Henkel. "Machine-Learning-based Side-Channel Attack Detection for FPGA SoCs". In: *IEEE Transactions on Circuits and Systems for Artificial Intelligence* (2024). DOI: 10.1109/TCASAI.2024.3483118.

[21] Simon Bothe, Hassan Nassar, Lars Bauer, and Jörg Henkel. "Client-Server Framework for FPGA Acceleration of Fan-Vercauteren-Based Homomorphic Encryption". In: *2024 International Conference on Microelectronics (ICM)*. 2024, pp. 1–5. DOI: 10.1109/ICM63406.2024.10815906.

[22] Mina Ibrahim, Martel Shokry, Lokesh Siddhu, Lars Bauer, Hassan Nassar, and Jörg Henkel. "An FPGA-Based RISC-V Instruction Set Extension and Memory Controller for Multi-Level Cell NVM". In: *2024 International Conference on Microelectronics (ICM)*. 2024, pp. 1–6. DOI: 10.1109/ICM63406.2024.10815826.

[23] Jayeeta Chaudhuri, Hassan Nassar, Dennis Gnad, Jörg Henkel, Mehdi Tahoori, and Krishnendu Chakrabarty. "Hacking the Fabric: Targeting Partial Reconfiguration for Fault Injection in FPGA Fabrics". In: *Asian Test Symposium*. accepted. 2024.

[24] Hassan Nassar, Ming-Liang Wei, Chia-Lin Yang, Jörg Henke, and Kuan-Hsun Chen. "REAP-NVM: Resilient Endurance-Aware NVM-based PUF against Learning-based Attacks". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. accepted as extended abstract. 2025.

[25] Can Lehmann, Lars Bauer, Hassan Nassar, Heba Khdr, and Jörg Henkel. "Hardware/Software Co-Analysis for Worst Case Execution Time Bounds". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. accepted as extended abstract. 2025.

[26] *7 Series FPGAs Configuration UG (v1.15)*. Xilinx. 2022.

[27] Emrah Abtioglu et al. "Partially reconfigurable IP protection system with ring oscillator based physically unclonable functions". In: *Proceedings - 2017 1st New Generation of CAS, NGCAS 2017* (2017), pp. 65–68. DOI: `10.1109/NGCAS.2017.66`.

[28] Mohammad Abdullah Al Faruque, Thomas Ebi, and Jorg Henkel. "Configurable links for runtime adaptive on-chip communication". In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. 2009, pp. 256–261. DOI: `10.1109/DATE.2009.5090667`.

[29] Md Mahbub Alam et al. "RAM-Jam: Remote Temperature and Voltage Fault Attack on FPGAs using Memory Collisions". In: *Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2019.

[30] Rashid Ali et al. "A Reconfigurable Arbiter MPUF With High Resistance Against Machine Learning Attack". In: *IEEE Transactions on Magnetics* 57.10 (2021), pp. 1–7. DOI: `10.1109/TMAG.2021.3102838`.

[31] Amazon Web Services (AWS). *EC2 F1 Instances*. Amazon.com, Inc. 2021. URL: `https://aws.amazon.com/ec2/instance-types/f1/` (visited on 11/18/2021).

[32] Inc. Amazon.com. *AWS Marketplace*. 2022. URL: `https://aws.amazon.com/marketplace` (visited on 11/23/2022).

[33] Nikolaos Athanasios Anagnostopoulos et al. "Securing IoT Devices Using Robust DRAM PUFs". In: *2018 Global Information Infrastructure and Networking Symposium (GIIS)*. 2018, pp. 1–5. DOI: `10.1109/GIIS.2018.8635789`.

[34] N. Nalla Anandakumar, Mohammad S. Hashmi, and Somitra Kumar Sanadhya. “Efficient and Lightweight FPGA-based Hybrid PUFs with Improved Performance”. In: *Microprocessors and Microsystems* 77 (2020), p. 103180. ISSN: 01419331. DOI: 10.1016/j.micpro.2020.103180. URL: https://doi.org/10.1016/j.micpro.2020.103180.

[35] Thomas E Anderson. “The performance of spin lock alternatives for shared-memory multiprocessors.” In: *IEEE TPDS* (1990).

[36] Florian Bache, Christina Plump, and Tim Güneysu. “Confident leakage assessment - A side-channel evaluation framework based on confidence intervals”. In: *DATE*. 2018, pp. 1117–1122.

[37] Kelly Baker. “Embedded Nonvolatile Memories: A Key Enabler for Distributed Intelligence”. In: *2012 4th IEEE International Memory Workshop*. Milano, Italy: IEEE, 2012, pp. 1–4. DOI: 10.1109/IMW.2012.6213637.

[38] Lars Bauer, Muhammad Shafique, and Jörg Henkel. “Run-time instruction set selection in a transmutable embedded processor”. In: *Proceedings of the 45th Annual Design Automation Conference*. DAC ’08. Anaheim, California: Association for Computing Machinery, 2008, pp. 56–61. ISBN: 9781605581156. DOI: 10.1145/1391469.1391486. URL: https://doi.org/10.1145/1391469.1391486.

[39] Georg T. Becker. “On the Pitfalls of Using Arbiter-PUFs as Building Blocks”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.8 (2015), pp. 1295–1307. ISSN: 02780070. DOI: 10.1109/TCAD.2015.2427259.

[40] Christophe Bobda et al. “The future of FPGA acceleration in datacenters and the cloud”. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 15.3 (2022), pp. 1–42.

[41] Leonid Bolotnyy and Gabriel Robins. “Physically unclonable function -based security and privacy in RFID systems”. In: *Proceedings - Fifth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2007* September (2007), pp. 211–218. DOI: 10.1109/PERCOM.2007.26.

[42] Lilian Bossuet and Carlos Andres Lara-Nino. “Advanced Covert-Channels in Modern SoCs”. In: *IEEE Hardware Oriented Security and Trust (HOST)*. 2023, pp. 80–88. DOI: 10.1109/HOST55118.2023.10133626.

[43] Andrew Boutros et al. "Neighbors From Hell: Voltage Attacks Against Deep Learning Accelerators on Multi-Tenant FPGAs". In: *International Conference on Field-Programmable Technology (ICFPT)*. 2020, pp. 103–111. DOI: 10.1109/ICFPT51103.2020.00023.

[44] F. Brglez, D. Bryan, and K. Kozminski. "Combinational profiles of sequential benchmark circuits". In: *ISCAS*. 1989, 1929–1934 vol.3.

[45] Batuhan Bulut, Saliha Nur Bicakci, and Ismail San. "HW/SW Co-Design of TFHE Homomorphic OR Gate via NTT-based Polynomial Multiplication on a programmable SoC". In: *Innovations in Intelligent Systems and Applications Conference*. New Jersey, USA: IEEE, 2022, pp. 1–6.

[46] Geoffrey W. Burr et al. "Phase change memory technology". In: *Journal of Vacuum Science & Technology B* 28.2 (2010), pp. 223–262. DOI: 10.1116/1.3301579. eprint: https://doi.org/10.1116/1.3301579. URL: https://doi.org/10.1116/1.3301579.

[47] Jeff Calhoun et al. "Physical Unclonable Function (PUF)-Based e-Cash Transaction Protocol (PUF-Cash)". In: *Cryptography* 3.3 (2019). ISSN: 2410-387X. DOI: 10.3390/cryptography3030018. URL: https://www.mdpi.com/2410-387X/3/3/18.

[48] Zhen Cao et al. "Reconfigurable Physical Unclonable Function Based on Spin-Orbit Torque Induced Chiral Domain Wall Motion". In: *IEEE Electron Device Letters* 42.4 (2021), pp. 597–600. DOI: 10.1109/LED.2021.3057638.

[49] Jayeeta Chaudhuri and Krishnendu Chakrabarty. "Detection of Malicious FPGA Bitstreams using CNN-Based Learning*". In: *ETS*. 2022.

[50] Jayeeta Chaudhuri and Krishnendu Chakrabarty. "Diagnosis of Malicious Bitstreams in Cloud Computing FPGAs". In: *Transactions on Computer Aided Design of Integrated Circuits & Systems (TCAD)* (2023).

[51] S. Chen et al. "Thermal Covert Channels Leveraging Package-on-Package DRAM". In: *IEEE (TrustCom/BigDataSE)*. 2019.

[52] Ying-Chen Chen et al. "Dynamic conductance characteristics in HfOx-based resistive random access memory". In: *RSC Adv.* 7 (21 2017), pp. 12984–12989. DOI: 10.1039/C7RA00567A. URL: http://dx.doi.org/10.1039/C7RA00567A.

[53] Ilaria Chillotti, Mark Joe, and Pascal Paillier. *Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks.* Zama AI, 2020.

[54] Ilaria Chillotti et al. *TFHE: Fast Fully Homomorphic Encryption over the Torus.* Cryptology ePrint Archive, Paper 2018/421. 2018.

[55] Haehyun Cho et al. "Prime+Count: Novel Cross-World Covert Channels on ARM TrustZone". In: *Annual Computer Security Applications Conference.* 2018.

[56] Marvin Damschen, Lars Bauer, and Jörg Henkel. "CoRQ: Enabling Runtime Reconfiguration under WCET Guarantees for Real-Time Systems". In: *IEEE Embedded Systems Letters* 9.3 (2017), pp. 77–80.

[57] Ghada Dessouky, Ahmad-Reza Sadeghi, and Shaza Zeitouni. "SoK: Secure FPGA Multi-Tenancy in the Cloud: Challenges and Opportunities". In: *2021 IEEE European Symposium on Security and Privacy (EuroS&P).* 2021, pp. 487–506. DOI: 10.1109/EuroSP51992.2021.00040.

[58] Dionysios Diamantopoulos et al. "Acceleration-as-a-μService: A Cloud-native Monte-Carlo Option Pricing Engine on CPUs, GPUs and Disaggregated FPGAs". In: *2021 IEEE 14th International Conference on Cloud Computing (CLOUD).* 2021, pp. 726–729. DOI: 10.1109/CLOUD53861.2021.00096.

[59] Elena Dubrova et al. "CRC-PUF: A Machine Learning Attack Resistant Lightweight PUF Construction". In: *Proceedings - 4th IEEE European Symposium on Security and Privacy Workshops, EUROS and PW 2019* (2019), pp. 264–271. DOI: 10.1109/EuroSPW.2019.00036.

[60] Thomas Ebi, Mohammad Abdullah Al Faruque, and Jörg Henkel. "TAPE: thermal-aware agent-based power economy for multi/many-core architectures". In: *Proceedings of the 2009 International Conference on Computer-Aided Design.* ICCAD '09. San Jose, California: Association for Computing Machinery, 2009, pp. 302–309.

[61] Thomas Ebi et al. "Economic learning for thermal-aware power budgeting in many-core architectures". In: *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis.* CODES+ISSS '11. Taipei, Taiwan: Association for Computing Machinery, 2011, pp. 189–196. ISBN: 9781450307154.

[62] Solutions for Economists. *VFI - Accelerator FPGA*. 2022. URL: https://aws.amazon.com/marketplace/pp/prodview-fboutdmufjkum (visited on 11/23/2022).

[63] Ken Eguro and Ramarathnam Venkatesan. "FPGAs for trusted cloud computing". In: *22nd International Conference on Field Programmable Logic and Applications (FPL)*. 2012, pp. 63–70. DOI: 10.1109/FPL.2012.6339242.

[64] Rana Elnaggar et al. "Learning Malicious Circuits in FPGA Bitstreams". In: *IEEE TCAD* (2022).

[65] R. Ernst, J. Henkel, and T. Benner. "Hardware-software cosynthesis for microcontrollers". In: *IEEE Design & Test of Computers* 10.4 (1993), pp. 64–75.

[66] Suhaib A. Fahmy, Kizheppatt Vipin, and Shanker Shreejith. "Virtualized FPGA Accelerators for Efficient Cloud Computing". In: *International Conference on Cloud Computing Technology and Science (CloudCom)*. 2015, pp. 430–435.

[67] Mohammad Abdullah Al Faruque, Thomas Ebi, and Jorg Henkel. "Runtime adaptive on-chip communication scheme". In: *2007 IEEE/ACM International Conference on Computer-Aided Design*. 2007, pp. 26–31. DOI: 10.1109/ICCAD.2007.4397239.

[68] Anis Fellah-Touta, Lilian Bossuet, and Carlos Andres Lara-Nino. "Combined Internal Attacks on SoC-FPGAs: Breaking AES with Remote Power Analysis and Frequency-based Covert Channels". In: *EuroS&PW*. 2023.

[69] Michael J. Foster, Marcin Łukowiak, and Stanisław Radziszowski. "Flexible HLS-Based Implementation of the Karatsuba Multiplier Targeting Homomorphic Encryption Schemes". In: *2019 MIXDES - 26th International Conference "Mixed Design of Integrated Circuits and Systems"*. 2019, pp. 215–220. DOI: 10.23919/MIXDES.2019.8787132.

[70] Yansong Gao et al. "Emerging Physical Unclonable Functions With Nanotechnology". In: *IEEE Access* 4 (2016), pp. 61–80. DOI: 10.1109/ACCESS.2015.2503432.

[71] Yansong Gao et al. "PUF-FSM: A Controlled Strong PUF". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.5 (2018), pp. 1104–1108. ISSN: 02780070. DOI: 10.1109/TCAD.2017.2740297. arXiv: 1701.04137.

[72] Fernando García-Redondo et al. "SPICE Compact Modeling of Bipolar/Unipolar Memristor Switching Governed by Electrical Thresholds". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 63.8 (2016), pp. 1255–1264. DOI: 10.1109/TCSI.2016.2564703.

[73] Navyata Gattu et al. "Power side channel attack analysis and detection". In: *ICCAD*. 2020, pp. 1–7.

[74] Serhan Gener et al. "An FPGA-based Programmable Vector Engine for Fast Fully Homomorphic Encryption over the Torus". In: *Secure and Private Systems for Machine Learning*. New York, USA: ACM, 2021, pp. 1–7.

[75] Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices". In: *ACM Symposium on Theory of Computing*. 2009.

[76] Hamid Reza Ghaeini et al. "PAtt: Physics-based Attestation of Control Systems". In: *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. Chaoyang District, Beijing: USENIX Association, Sept. 2019, pp. 165–180. ISBN: 978-1-939133-07-6. URL: https://www.usenix.org/conference/raid2019/presentation/ghaeini.

[77] Ilias Giechaskiel, Ken Eguro, and Kasper B. Rasmussen. "Leakier Wires: Exploiting FPGA Long Wires for Covert- and Side-Channel Attacks". In: *ACM TRETS* (2019).

[78] Ilias Giechaskiel, Kasper Rasmussen, and Jakub Szefer. "Reading between the dies: Cross-SLR covert channels on multi-tenant cloud FPGAs". In: *International Conference on Computer Design (ICCD)*. IEEE. 2019.

[79] Ilias Giechaskiel, Shanquan Tian, and Jakub Szefer. "Cross-VM Covert- and Side-Channel Attacks in Cloud FPGAs". In: *ACM TRETS* (2022).

[80] Dennis R E Gnad, Fabian Oboril, and Mehdi B Tahoori. "Voltage drop-based fault attacks on FPGAs using valid bitstreams". In: *Int. Conf. on FPL and Appl.* 2017, pp. 1–7.

[81] Dennis R E Gnad et al. "Remote Electrical-level Security Threats to Multi-Tenant FPGAs". In: *IEEE Design & Test* 37.2 (2020), pp. 111–119.

[82] Dennis R. E. Gnad et al. "Voltage-Based Covert Channels Using FPGAs". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES))* (2021).

[83] Dennis RE Gnad et al. "Stealthy logic misuse for power analysis attacks in multi-tenant FPGAs". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 1012–1015.

[84] Jeferson Gonzalez-Gomez, Lars Bauer, and Jörg Henkel. "Cache-Based Side-Channel Attack Mitigation for Many-Core Distributed Systems via Dynamic Task Migration". In: *IEEE TIFS* (2023).

[85] Jeferson Gonzalez-Gomez et al. "Smart Detection of Obfuscated Thermal Covert Channel Attacks in Many-core Processors". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. DOI: 10.1109/DAC56929.2023.10247844.

[86] Jeferson González-Gómez et al. "The First Concept and Real-world Deployment of a GPU-based Thermal Covert Channel: Attack and Countermeasures". In: *DATE*. 2023.

[87] Mathieu Gross, Robert Kunzelmann, and Georg Sigl. *CPU to FPGA Power Covert Channel in FPGA-SoCs*. Cryptology ePrint Archive. 2023.

[88] Changhee Hahn, Hyunsoo Kwon, and Junbeom Hur. "Toward Trustworthy Delegation: Verifiable Outsourced Decryption with Tamper-Resistance in Public Cloud Storage". In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 2018, pp. 920–923. DOI: 10.1109/CLOUD.2018.00136.

[89] Basma Hajri et al. "A Lightweight Reconfigurable RRAM-based PUF for Highly Secure Applications". In: *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. Esrin, Italy: IEEE, 2020, pp. 1–4. DOI: 10.1109/DFT50435.2020.9250829.

[90] Nemat H. El-Hassan, T. Nandha Kumar, and Haider Abbas F. Almurib. "Improved SPICE model for Phase Change Memory cell". In: *2014 5th International Conference on Intelligent and Advanced Systems (ICIAS)*. Petronas, Malaysia: IEEE, 2014, pp. 1–6. DOI: 10.1109/ICIAS.2014.6869529.

[91] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. New York, NY, USA: Springer, 2009.

[92] J. Henkel. "A low power hardware/software partitioning approach for core-based embedded systems". In: *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*. 1999, pp. 122–127.

[93] J. Henkel and Yanbing Li. “Energy-conscious HW/SW-partitioning of embedded systems: a case study on an MPEG-2 encoder”. In: *Proceedings of the Sixth International Workshop on Hardware/Software Codesign. (CODES/CASHE’98).* 1998, pp. 23–27. DOI: 10.1109/HSC.1998.666233.

[94] J. Henkel, W. Wolf, and S. Chakradhar. “On-chip networks: a scalable, communication-centric embedded system design paradigm”. In: *17th International Conference on VLSI Design. Proceedings.* 2004, pp. 845–851. DOI: 10.1109/ICVD.2004.1261037.

[95] Jörg Henkel et al. “Invasive manycore architectures”. In: *17th Asia and South Pacific Design Automation Conference.* 2012, pp. 193–200.

[96] D. Herrmann, J. Henkel, and R. Ernst. “An approach to the adaptation of estimated cost parameters in the COSYMA system”. In: *Third International Workshop on Hardware/Software Codesign.* 1994, pp. 100–107. DOI: 10.1109/HSC.1994.336718.

[97] *Heterogeneous Accelerated Compute Clusters (HACC).* URL: https://www.amd-haccs.io/index.html.

[98] Yohei Hori et al. “Pseudo-LFSR PUF: A compact, efficient and reliable physical unclonable function”. In: *Proceedings - 2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011* (2011), pp. 223–228. DOI: 10.1109/ReConFig.2011.72.

[99] Gabriel Hospodar, Roel Maes, and Ingrid Verbauwhede. “Machine learning attacks on 65nm Arbiter PUFs: Accurate modeling poses strict bounds on usability”. In: *WIFS 2012 - Proceedings of the 2012 IEEE International Workshop on Information Forensics and Security* (2012), pp. 37–42. DOI: 10.1109/WIFS.2012.6412622.

[100] Huxelerate. *Hugenomic Nanopolish.* 2022. (Visited on 11/23/2022).

[101] Taras Iakymchuk, Maciej Nikodem, and Krzysztof Kepa. “Temperature-based covert channel in FPGA systems”. In: *ReCoSoC.* 2011.

[102] *Introduction of F1 development environment.* Amazon Web Servises (AWS). 2021.

[103] Ihsan Jabbar and Saad Najim Alsaad. “Design and Implementation of Secure Remote e-Voting System Using Homomorphic Encryption”. In: *International Journal of Network Security* 19.5 (2017), pp. 694–703.

[104] Peter Jamieson et al. "Benchmarking and Evaluating Reconfigurable Architectures Targeting the Mobile Domain". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES))* (2010).

[105] Aditya Japa et al. "Hardware Security Exploiting Post-CMOS Devices: Fundamental Device Characteristics, State-of-the-Art Countermeasures, Challenges and Roadmap". In: *IEEE Circuits and Systems Magazine* 21.3 (2021), pp. 4–30. DOI: 10.1109/MCAS.2021.3092532.

[106] Soundarya Jayaraman, Bingyi Zhang, and Viktor Prasanna. "Hypersort: High-performance Parallel Sorting on HBM-enabled FPGA". In: *2022 International Conference on Field-Programmable Technology (ICFPT)*. New Jersey, USA: IEEE, 2022, pp. 1–11.

[107] Yang Jie and Liza Lin. "Alibaba Falls Victim to Chinese Web Crawler in Large Data Leak". In: *Wall Street Journal* (2021).

[108] Chenglu Jin et al. "FPGA Implementation of a Cryptographically-Secure PUF Based on Learning Parity with Noise". In: *Cryptography* 1.3 (2017), p. 23. ISSN: 2410-387X. DOI: 10.3390/cryptography1030023.

[109] Kevin B Johnson et al. "Precision Medicine, AI, and the Future of Personalized Health Care". In: *Clinical Translational Science* (2020).

[110] Michael Guilherme Jordan et al. "MUTECO: A Framework for Collaborative Allocation in CPU-FPGA Multi-tenant Environments". In: *SBCCI*. 2021.

[111] Moein Khazraee et al. "Specializing a Planet's Computation: ASIC Clouds". In: *IEEE Micro* 37.3 (2017), pp. 62–69. DOI: 10.1109/MM.2017.49.

[112] Dayoung Kim et al. "Selected Bit-Line Current PUF: Implementation of Hardware Security Primitive Based on a Memristor Crossbar Array". In: *IEEE Access* 9 (2021), pp. 120901–120910. DOI: 10.1109/ACCESS.2021.3108534.

[113] Myungsun Kim and Benjamin Z. Kim. "An experimental study of encrypted polynomial arithmetics for private set operations". In: *Journal of Communications and Networks* 19.5 (2017), pp. 431–441. DOI: 10.1109/JCN.2017.000075.

[114] Stephan Kleber et al. "Design of the Secure Execution PUF-based Processor (SEPP)". In: *Workshop on Trustworthy Manufacturing and Utilization of Secure Devices, TRUDEVICE 2015* 2 (2015), pp. 1–5.

[115] Stephan Kleber et al. “Secure Code Execution: A Generic PUF-Driven System Architecture”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11060 LNCS (2018), pp. 25–46. ISSN: 16113349. DOI: 10.1007/978-3-319-99136-8_2.

[116] Sebastian Kobbe et al. “DistRM: distributed resource management for on-chip many-core systems”. In: *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis.* CODES+ISSS ’11. Taipei, Taiwan: Association for Computing Machinery, 2011, pp. 119–128. ISBN: 9781450307154. DOI: 10.1145/2039370.2039392. URL: https://doi.org/10.1145/2039370.2039392.

[117] Yuichi Komano et al. “Single-Round Pattern Matching Key Generation Using Physically Unclonable Function”. In: *Sec. and Commun. Netw.* 2019 (Jan. 2019). ISSN: 1939-0114. DOI: 10.1155/2019/1719585. URL: https://doi.org/10.1155/2019/1719585.

[118] Joonho Kong et al. “PUFatt: Embedded platform attestation based on novel processor-based PUFs”. In: *Design Automation Conf.* 2014.

[119] J. Krautter, D. R. E. Gnad, and M. B. Tahoori. “FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES”. In: *Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* 3 (2018). ISSN: 2569-2925.

[120] Jonas Krautter, Dennis R. E. Gnad, and Mehdi B. Tahoori. “Remote and Stealthy Fault Attacks on Virtualized FPGAs”. In: *DATE.* 2021.

[121] Jonas Krautter, Dennis R. E. Gnad, and Mehdi B. Tahoori. “Mitigating Electrical-level Attacks towards Secure Multi-Tenant FPGAs in the Cloud”. In: *ACM TRETS* (2019).

[122] Jonas Krautter et al. “Active fences against voltage-based side channels in multi-tenant FPGAs”. In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* IEEE. 2019, pp. 1–8.

[123] Sandeep S. Kumar et al. “Extended abstract: The butterfly PUF protecting IP on every FPGA”. In: *2008 IEEE International Workshop on Hardware-Oriented Security and Trust.* Anaheim Convention Center, CA: IEEE, 2008, pp. 67–70. DOI: 10.1109/HST.2008.4559053.

[124] Klaus Kursawe et al. "Reconfigurable Physical Unclonable Functions - Enabling technology for tamper-resistant storage". In: *2009 IEEE International Workshop on Hardware-Oriented Security and Trust.* San Francisco, California, USA: IEEE, 2009, pp. 22–29. DOI: 10.1109/HST.2009.5225058.

[125] Tuan La et al. "Denial-of-Service on FPGA-based Cloud Infrastructures — Attack and Defense". In: *IACR TCHES* 2021 (2021).

[126] Tuan Minh La et al. "FPGADefender: Malicious Self-oscillator Scanning for Xilinx UltraScale+ FPGAs". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 13.3 (2020), pp. 1–31.

[127] Hyunsuk Lee et al. "Design and signal integrity analysis of high bandwidth memory (HBM) interposer in 2.5D terabyte/s bandwidth graphics module". In: *2015 IEEE 24th Electrical Performance of Electronic Packaging and Systems (EPEPS).* New Jersey, USA: IEEE, 2015, pp. 145–148. DOI: 10.1109/EPEPS.2015.7347149.

[128] Oliver Lenke et al. "PEPERONI: Pre-Estimating the Performance of Near-Memory Integration". In: *Proceedings of the International Symposium on Memory Systems.* MEMSYS '21. Washington DC, DC, USA: Association for Computing Machinery, 2022. ISBN: 9781450385701. DOI: 10.1145/3488423.3519329. URL: https://doi.org/10.1145/3488423.3519329.

[129] Dave Lewis. "iCloud Data Breach: Hacking And Celebrity Photos". In: *Forbes* (2014).

[130] Celia Li and Cungang Yang. "A Novice Group Sharing Method for Public Cloud". In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD).* 2018, pp. 966–969. DOI: 10.1109/CLOUD.2018.00147.

[131] WenLi Li and XiaoLi Wan. "An Analysis and Comparison for Public Cloud Technology and Market Development Trend in China". In: *2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom).* 2018, pp. 200–205. DOI: 10.1109/CSCloud/EdgeCom.2018.00043.

[132] Yanbing Li and J. Henkel. "A framework for estimating and minimizing energy dissipation of embedded HW/SW systems". In: *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175).* 1998, pp. 188–193. DOI: 10.1109/DAC.1998.724464.

[133] Ye Lin et al. “A Novel Lightweight PUFs Using Interconnect Line Mismatch for Hardware Security”. In: *2021 China Semiconductor Technology International Conference (CSTIC)*. 2021, pp. 1–2. DOI: 10.1109/CSTIC52283.2021.9461415.

[134] Xuyang Lu, Lingyu Hong, and Kaushik Sengupta. “CMOS Optical PUFs Using Noise-Immune Process-Sensitive Photonic Crystals Incorporating Passive Variations for Robustness”. In: *IEEE Journal of Solid-State Circuits* 53.9 (2018), pp. 2709–2721. DOI: 10.1109/JSSC.2018.2850941.

[135] Pieter Maene and Ingrid Verbauwhede. “Single-Cycle Implementations of Block Ciphers”. In: *Lightweight Cryptography for Security and Privacy*. 2016.

[136] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. “Lightweight secure PUFs”. In: *2008 IEEE/ACM International Conference on Computer-Aided Design*. 2008, pp. 670–673. DOI: 10.1109/ICCAD.2008.4681648.

[137] Ramya Jayaram Masti et al. “Thermal Covert Channels on Multi-Core Platforms”. In: *USENIX Security*. 2015.

[138] Kaspar Matas et al. “Power-hammering through Glitch Amplification – Attacks and Mitigation”. In: *FCCM*. 2020.

[139] Garrett McGrath and Paul R. Brenner. “Serverless Computing: Design, Implementation, and Performance”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. 2017, pp. 405–410. DOI: 10.1109/ICDCSW.2017.36.

[140] Vincent Migliore et al. “Fast polynomial arithmetic for Somewhat Homomorphic Encryption operations in hardware with Karatsuba algorithm”. In: *2016 International Conference on Field-Programmable Technology (FPT)*. 2016, pp. 209–212. DOI: 10.1109/FPT.2016.7929535.

[141] Vincent Migliore et al. “Hardware/Software Co-Design of an Accelerator for FV Homomorphic Encryption Scheme Using Karatsuba Algorithm”. In: *IEEE Transactions on Computers* 67.3 (2018), pp. 335–347. DOI: 10.1109/TC.2016.2645204.

[142] Ivan Miketic, Krithika Dhananjay, and Emre Salman. “Covert Channel Communication as an Emerging Security Threat in 2.5D/3D Integrated Systems”. In: *Sensors* (2023).

[143] Rich Miller. "How Machine Learning is Changing the Face of the Data Center". In: *Data Center Frontier* (2016).

[144] Nimish Mishra et al. "Time's a Thief of Memory". In: *Smart Card Research and Advanced Applications*. Ed. by Ileana Buhan and Tobias Schneider. 2023.

[145] Federico Nardi et al. "Resistive Switching by Voltage-Driven Ion Migration in Bipolar RRAM—Part I: Experimental Study". In: *IEEE Transactions on Electron Devices* 59.9 (2012), pp. 2461–2467. DOI: 10.1109/TED.2012.2202319.

[146] Phuong Ha Nguyen et al. "MXPUF: Secure PUF Design against State-of-the-art Modeling Attacks". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 572.

[147] Phuong Ha Nguyen et al. "The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019.4 (Aug. 2019), pp. 243–290. DOI: 10.13154/tches.v2019.i4.243-290. URL: https://tches.iacr.org/index.php/TCHES/article/view/8351.

[148] NIST. *CVE-2024-3094 Detail*. 2024. URL: https://nvd.nist.gov/vuln/detail/CVE-2024-3094.

[149] Rishab Nithyanand and John Solis. "A theoretical analysis: Physical unclonable functions and the software protection problem". In: *Proceedings - IEEE CS Security and Privacy Workshops, SPW 2012* May 2012 (2012), pp. 1–11. DOI: 10.1109/SPW.2012.16.

[150] Nafisa Noor and Helena Silva. "Phase Change Memory for Physical Unclonable Functions". In: *Applications of Emerging Memory Technology: Beyond Storage*. Ed. by Manan Suri. Singapore: Springer, 2020, pp. 59–91. ISBN: 978-981-13-8379-3.

[151] Nafisa Noor et al. "Reset Variability in Phase Change Memory for Hardware Security Applications". In: *IEEE Transactions on Nanotechnology* 20 (2021), pp. 75–82. DOI: 10.1109/TNANO.2020.3041400.

[152] *Official repository of the AWS EC2 FPGA Hardware and Software Development Kit*. 2023. URL: https://github.com/aws/aws-fpga (visited on 03/03/2023).

[153] *Partial Reconfiguration UG (v2018.1)*. Xilinx. 2018.

[154] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011).

[155] Ningduo Peng et al. “Query-biased preview over outsourced and encrypted data”. In: *Scientific World Journal* (2013).

[156] Koustubh Phalak et al. “Quantum PUF for Security and Trust in Quantum Computing”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11.2 (2021), pp. 333–342. DOI: 10.1109/JETCAS.2021.3077024.

[157] Joachim Pistorius et al. “Benchmarking method and designs targeting logic synthesis for FPGAs”. In: *International Workshop for Logic Synthesis (IWLS)*. 2007.

[158] George Provelengios, Daniel Holcomb, and Russell Tessier. “Mitigating Voltage Attacks in Multi-Tenant FPGAs”. In: *ACM TRETS* (2021).

[159] George Provelengios, Daniel Holcomb, and Russell Tessier. “Power Wasting Circuits for Cloud FPGA Attacks”. In: *Field-Programmable Logic and Applications (FPL)*. 2020.

[160] Mahmood Azhar Qureshi and Arslan Munir. “PUF-IPA: A PUF-based Identity Preserving Protocol for Internet of Things Authentication”. In: *2020 IEEE 17th Annual Consumer Communications and Networking Conference, CCNC 2020* (2020). DOI: 10.1109/CCNC46108.2020.9045264.

[161] Dylan Rankin et al. “FPGAs-as-a-service toolkit (FaaST)”. In: *IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*. IEEE. 2020, pp. 38–47.

[162] Martin Rapp et al. “MLCAD: A Survey of Research in Machine Learning for CAD Keynote Paper”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.10 (2022), pp. 3162–3181. DOI: 10.1109/TCAD.2021.3124762.

[163] Sven Rheindt et al. “NEMESYS: near-memory graph copy enhanced system-software”. In: *Proceedings of the International Symposium on Memory Systems*. MEMSYS ’19. Washington, District of Columbia, USA: Association for Computing Machinery, 2019, pp. 3–18.

[164] Sven Rheindt et al. “X-CEL: A Method to Estimate Near-Memory Acceleration Potential in Tile-Based MPSoCs”. In: *Architecture of Computing Systems – ARCS 2020*. Ed. by André Brinkmann et al. Cham: Springer International Publishing, 2020, pp. 109–123.

[165] R L Rivest, L Adleman, and M L Dertouzos. “On Data Banks and Privacy Homomorphisms”. In: *Foundations of Secure Computation, Academia Press* 1987 (1978).

[166] Enrico Rossi et al. “Preemption of the Partial Reconfiguration Process to Enable Real-Time Computing with FPGAs”. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11.2 (July 2018).

[167] Jihyeon Ryu, Keunok Kim, and Dongho Won. “A Study on Partially Homomorphic Encryption”. In: *International Conference on Ubiquitous Information Management and Communication.* 2023.

[168] Durga Prasad Sahoo, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. “Towards Ideal Arbiter PUF Design on Xilinx FPGA: A Practitioner’s Perspective”. In: *DSD.* 2015.

[169] Andre Schaller et al. “Intrinsic Rowhammer PUFs: Leveraging the Rowhammer effect for improved security”. In: *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST).* Mclean, Virginia, USA: IEEE, 2017, pp. 1–7. DOI: 10.1109/HST.2017.7951729.

[170] James C. Schatzman. “Accuracy of the Discrete Fourier Transform and the Fast Fourier Transform”. In: *SIAM Journal on Scientific Computing* 17.5 (1996), pp. 1150–1166.

[171] Falk Schellenberg et al. “An Inside Job: Remote power analysis attacks on FPGAs”. In: *Design, Automation and Test in Europe Conference & Exhibition (DATE).* 2018, pp. 1111–1116.

[172] Steffen Schulz, Ahmad-Reza Sadeghi, and Christian Wachsmann. “Short Paper: Lightweight Remote Attestation Using Physical Functions”. In: *Conf. on Wireless Network Security.* 2011.

[173] Muhammad Shafique et al. “A low latency generic accuracy configurable adder”. In: *Proceedings of the 52nd Annual Design Automation Conference.* DAC ’15. San Francisco, California: Association for Computing Machinery, 2015. ISBN: 9781450335201.

[174] K. Shivdikar et al. “Accelerating Polynomial Multiplication for Homomorphic Encryption on GPUs”. In: *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED).* Los Alamitos, CA, USA: IEEE Computer Society, 2022, pp. 61–72.

[175] Keeyoung Son et al. “Thermal Analysis of High Bandwidth Memory (HBM)-GPU Module considering Power Consumption”. In: *2023 IEEE Electrical Design of Advanced Packaging and Systems (EDAPS)*. New Jersey, USA: IEEE, 2023, pp. 1–3.

[176] Takeshi Sugawara et al. “Oscillator without a combinatorial loop and its threat to FPGA in data centre”. In: *IET Electronics Letters* (2019).

[177] G. Edward Suh and Srinivas Devadas. “Physical Unclonable Functions for Device Authentication and Secret Key Generation”. In: *2007 44th ACM/IEEE Design Automation Conference*. San Diego, California, USA: IEEE, 2007, pp. 9–14.

[178] SymbiFlow Team. *Project X-Ray*. 2018. URL: https://prjxray.readthedocs.io/en/latest.

[179] Weihang Tan et al. “High-Speed Modular Multiplier for Lattice-Based Cryptosystems”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 68.8 (2021), pp. 2927–2931.

[180] Yi Tang et al. “The Shift PUF: Technique for Squaring the Machine Learning Complexity of Arbiter-based PUFs: Work-in-Progress”. In: *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES* (2020), pp. 9–11.

[181] Jürgen Teich, Jörg Henkel, and Andreas Herkersdorf, eds. *Invasive Computing*. en. FAU University Press, 2022. DOI: 10.25593/978-3-96147-571-1. URL: https://open.fau.de/handle/openfau/20033.

[182] Shanquan Tian and Jakub Szefer. “Temporal thermal covert channels in cloud FPGAs”. In: *International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM/SIGDA. 2019.

[183] *UltraScale Architecture Configuration User Guide*. Xilinx. 2020.

[184] *VCU128 Board User Guide UG1302*. Xilinx, Inc. 2022.

[185] Flavio Vella et al. “GPU Computing in EGI Environment Using a Cloud Approach”. In: *2011 International Conference on Computational Science and Its Applications*. 2011, pp. 150–155. DOI: 10.1109/ICCSA.2011.61.

[186] *Vitis Tutorials: Hardware Acceleration XD099*. Xilinx, Inc. 2023.

[187] Antian Wang et al. “NoPUF: A Novel PUF Design Framework Toward Modeling Attack Resistant PUFs”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.6 (2021), pp. 2508–2521. ISSN: 15580806.

[188] Ziyu Wang et al. “Physical Unclonable Function Systems Based on Pattern Transfer of Fingerprint-Like Patterns”. In: *IEEE Electron Device Letters* 43.4 (2022), pp. 655–658. DOI: 10.1109/LED.2022.3154655.

[189] Oliver Willers. *MEMS sensors as physical unclonable functions*. 2019.

[190] Linjun Wu et al. “FLAM-PUF: A Response–Feedback-Based Lightweight Anti-Machine-Learning-Attack PUF”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.11 (2022), pp. 4433–4444. DOI: 10.1109/TCAD.2022.3197696.

[191] Zhifeng Xiao and Yang Xiao. “Security and Privacy in Cloud Computing”. In: *IEEE Communications Surveys & Tutorials* 15.2 (2013).

[192] Yang Yang et al. “FPGA Acceleration of Rotation in Homomorphic Encryption Using Dynamic Data Layout”. In: *International Conference on Field-Programmable Logic and Applications*. 2023, pp. 174–181.

[193] Yinghao Yang et al. “Poseidon: Practical Homomorphic Encryption Accelerator”. In: *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2023, pp. 870–881.

[194] Tian Ye, Rajgopal Kannan, and Viktor K. Prasanna. “FPGA Acceleration of Fully Homomorphic Encryption over the Torus”. In: *High Performance Extreme Computing Conference*. 2022.

[195] Bilgiday Yuce, Patrick Schaumont, and Marc Witteman. “Fault attacks on secure embedded software: Threats, design, and evaluation”. In: *Journal of Hardware and Systems Security* 2.2 (2018), pp. 111–130.

[196] Furqan Zahoor, Tun Zainal Azni Zulkifli, and Farooq Ahmad Khanday. “Resistive Random Access Memory (RRAM): an Overview of Materials, Switching Mechanism, Performance, Multilevel Cell (mlc) Storage, Modeling, and Applications”. In: *Nanoscale Research Letters* 15.1 (2020), p. 90. ISSN: 1556-276X. DOI: 10.1186/s11671-020-03299-9.

[197] *ZCU102 Evaluation Board User Guide*. Xilinx. 2023.

[198] Shaza Zeitouni et al. “Trusted Configuration in Cloud FPGAs”. In: *FCCM*. 2021.

[199] Jiliang Zhang et al. “CT PUF: Configurable Tristate PUF Against Machine Learning Attacks for IoT Security”. In: *IEEE Internet of Things Journal* 9.16 (2022), pp. 14452–14462.

[200] Le Zhang, Zhi Hui Kong, and Chip-Hong Chang. “PCKGen: A Phase Change Memory based cryptographic key generator”. In: *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*. Beijing, China: IEEE, 2013, pp. 1444–1447. DOI: `10.1109/ISCAS.2013.6572128`.

[201] Le Zhang et al. “Exploiting Process Variations and Programming Sensitivity of Phase Change Memory for Reconfigurable Physical Unclonable Functions”. In: *IEEE Transactions on Information Forensics and Security* 9.6 (2014), pp. 921–932. DOI: `10.1109/TIFS.2014.2315743`.

[202] Le Zhang et al. “Leakage-resilient memory-based physical unclonable function using phase change material”. In: *2014 International Carnahan Conference on Security Technology (ICCST)*. Rome, Italy: IEEE, 2014, pp. 1–6. DOI: `10.1109/CCST.2014.6987047`.

[203] Xian Zhang et al. “A novel PUF based on cell error rate distribution of STT-RAM”. In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. Macao, China: IEEE, 2016, pp. 342–347.

[204] Zhuangdi Zhu et al. “FPGA Resource Pooling in Cloud Computing”. In: *IEEE Transactions on Cloud Computing* (2021).