POLITECNICO
MILANO 1863

Department of Electronics, Information, and Bioengineering

Deep Learning Against Obfuscation: Boosting Side-Channel Analysis Across Segmentation, Attack, and Defense

Advisor:
Prof. Davide Zoni

Tutor:
Prof. Francesco Amigoni

Doctoral Candidate:
Davide Galli

Doctoral Program in Information Technology

Year 2026 - XXXVIII Cycle

"Cryptographic algorithms do not run on paper;
they run on silicon."

Abstract

Side-channel attacks exploit unintended information leakage emitted by cryptographic devices to extract sensitive data. Such attacks represent a critical threat to the security of modern computing platforms, especially in the context of pervasive IoT devices that continuously process sensitive data. This thesis addresses the challenges of evaluating and enhancing the security of digital circuits against side-channel attacks, focusing on realistic scenarios and reducing the costs and complexity of the analysis. To achieve this goal, it presents an open-source framework that supports the side-channel workflow end to end, combining deep-learning methods for segmentation and attack with hardware-assisted evaluation of countermeasures.

Isolating and aligning cryptographic operations within side-channel traces is crucial for successful attacks. This task becomes particularly challenging in real-world scenarios where noise and countermeasures can obscure the signals of interest. A primary contribution of this thesis is Hound, a deep-learning pipeline for advancing cryptographic operations segmentation in traces highly affected by hiding countermeasures, eliminating the need for trigger infrastructure. The novel Chameleon dataset is introduced publicly to provide a realistic benchmark for evaluating segmentation and attack methodologies.

Once segmented and isolated, side-channel traces can be analyzed to recover secret information. Classical attacks often struggle against traces protected by hiding techniques, which introduce significant temporal misalignment. This work explores attack strategies against dynamic frequency scaling countermeasures, introduces the DFS_DESYNCH dataset for benchmarking methodologies on highly desynchronized traces, and proposes Owl, a novel deep learning-assisted attack to tackle these challenges. Owl leverages deep learning to preprocess and realign traces, enabling effective key recovery even in the presence of significant temporal misalignment.

On the defense side, the thesis investigates the impact of run-time variability, such as dynamic voltage and frequency scaling, on side-channel leakage and attack resistance. It proposes Rabbit, a hardware module for fine-grained and random frequency changes on FPGAs, and demonstrates its effectiveness through extensive experimental campaigns.

Results identify frequency scaling as the most robust hiding technique among those inspected, while Rabbit demonstrates high resistance to the tested advanced attacks without compromising system performance.

All the experimental campaigns leverage JARVIS, a novel open-source hardware-software framework for side-channel research on FPGA-based IoT-class systems. JARVIS provides a complete environment for configuring, monitoring, and analyzing cryptographic operations, enabling researchers to deploy and assess countermeasures with high observability and control.

Overall, this thesis delivers a comprehensive workflow for side-channel analysis in realistic environments, combining open-source platforms, novel datasets, advanced segmentation and attack techniques, and innovative countermeasures. The contributions support the development of secure digital systems and provide valuable resources for the research community, advancing the state of the art in side-channel analysis.

Keywords: side-channel analysis, deep-learning, cryptography, security, AES, RISC-V, CNN

Contents

1 | Introduction

Computer security has become a paramount concern in contemporary everyday life. The pervasive nature of always-connected devices enables a wide variety of services, such as health monitoring, smart and interactive buildings, digital banking, and video surveillance. However, while such applications offer significant utility in modern society, they simultaneously introduce substantial challenges in guaranteeing the privacy and security of exchanged data, particularly when that data is personal, private, and sensitive.

Cryptography is at the core of ensuring security in any digital application. Indeed, it provides the foundation for secure data exchange by ensuring confidentiality, authenticity, and integrity, i.e., guaranteeing that data has not been tampered with, of the communicating parties. Traditionally, the security of these systems is believed to be guaranteed through the deployment of cryptographic primitives, e.g., RSA [RSA78], AES [oSND+23], and SHA-3 [Dwo15], alongside secure communication protocols like SSH [YL06] and TLS [RD08]. However, it has become increasingly evident that relying solely on the theoretical soundness and mathematical security of these cryptographic algorithms is, in many practical scenarios, insufficient. Vulnerabilities often arise from side-channel attacks (SCAs) (or analysis), which have emerged as one of the most significant and critical security threats to modern cryptographic implementations. Unlike attacks that target flaws in the cryptographic algorithms' theoretical specification, SCA exploits weaknesses and unintended information leakage in their physical implementation, even for well-established ciphers proven to be mathematically secure. SCA leverages information leakage caused by the dependency between data processed by a computing platform and observable environmental signals, known as side-channel signals. Such leaked information can manifest in various forms, including power consumption [KJJ99], electromagnetic emissions [QS01], and computational time [Koc96, KHF+19].

An effective analogy for understanding SCA is opening a safe without knowing the combination, relying only on a stethoscope to listen to the subtle sounds produced by the lock's mechanisms. In the same way, an attacker targeting an embedded device with SCA does not require direct access to the secret key, but instead carefully observes indirect physical

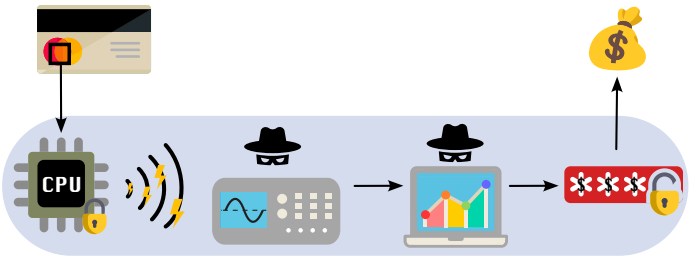


Figure 1.1: Generic side-channel attack flow. The attacker measures the side-channel signal from the target device, performs some statistical analysis, and obtains the secret key.

signals to infer sensitive information. Just as a skilled listener can eventually open the safe by interpreting these subtle cues, a determined attacker can extract cryptographic secrets by analyzing the device's side-channel leakages.

Figure 1.1 exemplifies the high-level flow of a SCA on an embedded device, such as a smart card. To execute a SCA, a malicious actor must first gain physical access to the target smart card. Once access is established, the attacker measures the chosen side-channel signal while the device performs cryptographic operations. The attacker then shall make predictions about how the secret data affects the side-channel signal, often by exploiting known data like plaintexts. The final step involves statistically correlating these predictions with the measured signal to infer the processed data and, ultimately, the secret cryptographic key, thereby retrieving the card's PIN and breaching into the target. In practice, SCA requires a way to model how secret data influences the physical signals and a method to compare these predictions with the measurements, so that the attacker can efficiently extract secret information.

The rapid growth of the Internet of Things (IoT) over the past fifteen years has made SCA an even more pressing security concern. Connected embedded devices, ranging from smart home gadgets and fitness trackers to industrial sensors, have become integral to our daily lives, handling sensitive information. Protecting them in real-world situations is crucial to ensuring data safety and maintaining trust in modern technology. Consequently, millions of products undergo thorough security evaluations in laboratories worldwide every day [ALAM19]. Throughout these assessments, SCA is rigorously examined by both industry and academic researchers in the fields of information security and cryptography [BBB+21, BBYS22, PPM+23]. Thus, the process of side-channel analysis itself acts as a fundamental verification mechanism for cryptographic devices, highlighting the symbiotic relationship between attack and defense: only through effective attacks can evaluators identify device vulnerabilities and assess the effectiveness of implemented

countermeasures. Such an iterative cycle is essential for refining defense techniques, since each newly demonstrated attack may reveal previously unknown weaknesses and prompt the development of more robust defenses. Despite these continuous efforts, a truly secure system cannot be said to exist in information security. Instead, there are only systems that have not yet been compromised. The US National Institute of Standards and Technology (NIST) has mandated SCA resistance as a crucial security requirement for cryptographic modules in FIPS 140-3 standard [CS19]. Besides assessing SCA resistance, this standard outlines countermeasures such as noise generation, randomization, and secure hardware design to mitigate vulnerabilities.

While this thesis focuses primarily on physical side-channels on embedded devices, it is worth noting that the threat landscape extends beyond isolated systems. Modern computing platforms with multiple cores and advanced microarchitectural features, introduce additional complexity that amplifies side-channel vulnerabilities [OST06, YF14, ABuH+19]. Shared hardware resources and speculative execution mechanisms may leak sensitive information through timing and power variations [LSG+18, KHF+19], underscoring that side-channel threats are pervasive across diverse computing architectures.

1.1. Motivation: side-channel analysis workflow

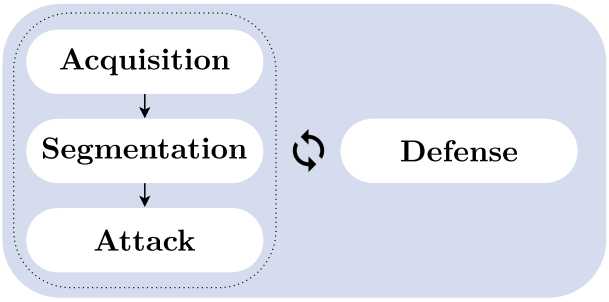


Figure 1.2: Side-channel analysis evaluation workflow. Attack and defense are in a symbiotic relationship.

Given the pervasive threat SCA poses to modern devices, a deeper understanding of the practical attack process is essential for developing effective countermeasures. SCA techniques leveraging power consumption have been successfully applied to a wide range of commercial devices, including payment and public transport smart cards [OP11], Intel microcontrollers [SRH16], and even complex systems such as iPhones [LKMM21]. The widespread deployment of these devices in critical systems worldwide underscores the practical relevance and impact of SCA in real-world scenarios. However, SCA has been primarily studied in controlled laboratory environments, where the evaluator can gain

complete access to the device under test and make idealizing assumptions about the target systems. Since SCA does not aim at the weaknesses of cryptographic algorithms but rather at the vulnerabilities of their implementations, executing an effective SCA in practice is a complex task that requires a deep understanding of the implemented target system, a high-cost infrastructure to measure and analyze side-channel signals, and the expertise to develop and apply sophisticated statistical techniques.

Starting from the seminal work in [KJJ99], a vast literature targeting SCA has emerged over the last two decades [CRR03, MPP16, BPS+20, WPP24]. Despite the diversity of SCA techniques, all approaches fundamentally depend on two critical prerequisites: *i)* multiple executions of the same cryptographic operation (CO) with varying inputs, and *ii)* accurate localization and alignment in time of COs within the side-channel traces. Figure 1.2 illustrates the SCA evaluation workflow necessary to meet these conditions. This workflow consists of several key steps: acquisition, segmentation, and attack. As a complementary and symbiotic step, the development of defensive countermeasures is also crucial.

Acquisition The first step in the workflow is the acquisition phase, where multiple COs are executed with varying inputs, i.e., plaintexts and keys. During this process, both the physical side-channel signal and the corresponding cryptosystem inputs are collected. In laboratory settings, dedicated evaluation boards such as SASEBO SAKURA-II [Inr15], Keysight Riscure icWaves [Key20], and NewAE ChipWhisperer Pro [New20a] are employed, carefully isolated from external interference and powered by clean, stable supplies. These boards are meticulously designed to minimize noise from the measurement setup and to maximize the desired signal leakage by carefully positioning ground and signal lines. A small shunt resistor is typically inserted in series with the power line of the device under test to facilitate precise measurement of power consumption.

Segmentation The second stage, segmentation, focuses on accurately localizing and aligning in time the COs within the raw,noisy traces just acquired. Effective segmentation guarantees that the relevant parts of the trace, i.e., the parts containing the information leakages related to the target CO, are properly isolated and synchronized. In a controlled laboratory environment, the acquisition and segmentation phases are typically carried out simultaneously. Security evaluation boards [Inr15, New18] used during acquisition are equipped with trigger pins, enabling precise synchronization between COs and the associated side-channel signals. This ad-hoc hardware support significantly streamlines the segmentation step for the evaluator. In research settings, the side-channel community frequently relies on publicly available datasets [BPS+20, Kiz20], which typically feature

low background noise, comprehensive metadata, and are organized with one CO per trace, meaning they are already segmented and perfectly synchronized for convenience.

Dedicated acquisition boards and open datasets offer significant advantages to researchers, enabling them to concentrate on the attack phase, which is often the most critical and technically demanding part of the workflow. However, these resources tend to overlook the segmentation phase, which becomes particularly challenging in real-world scenarios where hiding techniques have been widely adopted as a cost-effective solution to hinder the pre-processing of side-channel traces. Attackers must deal with raw traces from real devices, without relying on trigger pins or other tools. As a result, attackers are required to manually identify and isolate COs within the side-channel traces, a process complicated by noise, signal variability, and the presence of advanced obfuscation countermeasures.

Attack The final and most challenging stage is the attack phase, where the attacker exploits the identified COs to extract sensitive information. This phase often requires sophisticated techniques and a deep understanding of the target system's vulnerabilities. In the last decade, the adoption of machine learning techniques [MPP16, BPS+20, RWPP21] has revolutionized the attack phase, enabling attackers to automatically learn complex leakage models directly from raw traces, bypassing the need for manual feature engineering and expert knowledge. Devices protected by countermeasures previously considered robust, such as masking and hiding, are now vulnerable to these advanced techniques, driving ongoing research into new and improved countermeasures.

Defense Complementing the attack phase, the development of effective defense countermeasures is crucial to protect cryptographic devices from emerging threats. Countermeasures generally fall into two main categories: masking and hiding. Masking techniques [GP99, KMMS21] divide sensitive data into multiple independent parts, called shares, which are processed separately. This approach reduces the link between each share and the secret key, making it harder for attackers to extract useful information. Hiding methods [YWV+05, CK09] focus on introducing randomness into the side-channel signals, obfuscating the leakage and making it more difficult for attackers to find patterns and exploit leaked information. Hiding actuators are widely available, even on low-end IoT devices, making them a cost-effective solution to harden side-channel security.

Despite the growing security threat of SCA to IoT devices, the hardware platforms on which they are built often lack robust defenses. Most commercial microcontroller-based system-on-chips (SoCs) run cryptographic primitives even though security was not a primary design goal, primarily due to tight deadlines, constrained budgets for power,

performance, and area (PPA), and cost-saving considerations [BBIP21]. Side-channel vulnerabilities challenge the critical need for a careful balance between performance and security. Implementing robust countermeasures often incurs a non-negligible cost, as these techniques can reduce system performance, increase power consumption, and require additional silicon area for secure cryptographic accelerators. Achieving absolute security is unrealistic, instead, practical solutions aim to maximize protection while minimizing impact on performance and system complexity.

1.2. Research questions

This thesis aims to develop an innovative methodological framework to evaluate the security of digital circuits in realistic scenarios, reducing the costs and complexity of the analysis. Deep-learning techniques are an essential tool for achieving this goal, as they can automatically learn complex patterns in raw side-channel traces, allowing us to get closer to a realistic attack, removing the need for some common laboratory assumptions.

To this end, the thesis addresses three research questions (RQs), each targeting a specific aspect of the side-channel analysis workflow, from the practical difficulties of trace segmentation to the effectiveness of current countermeasures and the need for efficient security solutions.

RQ1 How can cryptographic operations be accurately located within side-channel traces containing millions of time samples in realistic scenarios, e.g. dealing with obfuscated traces and without relying on common laboratory assumptions?

RQ2 How can information leakage be restored from side-channel traces desynchronized by real physical noise via hiding countermeasures to enable successful attacks?

RQ3 What strategies enable efficient side-channel security for digital circuits without incurring significant hardware redesign costs?

To assess the three research questions, several quality metrics are considered throughout this thesis. First, the evaluation focuses on the ability to accurately locate all cryptographic operations within obfuscated traces and to segment them with precise temporal accuracy. Notably, only a sufficiently precise segmentation enables the temporal alignment of individual COs, which is necessary to perform effective side-channel attacks. Second, analysing how hiding countermeasures resist deep-learning attacks requires quantifying the strength of the attack. In particular, the success rate, the resources required, such as the number of traces needed, and the computational effort involved, including the time and number of neural networks that must be explored before satisfactory results are

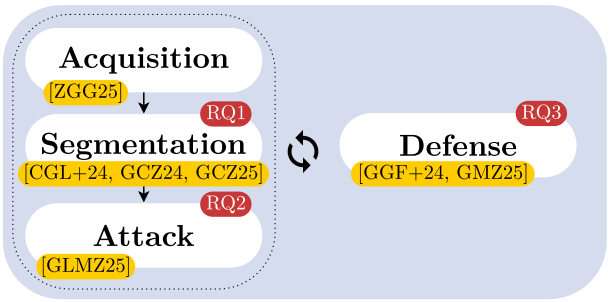


Figure 1.3: Contributions and research questions for each side-channel attack workflow stage.

achieved, are considered. Finally, the efficiency and cost of the defense mechanisms are analyzed by considering PPA metrics, the engineering complexity required to implement such countermeasures, and their overall effectiveness, i.e., whether the system remains vulnerable to attacks and what level of effort is needed to compromise its security.

1.3. Contributions

Over the past decades, the side-channel research has made substantial progress in understanding and mitigating the risks posed by SCA. While much of the state-of-the-art research has focused on the critical attack and defense phases, the sensitive acquisition and segmentation steps have received less attention. Acquisition and segmentation stages are crucial for obtaining high-quality side-channel traces but they require ad-hoc setups, stable measurement conditions, and expertise, which are not always available. To streamline the analysis process and focus on the most critical and challenging phases, the community often falls back on open datasets [rg14, BPS+20, CM24] that provide clean traces and comprehensive metadata. Nevertheless, these datasets may not always capture the specific scenarios of interest, and their limitations can hinder evaluating novel analysis techniques. When suitable public datasets are unavailable, researchers must undertake the entire end-to-end workflow themselves.

This thesis advances each step of the SCA workflow by emphasizing realistic scenarios and providing practical tools and methodologies. These contributions enable the evaluation of countermeasures and the benchmarking of analysis techniques under conditions that closely reflect real-world challenges. Figure 1.3 summarizes the co-authored research papers contributing to each SCA stage and their connection to the research questions addressed in this thesis.

Acquisition To ensure a rigorous and reproducible evaluation across the entire workflow, a unified experimental infrastructure for collecting side-channel signals on IoT-class devices is presented. Specifically, the experimental acquisition phase relies on JARVIS [ZGG25], an open-source hardware and software platform designed for FPGA-based research. Serving as the foundational instrumentation layer, JARVIS provides a comprehensive environment, including a RISC-V processor, dedicated modules for implementing countermeasures, and a custom debug system that enhances the visibility and control of the experimental setup. Its minimal FPGA-based architecture simplifies the development, validation, and refinement of SCA methodologies and countermeasure, as well as reducing unnecessary leakage sources and thus making analysis more straightforward. The platform also ensures that the prototype and its emulated version behave consistently, thanks to dedicated hardware features for accurate signal collection. Consequently, JARVIS allows for extending traditional PPA evaluation metrics with a concrete security perspective, offering a ready-to-use ecosystem for robustly evaluating the segmentations, attacks and defenses proposed in this work. The platform is publicly available[1], making the experimental setup accessible to the community.

Segmentation To further advance segmentation research, this thesis presents Hound [CGL+24, GCZ24, GCZ25], a deep-learning tool for locating cryptographic primitives in traces heavily affected by hiding countermeasures. Hound eliminates the need for trigger infrastructure, easing the COs segmentation in real-world environments, and it is successfully validated on multiple scenarios, testing various cryptographic algorithms and hiding techniques.

The thesis also introduces the Chameleon dataset [GCZ25], which is the first to capture power traces from a real device implementing four different hiding countermeasures, i.e., dynamic frequency scaling [GMZ25], random delay [CK09], morphing [ABPS15b], and chaffing [ABPS15a]. Each trace contains several executions of the CO of choice interleaved with multiple general-purpose applications, where the interleaving strategy has been selected to mimic the concurrent execution of the applications on the computing platform. In contrast to any previous dataset, the noise due to the actuators implementing the hiding countermeasure is measured rather than artificially modeled, making it a more realistic and valuable resource for assessing the effectiveness of side-channel countermeasures. The acquisition process was extensive, requiring over 60 hours and producing a dataset of 2 560 annotated traces. Chameleon is the first dataset designed to support both segmentation and attack stages, thanks to detailed metadata marking the start and

[1] JARVIS: https://github.com/hardware-fab/JARVIS

end of each CO execution.

Both the tool and its dataset are open-source[2] [3], supporting reproducibility and further research.

Attack Another key contribution is the development of new datasets and methodologies for analyzing highly desynchronized traces by dynamic frequency scaling (DFS) countermeasures. The thesis introduces the `DFS_DESYNCH` dataset [GLMZ25], which features AES power traces collected from a real system employing DFS to induce high temporal misalignment. Unlike previous open datasets, which exhibit only minor desynchronizations of 100 samples between two traces, `DFS_DESYNCH` provides traces with significantly larger timing variations, on average of 64 536 samples. A digital clock labeling system is also proposed, allowing for precise monitoring of frequency changes without the need for expensive equipment.

A novel deep-learning pre-processing named Owl [GLMZ25] is presented to handle these challenging desynchronized traces. The approach uses deep learning to resynchronize and realign the traces to a common frequency, validated by a subsequent SCA to recover the secret key. Experiments on the DFS_DESYNCH dataset demonstrate a successful recovery of the secret key, confirming the effectiveness of the approach.

The tool and dataset are publicly available[4] to foster further research.

Defense A thorough experimental campaign was conducted on several FPGA devices to study the impact of random dynamic voltage and frequency scaling (DVFS) and inter-chip variability on trace alignment and side-channel leakage [GGF+24]. Main results illustrate that analog components in the DVFS actuator can hinder vulnerabilities, but using a high-pass filter restores most of the leakage. Variability between FPGA chips is insufficient to compromise SCA, while frequency scaling proves to be the most effective method for hiding. The effectiveness of desynchronization improves with the distance between operating frequencies rather than depending on the number of frequencies used.

The thesis proposes Rabbit [GMZ25], a hardware module for AMD Xilinx FPGAs that performs random dynamic and fine-grained frequency scaling. Rabbit leverages a dual-MMCM design to perform rapid temporal dynamics while keeping the system running smoothly during reconfiguration, achieving high performance with minimal overhead. Fi-

[2] Hound: https://github.com/hardware-fab/DL-to-locate-COs-for-SCA, https://github.com/hardware-fab/Hound

[3] Chameleon: https://github.com/hardware-fab/chameleon

[4] Owl and DFS_DESYNCH: https://github.com/hardware-fab/DLaTA

nally, the security of Rabbit was tested against various attacks, including correlation power analysis, template attacks, and deep-learning methods. The Test Vector Leakage Assessment (TVLA) was also applied with no leakage detected in up to ten million traces, demonstrating Rabbit's strong resistance to SCA.

1.4. Thesis structure

This thesis is organized into seven chapters, each corresponding to a key phase of the side-channel analysis workflow and addressing the research questions associated with it.

- **Chapter 2: Background and State of the Art**
  Reviews the fundamentals of side-channel attacks, including acquisition, segmentation, attack methodologies, and countermeasures. It introduces relevant frameworks, open datasets, and hardware platforms, and discusses classical and deep-learning-based attack techniques as well as defense strategies.
- **Chapter 3: Cryptographic Operations Segmentation in Real-World Scenarios**
  Details the methodology for segmenting COs in obfuscated side-channel traces. It presents the Hound deep-learning pipeline for segmentation, describes the training and inference processes, and discusses heuristics for segmentation errors. The chapter includes materials from [CGL+24, GCZ24, GCZ25].
- **Chapter 4: Attack and Defense Strategies Using Deep Learning**
  Explores advanced attack techniques, including the deep learning-assisted SCA Owl and a dynamic clock randomization countermeasure. It describes the Rabbit hardware architecture for frequency randomization and evaluates its effectiveness against SCAs. The chapter includes materials from [GGF+24, GLMZ25, GMZ25].
- **Chapter 5: Setup and Benchmark for Methodology Evaluation**
  Introduces the JARVIS hardware-software framework for side-channel research and presents two novel datasets, Chameleon and DFS_DESYNCH, designed to benchmark segmentation and attack techniques under realistic hiding conditions. The chapter includes passages from [CGL+24, GLMZ25, ZGG25, GCZ25].
- **Chapter 6: Experimental Results and Comparative Evaluation**
  Presents a comprehensive experimental campaign to evaluate the proposed methodologies and their impact on side-channel analysis. It evaluates segmentation and attack metrics, analyzes the effectiveness of countermeasures, and reports results for Hound, DLaTA, Rabbit, and other defense mechanisms. The chapter includes

results from [GGF+24, CGL+24, GCZ24, GLMZ25, GMZ25, GCZ25].

- **Chapter 7: Conclusions and Future Directions**
  Summarizes the main findings, discusses the implications for side-channel security, and outlines directions for future research.

2 | Background and State of the Art

Side-channel analysis was first introduced in the late 1990s as a method to exploit physical leakages from cryptographic devices [Koc96, KJJ99, QS01]. Over the years, SCA has evolved into a comprehensive field encompassing various techniques for acquiring, processing, and analyzing side-channel information. Table 2.1 summarizes representative contributions across the SCA workflow. Entries are grouped by acquisition, segmentation, pre-processing and attack technique, and defense. Most studies focus on attack and countermeasures, while acquisition and segmentation persistently receive less attention, with only a few works addressing these sensitive phases. This thesis aims to contribute to each of these phases.

This chapter overviews the basic concepts and techniques pertinent to this thesis. It first introduces trace acquisition, common frameworks, and open datasets, with their limitations (see Section 2.1). It then presents methods for segmenting side-channel traces (see Section 2.2). Next, it reviews side-channel analysis and attack methodologies, highlighting the growing role of deep learning (see Section 2.3). Finally, it reviews countermeasures, emphasizing dynamic frequency scaling implemented via AMD Xilinx clock managers (see Section 2.4).

2.1. Acquisition

Side-channel power analysis begins with the acquisition of side-channel traces. A side-channel trace is a time series of physical measurements, such as power consumption, collected during the execution of a CO on a target device. Therefore, the acquisition phase involves triggering the execution of multiple COs with different inputs, i.e. plaintexts and keys, while recording the physical side-channel signals and the corresponding inputs. Side-channel evaluators in laboratory environments rely on specialized boards designed to reduce noise and ensure stable measurements, such as SASEBO SAKURA-II [Inr15], Keysight Riscure icWaves [Key20], and NewAE ChipWhisperer Pro [New20a]. Few works

Table 2.1: Review of State-of-the-Art acquisition, segmentation, attack, and defense methodologies for side-channel analysis.

Acquisition	**Segmentation**	**Attack**		**Defense**
		Pre-processing	**Technique**	
[BUS21]	[BBG+01]	[CCD00]	[KJJ99]	[GP99]
[HGC+24]	[BFP22]	[Man04]	[QS01]	[CCD00]
[Low24]	[TBW+21]	[APSQ06]	[CRR03]	[YWV+05]
	[QWY+24]	[DRS+13]	[BCO04]	[LOM08]
		[HGG19]	[GBTP08]	[CK09]
		[WP20]	[HMH+12]	[GM11]
		[WPP22]	[HZ12]	[ABPS15a]
		[KPP24]	[BCD+13]	[ABPS15b]
			[VCGS14]	[GMK16]
			[HPGM17]	[CRZ18]
			[MPP16]	[DCEM18]
			[CDP17]	[JIP19a]
			[HGG19]	[BFPZ20]
			[PHJ+18]	[HDL+20]
			[BPS+20]	[ABP21]
			[ZBHV20]	[DHL+21]
			[LZC+21]	[KMMS21]
			[RWPP21]	[SBM21]
			[WPP22]	[CDGT23]
			[GLBG23]	[CPW24]
			[SM23]	[FRBSG24]
			[AHC+24]	[GD24]
			[BIK+24]	[WFW+24]
			[GR24]	[BLH+25]
			[SKP+24]	[LMDM25]
			[WPP24]	[TCG+25]

focus on the acquisition phase [BUS21, HGC+24], studying acquisition parameters such as sampling rate, chip voltage, and clock frequency to obtain the best possible side-channel signal quality.

However, to streamline the analysis process and let researchers focus on the attack rather than the acquisition, the side-channel community often does not perform the acquisition themselves but employ open datasets [BPS+20, Kiz20] that provide quality traces with detailed metadata.

2.1.1. Side-channel analysis frameworks

Several commercial and open-source frameworks are available for side-channel analysis. Company such as Keysight [Key20] and NewAE [New20a, New18, New20b] offer mature platforms for acquisition, triggering, and analysis, with most tools and infrastructures designed for conventional microcontrollers, like ARM and AVR, or for AES accelerators. Much of the existing research focuses on older hardware platforms, such as STM32 32-bit Arm Cortex-M [ABPS15b] and ATmega 8-bit AVR [BPS+20] microcontrollers, underscoring the importance of advancing SCA studies on newer IoT-class devices, such as RISC-V platforms. RISC-V currently lacks adequate SCA frameworks, with only a few ASIC-based or open solutions available.

ASIC-based options, for example by mounting an FE310-G002 daughter board on NewAE's CW308 [New20b], have two main limits. First, locating side-channel leakage inside the microarchitecture is difficult because the platform offers low observability and is not fully open. Second, researchers cannot easily change the hardware to implement or test countermeasures. On the other hand, NewAE's CW305, CW310, and CW340 boards also allow the users to flash their own custom bitstreams, including RISC-V-based designs. This requires however either porting a third-party computing platform to the new target or designing a new one, and both paths notably require a large amount of effort. lowRISC provides an open-source SCA setup for OpenTitan [Low24] that can instantiate the OpenTitan root of trust on a NewAE board to test its cryptographic blocks against power SCA. However, OpenTitan necessitates a significant amount of FPGA resources and typically requires large, costly FPGAs (e.g., AMD Xilinx Kintex-7), increasing expenses. It also does not include actuators for SCA protections. Due to its size and complexity, modifying OpenTitan to add or test new countermeasures is challenging, and the setup complicates the task of locating the exact source of leakage within the microarchitecture.

On the software side, there exist several open-source projects and libraries [BIKP19, PWP21, MMH+25] that support the community in the processing and analysis phase, while the acquisition phase is often abstracted away.

2.1.2. Side-channel analysis open datasets

To ease the development of novel side-channel attacks bypassing the acquisition phase, there is a vast availability of open side-channel datasets, with each offering different levels of complexity and protection. Notably, all these datasets are based on different versions and implementations of the AES algorithm [oSND+23]. Many of these datasets are widely used for benchmarking attack techniques, but they still present notable limitations, es-

pecially when considering modern and practical scenarios [PPM+23]. The datasets from *DPAv4* [rg14] and *CHES CTF* [CBT20] were introduced as part of security competitions. DPAv4 includes 100 000 measurements from an Atmel ATMega-163 smartcard executing a masked AES-256 and AES-128. CHES CTF provides 50 000 traces recording the computation of a masked AES-128 implementation on an STM32 Cortex-M4 microcontroller.

ASCADv1 [BPS+20] contains electromagnetic traces from an ATMega-8515 microcontroller running a boolean masked version of AES-128. ASCADv1 is available in two sets, the first set of 60 000 measurements with a fixed key and the second set of 300 000 traces with random keys. ASCADv2 [dlSdSd21] expands the collection with 810 000 traces from software AES-128 executions protected by shuffling and affine masking, measured on an STM32 Cortex-M4 microcontroller. All traces in both ASCAD collections are measured at a constant frequency, and the authors supply scripts to artificially add desynchronization. This means the dataset only simulates desynchronization in post-processing, missing the real effects of a physical frequency switching with an actual actuator. As a result, algorithms tested on ASCAD [WP20, RWPP21] may behave differently in practical applications.

An additional 50 000 electromagnetic traces are available in the AES_HD dataset [BJP20]. The measurements are collected from a SASEBO-GII board featuring an AMD Xilinx Virtex-5 FPGA where an AES-128 accelerator is deployed.

SMAesH [CM24] allows researchers to compare the power leakages of a masked AES accelerator deployed on two different FPGAs, i.e., AMD Xilinx Artix-7 and Spartan-6. The dataset contains 6×2^{24} traces, recording only the first AES round.

The eShard dataset [VTM21] features electromagnetic measurements from a AES-128 computation on an STM32F446 microcontroller, leveraging boolean masking and shuffling for protection.

None of these datasets include traces from devices using hiding countermeasures to introduce temporal desynchronization. The only exception is AES_RD [Kiz20], which offers 50 000 power traces recorded from an Atmel AVR microcontroller with a random delay countermeasure [CK09].

As discussed in [PPM+23], there are three main limitations in current datasets. First, support for hiding countermeasures is minimal, with only AES_RD implementing one such technique. Second, all datasets focus solely on the attack phase, with each trace containing a single AES execution, removing the need to locate cryptographic operations within raw measurements and preventing analysis of their exact position. Third, traces

only include cryptographic operations, creating unrealistic scenarios where the application runs in isolation, unlike real systems where cryptographic tasks run alongside other workloads.

2.2. Segmentation

During side-channel evaluation in laboratory environments, the segmentation of COs in side-channel traces is typically facilitated by trigger signals, either hardware-based, as offered by security evaluation boards such as SASEBO SAKURA-II [Inr15] and NewAE CW305 [New18], or software-based via virtual triggering features provided by commercial platforms like Keysight Riscure icWaves [Key20] and NewAE ChipWhisperer Pro [New20a]. These platforms detect predefined patterns in real time and emit synchronization signals to aid CO localization. Therefore, traces acquired using these platforms are already segmented, with each trace containing a single CO execution.

Without access to a proper trigger infrastructure, the segmentation of COs in side-channel traces becomes a challenging task. [QWY+24] introduces a novel framework to systematically enable precise CO segmentation without relying on source code knowledge or trigger signals. During the profiling phase, the authors instrument key firmware instructions to toggle a GPIO pin when a CO occurs. This creates annotated SCA traces used to build CO templates later used during inference, matching the CO templates against raw traces to locate similar CO instances automatically. A similar approach was previously adopted in [LKMM21] to breach into an iPhone's chip. The authors managed to tamper with the bootloader to query the AES hardware engine and to reconfigure a GPIO pin for triggering the SCA measurements.

Alternative methods leverage pre-computed templates for pattern matching [BBG+01] or utilize matched filters to detect specific cryptographic routines like AES-128 [BFP22], offering robustness in the presence of interrupts and busy waits. However, these techniques typically require consistent, hand-crafted templates and fail to handle interrupted or deformed instances of the target operation, often discarding them entirely. To overcome this rigidity, Trautmann et al. [TBW+21] proposed a semi-automatic template-building technique that derives a CO template online based on known operation characteristics, such as the number of encryption rounds.

Still, architectural-level countermeasures like time-sharing multithreading [FI16], interrupt-driven routines, random delays, and dynamic frequency scaling (DFS), introduce timing variability and distort trace morphology, leaving these traditional pattern-matching approaches ineffective.

2.3. Side-channel analysis and attacks methodologies

Over the last two decades, SCAs demonstrated a large variety of ways of exfiltrating sensitive information, e.g., the secret key of a cryptographic primitive, from embedded computing platforms. Attack methodologies are typically categorized into *i)* non-profiling and *ii)* profiling attacks. Non-profiling attacks, such as Differential Power Analysis [KJJ99] and Correlation Power Analysis [BCO04], target the device directly by correlating partially known input data with the observed physical signals generated during cryptographic computations. Profiling attacks [CRR03, HZ12, MPP16] require the attacker to have access to a clone of the target device. This allows the attacker to build a model by collecting power traces while running the clone device with known keys and plaintexts. Ideally, the attacker can gather as many measurements as needed to create an accurate profile. Then, the attacker queries the actual target device, whose key is unknown, with known plaintext and collects the corresponding side-channel traces. The built model is then used to analyze these traces and recover the secret key. Profiling attacks are generally more effective but require greater effort and resources to execute.

Since the introduction of profiling attacks with the Template Attack [CRR03], the domain has experienced a progression in attack complexity. The adoption of machine learning [HZ12, HPGM17] and, more recently, deep learning approaches [PPM+23] has revealed the potential for compromising cryptographic devices, thereby fueling the continuous investigation of advanced countermeasures.

2.3.1. Pre-processing

As a first step in side-channel analysis, traces are typically pre-processed to reduce computational complexity and enhance information extraction, such as by removing noise. Point-of-Interest (PoI) selection methods are used to identify the most relevant time samples in a trace, thereby reducing data dimensionality. Common approaches include Signal-to-Noise Ratio (SNR) analysis [Man04], Linear Discriminant Analysis (LDA) [Fis38], and Principal Component Analysis (PCA) [WEG87, APSQ06]. Machine learning techniques have also been introduced to automatically combine multiple time samples, such as triplet embedding for efficient data representation [WPP22] or Layer-wise Relevance Propagation (LRP) applied to trained CNNs to pinpoint the most informative time samples [HGG19].

Desynchronization and random noise are persistent challenges in side-channel analysis, hardening the task of effective trace alignment and analysis. Two main strategies are commonly adopted to address desynchronization effects caused by countermeasures like

clock jitter [CDP17], Random Delay Insertion [CK09], and shifting [BPS+20]. The first consists of increasing the number of traces used for profiling by a factor proportional to the desynchronization magnitude. Conversely, the second is to apply various realignment techniques to resynchronize the multiple traces. A common technique consist in averaging continuous samples in a sliding window fashion to reduce the impact of small shifts [CCD00], effectively resampling the traces. Durvaux et al. [DRS+13] proposed the use of Hidden Markov Models pattern recognition to remove any dummy operation used as random delay. Other techniques are signal-processing oriented leveraging waveform matching and elastic alignment, proven to be more adapted to hardware countermeasures [NHI+07, vWWB11]. An approach based on autoencoders has been presented by Wu and Picek in [WP20]. Their proposed neural networks can clean up traces and realigning them by treating desynchronization as noise. However, the training of these autoencoders presupposes the availability of a device that can selectively deactivate countermeasures. Karayalcin et al. [KPP24] present a diffusion model-based approach to remove measurement noise from side-channel traces, demonstrating that denoising traces with this method significantly reduces the complexity of mounting attacks.

2.3.2. Classical side-channel analysis methodologies

Classical side-channel refers to statistical analyses and attack techniques that do not leverage machine or deep learning. These techniques were historically the first to be developed, and although they are less powerful than deep learning techniques, they are still crucial to validate leakage in measurements.

Correlation power analysis

Correlation Power Analysis (CPA) [BCO04] is a non-profiled attack technique which works by building a model of the device's power consumption and comparing it to the actual measured traces.

In practice, the attacker models power consumption by calculating the Hamming Weight of intermediate values produced during cryptographic operations. For AES, this often means computing the Hamming Weight of the output from the S-Box. The attacker then generates a power model for each possible key byte value and correlates these models with the collected power traces to identify the correct key. The strength of this correlation is measured using the Pearson Correlation Coefficient (PCC), shown in Equation 2.1.

$$PCC(M,P) = \frac{Cov(M,P)}{\sqrt{Var(M)}\sqrt{Var(P)}} \tag{2.1}$$

In this equation, M is the measured power consumption and P is the predicted power consumption from the model.

To recover a key byte, the attacker calculates the Pearson Correlation Coefficient for each possible guess, from 0 to 255. The most likely candidate key byte is the one producing the highest correlation.

Template attacks

Template Attack [CRR03] is a profiling attack which relies on Bayes' theorem for key prediction.

During the profiling stage, the attacker is assumed to have complete access to a clone of the target device, enabling the collection of a large set of side-channel measurements. These traces are used to construct statistical models, known as templates, which characterize the leakage behavior of the device. Each template is built relying on a multivariate Gaussian distribution, defined by a mean vector and a covariance matrix, capturing the noise and signal patterns associated with specific plaintext-key pairs $(p, k) \in \mathcal{P} \times \mathcal{K}$. During the attack phase, traces from the target device are classified against the precomputed templates using a maximum likelihood estimation approach. The template that best matches the observed traces reveals the most probable key candidate, effectively identifying the secret key.

Generally, a single trace is not sufficient for accurate key recovery, necessitating the use of multiple traces to improve the attack's success rate. However, if built correctly, template attacks require a significant amount of less data than non-profiling attacks during the attack phase.

Test Vector Leakage Assessment

Test Vector Leakage Assessment (TVLA) [BCD+13] is a statistical method used to check for the presence of side-channel leakage, without retrieving the secret key. TVLA collects side-channel traces during COs execution on specific test vectors and applies statistical tests to determine if sensitive data affects the traces. Among the available tests, the non-specific or fixed versus random test is the most widely used. This test relies on Welch's t-test, shown in Equation 2.2, where $\bar{X}_1$ and $\bar{X}_2$ are the average values of two groups of power traces, s_1^2 and s_2^2 are their variances, and n_1 and n_2 are the number of traces in

each group.

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{2.2}$$

The TVLA procedure involves collecting two sets of power traces during encryption with a fixed key. In the first set, N traces are recorded using different random plaintexts. In the second set, N traces are taken with the same plaintext. The traces are mixed randomly within their own group to reduce bias from environmental changes. Welch's t-test is applied to the first $N/2$ traces from both sets to get a t-score, and then repeated on the remaining $N/2$ traces for another t-score. Running the test twice helps avoid false positives. If both t-scores are outside the ± 4.5 range, the device is considered to have failed the test and information leakage is present. If the t-scores stay within ± 4.5, the device is considered resistant up to N traces. The ± 4.5 threshold has been set to reject the null hypothesis with $> 99.999\%$ confidence.

Notably, TVLA is a qualitative test that only indicates the presence of leakage, not its severity or exploitability. Detected leakage may be very difficult to exploit in practice.

2.3.3. Deep-learning-based attacks

Deep-learning-based SCAs are typically profiled attacks that exploit a neural network to classify a side-channel trace based on a key-dependent operation, like the AES S-Box.

In supervised machine learning, the objective is to learn a function $g\colon \mathcal{X} \to \mathcal{Y}$ that maps inputs to outputs using example pairs. This function is parameterized by θ^z, where z is the number of trainable parameters and θ is the vector of weights learned during profiling. In side-channel analysis, the input is a one-dimensional time series, with each sample representing a feature. During profiling, the model learns θ by minimizing a loss function, often categorical cross-entropy, using a training dataset to ensure reliable performance [PPM+23].

Deep-learning SCAs typically use convolutional neural networks [PHJ+18, BPS+20, ZBHV20] or multilayer perceptrons [PHJ+18, BPS+20], which are effective at finding patterns in side-channel data [PPM+23]. Although transformers [BIK+24], long short-term memory networks [MPP16], and generative adversarial networks [MS23] have been explored in side-channel analysis, their adoption remains limited in the literature. The increased architectural complexity of these models is often not justified by corresponding improvements in attack effectiveness. The universal approximation theorem [HSW89] implies that

multilayer perceptrons can approximate any continuous function, theoretically enabling them to model the complex leakage functions associated with any protection scheme, provided sufficient model capacity and training data. To improve model efficiency, different Neural Architecture Search (NAS) algorithms have been presented with ad-hoc search strategies for side-channel [RWPP21, WPP24]. By rewarding the NAS agent with information on attack success and efficiency, it is possible to automatically explore networks architectures and create models suited for different cryptographic datasets.

For deep-learning-based SCAs, it is best practice to standardize and normalize data before training neural networks. Data augmentation can also help improve generalization when the number of profiling traces is limited. However, training on raw, unprocessed traces is recommended, as it preserves the original information in the power measurements [PPM+23].

Convolutional neural networks

Convolutional Neural Networks (CNNs) are a popular deep learning architecture widely applied in side-channel analysis [PPM+23]. CNNs are well-suited for processing time series data, such as power traces, because they can automatically learn hierarchical features from the input. A typical CNN is built from three main types of layers: convolutional layers, pooling layers, and fully connected layers. Convolutional layers use small filters (kernels) that move across the input data. At each position, the filter computes a dot product between its weights and the input values, producing feature maps. Pooling layers help reduce the size of the data by summarizing groups of neighboring outputs, which makes the model more efficient and less sensitive to small changes. Fully connected layers link every neuron in one layer to every neuron in the next, and apply an activation function to introduce non-linearity. CNNs often include batch normalization layers as well to stabilize and speed up training by reducing internal covariate shift.

Gradient-weighted class activation mapping Gradient-weighted Class Activation Mapping (Grad-CAM) [SCD+17] is a technique to understand how CNNs make decisions. Grad-CAM creates a heatmap highlighting the input regions most important to CNN's classification. This heatmap can then facilitate input segmentation, which, in the context of this work, involves assigning a frequency label to each sample based on its temporal window. A significant benefit of employing Grad-CAM is its ability to operate without the need for additional training. It leverages a pre-trained classification network, thereby streamlining the segmentation process. Typically, segmentation tasks demand more complex networks and extensive training. However, Grad-CAM circumvents this by reducing

the training requirement to a classification problem. Moreover, Grad-CAM enables the creation of a segmentation model even when feature- wise labels for training are absent. This is particularly advantageous since such labels are prerequisites for specialized segmentation task architectures.

Grad-CAM has been demonstrated to provide high-resolution and class-discriminative visualizations when applied to image classification tasks [SCD+17]. Inspired by the object localization performance obtained by Grad-CAM in the image domain, we built upon the same ideas to localize the regions referring to the different frequencies in the power traces, which, for the considered task, we demonstrate to be a valid approximation of the desired segmentation.

2.4. Side-channel countermeasures

Since the emergence of side-channel attacks, a variety of defensive techniques were proposed to mitigate information leakage and raise the attacker's cost [KJJ99]. These countermeasures differ in the threat model they address, their implementation cost, and the security guarantees they can provide.

Countermeasures are commonly organized into two broad categories: masking and hiding. Masking countermeasures [GP99, GMK16, KMMS21] split the sensitive intermediate values into different shares that are computed independently to minimize the dependency of each share from the secret key. Hiding countermeasures [CCD00, YWV+05] aim to randomize the target side-channel signal with the goal of reducing the exploitable side-channel information leakage. Notably, the randomization induced by hiding techniques can be defined as adding a correlated noise to the side-channel signal. Indeed, the effectiveness of any hiding technique strongly depends on the difficulty for the attacker to separate the noise from the side-channel information leakage (de-noising). The success of hiding techniques is due to the complexity and the cost of designing secure masking solutions compared to the high availability of actuators to implement hiding techniques even on low-end and low-cost IoT devices.

2.4.1. Desynchronization and dynamic frequency scaling countermeasures

A common requirement for the numerous SCA methodologies lies in the measurements to be aligned and synchronous with each other to work correctly. Therefore, a natural hiding countermeasure involves the desynchronization of the trace time samples. Con-

sidering deep sub-micron technology nodes, clock jitter represents a process variability effect that has been observed to contribute severely to trace desynchronization [CDP17]. Notably, clock jitter provokes small and random variations in the period of the clock signal. Conversely, random delay countermeasure leverages Random Process Interrupts or software dummy cycles to insert a random number of delay cycles during the execution of a CO [CCD00, LOM08, CK09].

One of the most common desynchronization techniques consists of randomly changing the operation frequency of the computing platform at run-time using a Dynamic Frequency Scaling (DFS) actuator [YWV+05]. The DFS proposed by Guneysu and Moradi [GM11] generates eight different clocks, each phase-shifted by a fixed amount of 45°, with the same frequency using two Digital Clock Managers in AMD Xilinx FPGA. Then, a cascaded network of clock multiplexers randomly combines the phase-shifted clocks into a single clock output that drives the cryptographic core. This method increases the Differential Power Analysis (DPA) resistance but lowers the clock signal's pulse rate and reduces the throughput. The first Mixed-Mode Clock Manager (MMCM)-based SCA countermeasure was introduced by Fritzke [Fri12]. A single MMCM was used to generate four different clocks with frequencies that were multiples of the input frequency (3×, 4×, 5×, 6×). The cryptographic core randomly chose one of these clocks to run. The design without protection needed 500 000 encryptions to expose the secret key, while the proposed design resisted DPA up to three million encryptions. Jayasinghe et al. [JIP19a] have proposed a more advanced design based on MMCMs called Runtime Frequency Tuning Countermeasure (RFTC). RFTC exploits the dynamic reconfigurability of three MMCMs to achieve 3 072 distinct frequencies, ranging from 24 to 48 MHz. Frequency configurations are stored in 20 on-chip BRAM units. The clock is dynamically switched among any of the three MMCM outputs through a cascaded network of BUFGs while the encryption is running. Nevertheless, the computation must be frequently stopped to reconfigure the MMCMs. The authors have tested their design against various CPA attacks and found no leakage in four million traces. They also apply a similar method to an open-source RISC-V Processor in another work [JIP19b]. Hettwer et al. [HDL+20] exploit the reconfigurability of AMD Xilinx MMCMs to change the frequency configuration after a certain number of AES encryption. Their design uses a single MMCM with up to eight clock outputs, and a clock multiplexer randomly switches the output clock while the cryptographic core is running. The authors demonstrate that changing the MMCM configuration every 10^3 encryption is enough to resist powerful CNN attacks. Dao et al. [DHL+21] implement a Random Dynamic Frequency Scaling countermeasure for FPGA that changes clock frequency every AES computation. The authors show that with 219 000 different frequencies, this method

reduces leakage and makes CPA fail with five million traces, but it is not enough to stop a deep learning attack that recovers 3 bytes out of 16.

Clock managers on AMD Xilinx FPGAs

In AMD Xilinx designs, a common approach to dynamically changing the operating clock is to use the Mixed-Mode Clock Manager (MMCM) [AT]. Therefore, understanding the MMCM's capabilities and how it is reconfigured at run-time is essential to evaluate both the security effectiveness and the system-level trade-offs of DFS-based hiding. The MMCM is a specialized AMD Xilinx primitive that generates up to seven output clocks from a single input clock. It combines digital and analog elements and can be reconfigured during operation by updating its control registers via the Dynamic Reconfiguration Port [AT]. An MMCM can adjust output frequency, phase, and duty cycle, producing different simultaneous clock outputs with independent divider settings. This flexibility makes the MMCM well suited for DFS countermeasures introducing temporal variability and hampering alignment-based attacks.

The relationship between the input frequency f_{in} and the j-th output frequency f_{out_j} of an MMCM is described by Equation 2.3, where M is the clock feedback multiplier, D and O_j are clock dividers. While M and D are shared for all seven outputs, O_j can be set differently for each output frequency.

$$f_{out_j} = f_{in} \frac{M}{D \times O_j} \tag{2.3}$$

MMCM's scaling parameters (M, D, and O_j) are overwritten during reconfiguration, delivering a new frequency.

The MMCM manipulates the input clock signal to generate multiple customized output clocks, employing a series of programmable dividers. The MMCM architecture incorporates a programmable counter divider (D) at the input stage. This divider functions as a scaling mechanism, reducing the input clock frequency by a user-defined integer value. Following the initial scaling by divider D, the MMCM distributes the processed clock signal to a network of programmable output dividers (O_j and M). Dividers O_1 to O_6 operate as integer dividers, offering a precise reduction of the clock signal to generate various output clock frequencies tailored to specific application requirements. For even finer control over the output frequencies, the MMCM incorporates fractional counters O_0 and M. These fractional counters function similarly to their integer counterparts but allow adjustments in smaller increments, typically 0.125. This fractional division capability empowers the MMCM to create a broader range of highly precise output clock frequencies,

particularly for the O_0 output.

While the MMCM primitive's programmable counters offer wide value ranges, adherence to AMD Xilinx's prescribed limitations is crucial for ensuring MMCM's stability.

3 | Cryptographic Operations Segmentation in Real-World Scenarios

Segmenting COs within side-channel traces is a challenging and underexplored step in the side-channel analysis workflow. While traditional laboratory setups often rely on dedicated hardware triggers, real-world scenarios deal with commercial devices lacking such infrastructure whose measurements are more susceptible to noise, timing variability, and often protected by obfuscation techniques, making it difficult to accurately locate and align the relevant operations. Despite its critical role, segmentation is frequently overlooked in the literature, leaving open problems in automating this process, handling highly desynchronized traces, and benchmarking solutions on realistic datasets. Addressing these challenges is essential for advancing side-channel security in modern digital systems.

Hound is a flexible and adaptable methodology designed for segmenting COs within large side-channel traces containing millions of samples. Following testing and validation on controlled problems, the necessity for a more robust approach emerged. These improvements led to the development of three versions over time, each introducing new strategies to address the increasing complexity of obfuscation countermeasures encountered in real-world scenarios.

- **Hound v1** was initially conceived to tackle the random delay countermeasure [CGL+24]. This first version introduced the core deep-learning approach for CO segmentation, focusing on identifying the start of cryptographic operations in the presence of desynchronization.
- **Hound v2** extended the methodology to handle DFS, a more challenging countermeasure that alters the timing and morphology of side-channel traces [GCZ24]. Hound v2 was later further generalized to support other obfuscation mechanisms such as morphing and chaffing, in addition to random delay [GCZ25].
- **Hound v3** introduced the capability to detect not only the start but also the end

of each CO. This version also integrates heuristics to correct segmentation errors, such as false positives and false negatives, making the approach more robust and suitable for real-world scenarios [GCZ25].

This chapter begins by introducing Hound as common core methodology in Section 3.1. Such section details each step of Hound's segmentation and then introduces the differences between the various versions. Finally, Section 3.2 presents heuristics for correcting identification errors to enhance overall robustness.

3.1. Hound: deep learning methodology for cryptographic operations segmentation in obfuscated side-channel traces

In contrast to state-of-the-art techniques that rely on consistent manual templates for CO identification [TBW+21, BFP22], Hound exploits deep learning to learn the optimal CO templates to handle the variability generated by hiding countermeasures. The learned template is then convolved with a new side-channel trace to identify samples corresponding to a CO.

Figure 3.1 depicts the proposed Hound methodology consisting of a training and inference pipeline. The individual stages within the pipelines have been subject to adjustment during Hound evolution based on the targeted obfuscation mechanism, particularly with

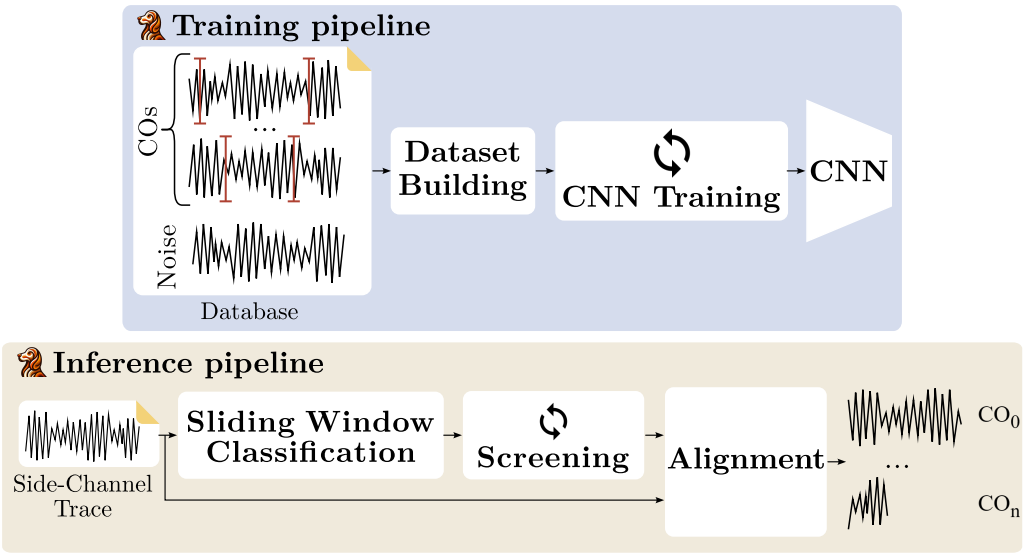


Figure 3.1: Overview of the proposed Hound pipeline for segmenting cryptographic operations in obfuscated side-channel traces, divided into *training* and *inference* pipelines.

regard to the creation of the training dataset and the `Screening` procedure. The overall architecture of the `CNN`, its training algorithm, the `Sliding Window Classification`, and the `Alignment` stage are consistent across all Hound's versions, ensuring a unified approach to CO segmentation.

Threat model Similarly to profiled attacks, we assume the attacker has access to an identical copy of the target device. However, we consider a realistic scenario where the attacker can only run chosen applications and measure the resulting side-channel signals. The attacker does not have full control over the cloned device, i.e., they cannot enable or disable the obfuscation countermeasure, nor can they alter or tamper the hardware by adding trigger pins.

3.1.1. Training pipeline

Hound trains a neural network to classify individual windows from large side-channel traces. While precisely identifying the entire CO is desirable, the primary objective is to accurately detect and isolate COs from background noise. To achieve this, the training pipeline develops a robust CNN classifier to label side-channel trace windows as either *beginning of a CO* or not. Hound's training approach is designed to identify the start of COs without disabling any obfuscation countermeasures during side-channel measurements. The attacker is assumed to have access to an identical device, which is used to collect both noise traces and cipher traces for training, but cannot bypass the active countermeasures. Noise traces are recorded by running general-purpose applications that do not perform cryptographic tasks. Such data is essential for teaching the neural network to distinguish between the start of COs and other types of computation. Cipher traces are recorded during the execution of a single CO, with the attacker able to select the plaintext and secret key. All traces are collected with the obfuscation mechanism enabled, as the attacker cannot deactivate it. Figure 3.1 shows the training pipeline. Raw traces are processed by the `Dataset Building` stage, which creates a dataset of N-sample windows from both noise and cipher traces. This dataset is then used to train the `CNN` classifier.

Dataset building

The first stage in the training pipeline is `Dataset Building`, which receives a collection of side-channel traces and generates a dataset suitable for CNN training. On a cloned platform, the attacker collects a noise trace, i.e., running general-purpose applications, and CO traces with the obfuscation mechanism remaining active during all trace collection as it cannot be disabled.

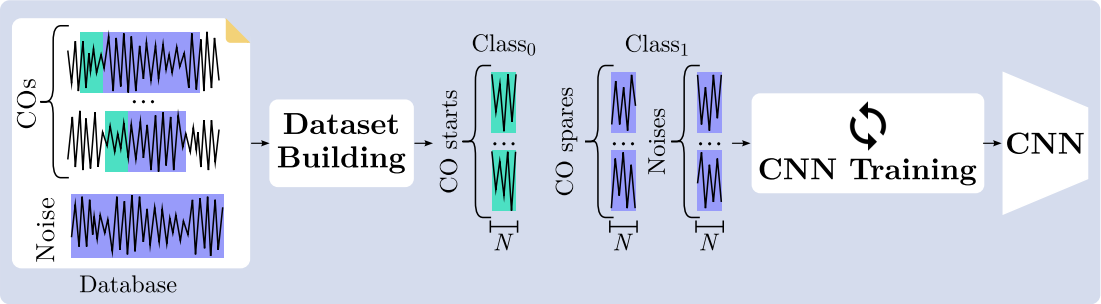


Figure 3.2: Hound v1's training pipeline.

For each CO i of length L_i samples, the starting N samples are tagged as *CO starts*. The remaining $L_i - N$ samples are equally split into consecutive windows of width N and tagged as *CO spares*. Additionally, a random set of N-sample windows are extracted from the noise trace and tagged as *noises*. Tagging the traces in these distinct categories, i.e., *CO starts*, *CO spares*, and *noises*, is crucial for enabling the CNN to identify just the beginning of COs, rather than their entirety. The granular tagging allows for the precise identification and isolation of consecutive COs, as well as those interleaved with noisy general-purpose applications. The three window's tags can then be mapped into two or three different target labels for the CNN, depending on the specific methodology employed. The number of collected CO traces, the length of the noise trace, and the size N of the time windows are all configurable parameters.

Hound v1's dataset This dataset is specifically designed for the random delay countermeasure, which involves random alternation of NOP instructions with consecutive program instructions to create desynchronization. Historically, Hound was originally conceived to overcome this type of obfuscation during the segmentation process [CGL+24]. Under the assumed threat model, the attacker cannot use trigger pins or disable desynchronization, but can execute chosen applications and measure the resulting side-channel signals on the clone device. To facilitate dataset creation, a sequence of NOP instructions is inserted at the start of each CO during the acquisition. The clear difference in power consumption between the NOPs and the actual CO execution enables straightforward identification of the CO's start within each cipher trace, without requiring a triggering infrastructure. This approach is solely used for labeling during training; once Hound v1 is trained, it can segment traces on the target device without requiring any NOP insertion.

Following the general `Dataset Building` description, the collected side-channel traces are chunked into N-sample windows and tagged as *CO starts*, *CO spares*, and *noises*. These three categories are then mapped to two distinct classes, i.e., *Class 0* representing the beginning of CO, and *Class 1* for everything else, as shown in Figure 3.2.

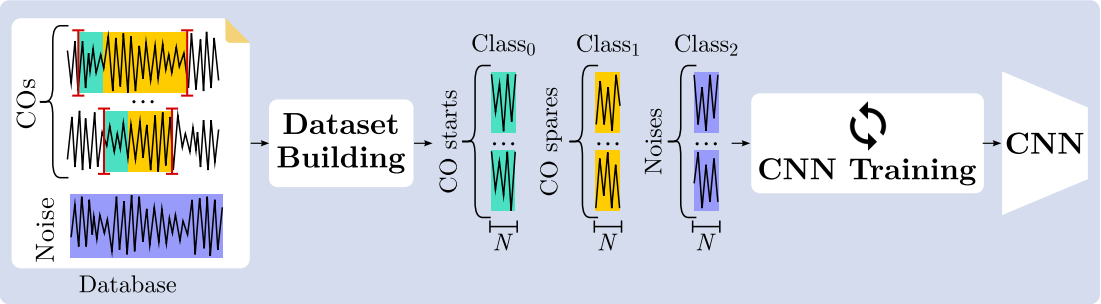


Figure 3.3: Hound v2 and v3's training pipeline.

Hound v2 and v3's dataset Hound v2 was conceived to address DFS countermeasures, which aim to randomly change the clock frequency of the computing platform, profoundly altering operation timing and continuously morphing the side-channel signal [GCZ24].

In this scenario, the use of NOP instructions for marking CO beginnings as in Hound v1 is ineffective since DFS significantly alters power consumption independently of NOP or CO execution, rendering them indistinguishable. To circumvent this, the attacker is assumed to retain the ability to choose and modify the executed application, thus triggering a chosen GPIO at the CO's execution, similar to realistic scenarios discussed in [LKMM21] and [QWY+24]. Alternatively, by probing the clone device, the attacker can detect when the CO initiates, facilitating precise labeling of the beginning of each CO on the clone. In contrast, probing is not required for the noise trace. Once trained, the Hound v2 works on the target architecture that implements DFS without the need for probing or binary modification. Notably, DFS introduces a significant amount of randomization in the side-channel trace. To address this, the `Dataset Building` procedure has been adaped to label the side-channel collection into three classes, i.e., *CO starts*, *CO spares*, and *noises*, as shown in Figure 3.3. This modification allows the CNN to learn the pattern of the highly obfuscated side-channel trace better, maximizing the discrimination between the start of the CO and the other two classes.

This way of structuring the dataset was then also shown to be effective in dealing with morphing and chaffing obfuscation, as well as random delay [GCZ25].

CNN architecture

The architecture of the CNN, as illustrated in Figure 3.4, is specifically designed to process one-dimensional side-channel traces and robustly classify each input window. The network is adapted from a 2D ResNet [HZRS16] to use the residual learning framework for the analysis of 1D sequential data [WYO17]. A 1D ResNet architecture has been chosen

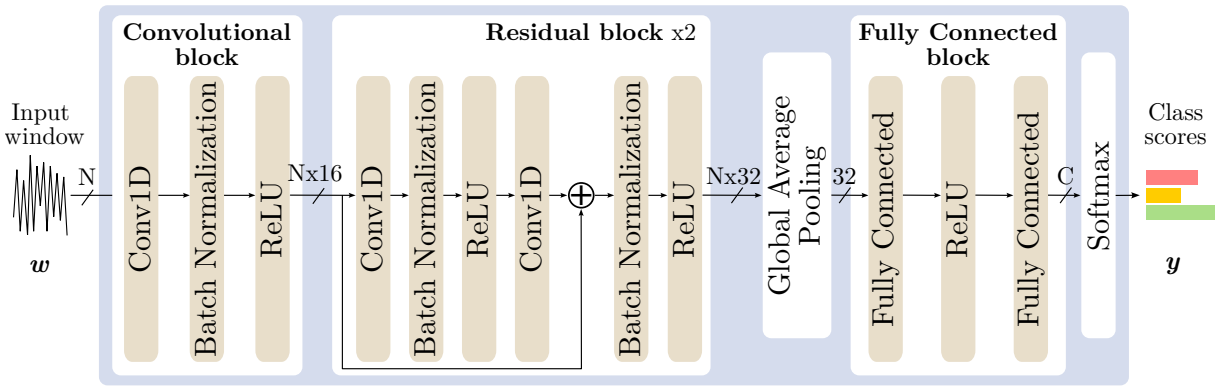


Figure 3.4: Hound's convolutional neural network architecture.

for its effectiveness in processing one-dimensional data like side-channel traces, capturing local and global dependencies. By design, ResNet are intrinsically translation invariant, making them the ideal choice for analyzing desynchronized signals caused by obfuscation techniques. Moreover, the ResNet structure facilitates efficient gradient propagation, improving the training procedure, which is crucial for extracting subtle patterns from 1D time-series.

The CNN takes a N-sample window w from the side-channel trace as input and outputs a classification score vector y. The CNN's structure is organized into six sequential components: a convolutional block, a pair of residual blocks, a global average pooling layer, a fully-connected block, and a softmax layer.

The network begins with an initial convolutional block, which consists of a 1D convolutional layer with 16 filters followed by batch normalization [IS15] and a ReLU activation function [NH10]. This block is responsible for extracting low-level features from the raw input window of length N. The convolutional layer uses a kernel size of 64, a stride of 1, and zero padding to preserve the temporal dimension to N, and it outputs 16 feature maps, i.e., the shape of the output tensors is $N \times 16$.

Following the initial block, the architecture includes two consecutive residual blocks [HZRS16], designed to enhance feature learning and mitigate the vanishing gradient problem. Each residual block contains two convolutional sub-blocks. The first residual block maintains 16 filters, while the second increases the number of filters to 32, allowing the network to learn more complex representations. Shortcut connections are employed within each residual block, enabling the input of the block to be added element-wise to its output, thus facilitating the flow of information and gradients through the network. The shape of the output tensor after the two residual blocks is $N \times 32$.

After the residual blocks, a global average pooling layer is applied averaging the value

of each feature map across the entire temporal dimension N, effectively summarizing the learned features and reducing the data dimensionality to 1×32. Notably, the structure of the global average pooling layer allows the potential use of different N values for the training and attack pipelines. The resulting feature vector is then passed to a fully connected block, which consists of two dense layers. The first dense layer features 32 neurons and a ReLU activation, while the second layer consists of C neurons, where C is the number of classes. These dense layers further process and combine the extracted features into an encoded representation, ready for the final classification.

Finally, the output of the fully connected block is fed into a softmax layer, which produces a probability distribution over the possible C classes. The network outputs a C-dimensional vector, with each element representing the classification score for one of the classes. This architecture enables the network to effectively distinguish between different segments of the side-channel trace, even in the presence of obfuscation and noise.

CNN training The designed CNN is trained using N-sample side-channel windows extracted from the available trace collection. The network parameters θ are optimized through an iterative procedure. At each step, a trace window w is fed into the network, producing an output $y = g(w, \theta)$, where g is the non-linear function implemented by the network. The output y represents the predicted class probabilities over the C possible classes for the input window. To guide the training, the CNN's output probabilities are compared with the ground truth labels c using the cross-entropy loss, as shown in Equation 3.1.

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{c}) = -\sum_{j=1}^{C} \boldsymbol{c}_j \log(\boldsymbol{y}_j) = -\sum_{j=1}^{C} \boldsymbol{c}_j \log(g_j(\boldsymbol{w}, \boldsymbol{\theta})) \tag{3.1}$$

The loss value is then used to update the network parameters θ, progressively improving its ability to distinguish between different patterns in the data.

3.1.2. Inference pipeline

The inference pipeline relies on the trained CNN to locate COs in a new side-channel trace captured from the target device. The pipeline is composed of three main stages, as illustrated in Figure 3.5: `Sliding Window Classification`, `Screening`, and `Alignment`. Given a single side-channel trace, the pipeline segments it to identify the start of each CO. First, the `Sliding Window Classification` stage slices the trace into overlapping windows and classifies each window using the trained CNN. Based on these classifications, i.e., the segmentation of the input trace, the `Screening` stage applies post-processing to determine the time points where each CO begins. Finally, the `Alignment` stage uses these

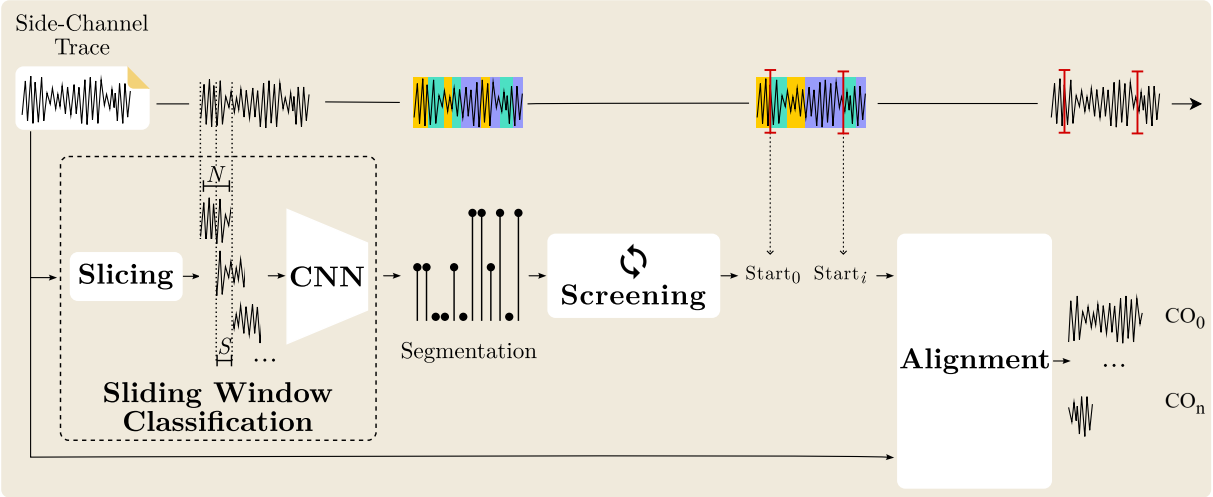


Figure 3.5: Hound's inference pipeline.

time points to split the original trace, organizing the detected COs for further analysis.

Sliding window classification

The first stage in the inference pipeline is `Sliding Window Classification`, which is visually represented in Figure 3.5. Here, the input is a raw side-channel trace which is systematically sliced into overlapping windows through the *Slicing* process. The sliced windows have fixed length N and a configurable stride s of overlap. Each window is then passed to the trained CNN to be independently processed. The CNN outputs a classification score for each input, assigning a probability distribution over the possible classes, e.g., *CO starts*, *CO spares*, or *noises*.

Screening

The CNN's output can be noisy and may not directly pinpoint CO locations. Therefore, the inference pipeline includes a subsequent `Screening` stage to refine these results (see Figure 3.5). This stage applies heuristics and post-processing algorithms to filter spurious detections and precisely identify true CO start points, significantly improving segmentation precision, especially with obfuscated side-channel traces. The specific screening procedure, however, may vary depending on the hiding countermeasures implemented in the trace, as detailed in the following for Hound v1 and Hound v2.

Hound v1's screening The CNN in `Sliding Window Classification` stage outputs a signal where each sample is a classification value. The softmax output of the CNN is a probability distribution of classes, which has been observed to hide a structured

Algorithm 3.1 Hound v1's screening algorithm.

Input: *seg*, *s*, *k*
Output: *starts*

1: *threshold* ← `mean`(*seg*) ▷ Convert to Square Signal
2: *squareSignal* ← [ ]
3: **for all** *sample* **in** *seg* **do**
4: **if** *sample* < *threshold* **then**
5: *squareSignal*.`update`(−1)
6: **else**
7: *squareSignal*.`update`(1)
8: *square_signal* ← `medianFilter`(*squareSignal*, *k*) ▷ *k*: kernel size
9: *starts* ← [ ]
10: **for all** *i*, *sample* **in** `enumerate`(*squareSignal*) **do** ▷ Extract Edges
11: **if** *sample* is rising edge **then**
12: *starts*.`update`(*i* × *s*)
13: **return** *starts*

recurrent pattern that can be exploited for locating the COs. The pattern is more visible in the linear output of the fully-connected block, which makes localization easier. Thus, Hound v1 considers as inference CNN output the fully connected score outcome of class one. Depending on whether the N-sample window has been classified as the beginning of the CO or not, a higher or lower class score is assigned to the corresponding windows. Notably, the CNN can produce a noisy output that cannot be directly used to infer the precise location of the COs. To this end, the linear output of the CNN is polished by the subsequent `Screening` stage in the inference pipeline (see Segmentation in Figure 3.5). Algorithm 3.1 highlights the screening procedure used in Hound v1.

The screening algorithm (see Algorithm 3.1) is fed with the sliding window classification output *seg* and outputs the list of samples *starts* identifying the beginning of each CO in the processed side-channel trace.

At first, the algorithm transforms *seg* into a square wave signal by comparing each sample of *seg* with a threshold. For each sample in *seg*, a corresponding output of value -1 or +1 is defined depending if the sample is below or above the threshold value, respectively (see lines 3-9 in Algorithm 3.1).

Then, a median filter is applied to improve further the obtained square wave signal (see line 10 in Algorithm 3.1). The median filter is fed with the square wave signal, i.e., *squareSignal*, and a size k, which gives the size of the median filter window. The window slides over the input square wave signal for which each sample is replaced with the median of the k neighbors.

Algorithm 3.2 Hound v2's screening algorithm.

```
    Input: seg, s, k, avgCO
    Output: starts
 1: starts ← []
 2: do
 3:     polishedSeg ← POLISH(seg, k)
 4:     newStarts ← EXTRACT(polishedSeg, s)
 5:     k, seg ← REFINE(k, avgCO, newStarts, seg)
 6:     starts.update(newStarts)
 7: while seg not empty && k ≥ 1
 8: return starts
 9:
10: procedure POLISH(seg, k)
11:     polishedSeg ← majorityFilter(seg, k)
12:     return polishedSeg
13:
14: procedure EXTRACT(seg, s)
15:     newStarts ← [ ]
16:     for i, sample in enumerate(seg) do
17:         if sample is falling edge to class 0 then
18:             newStarts.update(i × s)
19:     return newStarts
20:
21: procedure REFINE(k, avgCO, starts, seg)
22:     k ← refineK(k)
23:     minCO ← refineMin(avgCO, starts)
24:     refineSegs ← []
25:     for s, s_next in starts do          ▷ Look if can be still two consecutive COs
26:         if s_next − s > 2 × minCO then
27:             subSeg ← seg[s : s_next]
28:             refineSegs.update(subSeg)
29:     return k, refineSegs
```

At last, the algorithm returns the number of the sample identifying the rising edges, i.e., the points in the obtained square wave signal where two consecutive samples assume a value of -1 and +1 respectively (see lines 12-16 of Algorithm 3.1). The number of the sample is multiplied by s, i.e., the stride value used during the sliding windows classification. Such points are identified as the beginning of each CO in the analyzed input trace and returned as *starts*.

Hound v2's screening The CNN's softmax output in `Sliding Window Classification` is a 3-dimensional probability distribution of classes, classifying each N-sample window

as *start of the CO*, *spare part of the CO* or *noise*. The Hound v2's inference process considers the class with the highest probability outcome among the three as the CNN's output. Algorithm 3.2 outlines the screening procedure in Hound v2.

The screening algorithm takes the segmentation output from the `Sliding Window Classification` stage, denoted as *seg*, and some parameters, i.e., the stride s used for the sliding window, the kernel size k for the polish procedure (see line 10 of Algorithm 3.2), and the average CO length $avgCO$. In the end, it returns the list *starts* of samples corresponding to the beginning of each CO instance in the processed side-channel trace.

The algorithm is an iterative process that involves three main steps: `POLISH`, `EXTRACT`, and `REFINE`. The core idea behind Algorithm 3.2 is to start with an aggressive polish and incrementally refine it by attempting to uncover new COs. The polishing step (see line 10 of Algorithm 3.2) employs a majority filter across the whole segmentation. This filter traverses the input signal for a given kernel size k, replacing each point with the most frequently occurring value within a k-sized window. The extraction step (see line 15 of Algorithm 3.2) identifies the initial sample of each CO. It pinpoints the indices of the falling edge down to class 0, meaning those samples with a value of 0 where the preceding value is not 0. These samples are then scaled by s, i.e., the stride in the sliding window classification. Finally, the refining step (see line 25 of Algorithm 3.2) fine-tunes the algorithm's parameters by reducing kernel size k and generating a list of sub-segmentation for subsequent iterations. These sub-segments encompass those regions of the input segmentation for which multiple hidden COs might still exist. The process begins by determining the minimum length of a CO $minCO$ (see line 25 of Algorithm 3.2), which is derived by comparing the input's average CO length and the CO's starting points identified so far. Next, the algorithm examines pairs of consecutive CO starting points (see line 28 of Algorithm 3.2). If the distance between them exceeds twice the $minCO$, it indicates the potential presence of additional COs within that interval. Consequently, the corresponding sub-segmentation is queued for further processing.

The algorithm persists until no further refinement is possible, i.e., when k drops below one or the refining step yields an empty list of sub-segmentations *seg* (see line 7 of Algorithm 3.2).

Hound v3's screening The third version of screening procedure follows the iterative approach described for Hound v2, using a majority voting mechanism to select the most likely class and iteratively polishing the segmentation (see Algorithm 3.2). However, the `EXTRACT` step has been improved to enable the extraction of both CO start and end points from the CNN output, as detailed in Algorithm 3.3. In fact, although the CNN classifier

Algorithm 3.3 Hound v3's EXTRACT procedure for detecting CO's end points.

Input: *seg*, *s*
Output: *newStarts*, *newEnds*

```
1: procedure EXTRACT(seg, s)
2:     newStarts ← [ ]
3:     newEnds ← [ ]
4:     for i, sample in enumerate(seg) do
5:         if sample is falling edge to class 0 then
6:             newStarts.update(i × s)
7:         if sample is rising edge to class 2 OR sample is rising edge to class 1 then
8:             newEnds.update(i × s)
9:     return newStarts, newEnds
```

has been specifically designed to detect the beginning of the CO's, it is capable of detecting the end of the CO executions by looking at *Class 1*, as the classifier has been trained to identify the spare part of the COs.

The extraction step identifies the indices of falling edges to class 0 as CO start points, and rising edges to class 1 or 2 as CO end points. Each detected index is scaled by s, the sliding window stride, to map to the original trace. This dual extraction enables more precise segmentation by providing both start and end markers for each CO.

The refining step is also improved by leveraging the availability of both start and end points. Instead of analyzing intervals between consecutive start points, the algorithm now considers start-end pairs, allowing for more accurate detection of missed or spurious COs and better handling of segmentation ambiguities.

Alignment

Finally, the `Alignment` stage used the refined start indices from the screening stage are used to segment the original side-channel trace into individual COs. The output of this stage is a set of temporally aligned trace segments, each containing a single CO. This alignment enables further analysis or attacks on the isolated cryptographic operations.

3.2. Heuristics for segmentation errors

Hound's inference pipeline may produce errors when the CNN misclassifies a window or when the `Screening` stage incorrectly extracts start or end points. These mistakes lead to false positives, i.e., detecting a CO where none exists, or false negatives, i.e., missing a real CO. To improve segmentation accuracy, Hound v3 [GCZ25] applies two

practical heuristics that leverage the detection of both start and end points of the CO (see Algorithm 3.2 and Algorithm 3.3).

For false positives, the heuristic pairs each detected start with the closest following end, ensuring that every start has a corresponding end and vice versa. If a start does not have a matching end, it is flagged as an error. The algorithm then removes the start point that is associated with the shortest interval, as it is unlikely to represent a valid CO. Similarly, any end point without a preceding start is marked and handled.

For false negatives, the heuristic checks the intervals length between matched start and end points. If an interval is much longer than the average CO duration, it suggests that a CO may have been missed. In such cases, the algorithm splits the long interval and examines the new segments to verify that they do not belong to general-purpose applications.

These approaches help recover missed COs and improve overall segmentation reliability.

4 | Attack and Defense Strategies Using Deep Learning

Side-channel attacks and their corresponding defenses are in continuous investigation. While classical attack techniques have demonstrated the ability to extract sensitive information from cryptographic devices, the adoption of countermeasures such as masking and hiding has made practical exploitation more difficult. However, the rapid evolution of deep learning and machine learning methods has exposed new vulnerabilities, allowing attackers to bypass protections previously considered robust.

Despite these advancements, several open challenges remain. Many countermeasures, especially those based on Dynamic Frequency Scaling (DFS) and run-time variability, introduce significant complexity in trace alignment and analysis, but often come with trade-offs in performance and resource usage. Furthermore, the lack of realistic datasets and open-source platforms has limited the reproducibility and benchmarking of both attacks and defenses in real-world scenarios. This chapter addresses these challenges by introducing novel methodologies and hardware solutions to improve both the effectiveness of SCA techniques and the resilience of countermeasures for systems that employ DFS. Through the development of deep-learning-assisted attack pipelines and efficient hardware modules for dynamic frequency randomization, the work advances the state-of-the-art in side-channel security and provides practical tools for evaluating and enhancing the protection of modern digital systems with lightweight hiding countermeasures.

The chapter is organized as follows: Section 4.1 presents Owl, a novel deep learning-assisted SCA against highly desynchronized traces through a commercial DFS implementation, Section 4.2 presents Rabbit, a novel hardware architecture for dynamic clock randomization designed to overcome the limitations of existing approaches while ensuring robust security against various SCAs.

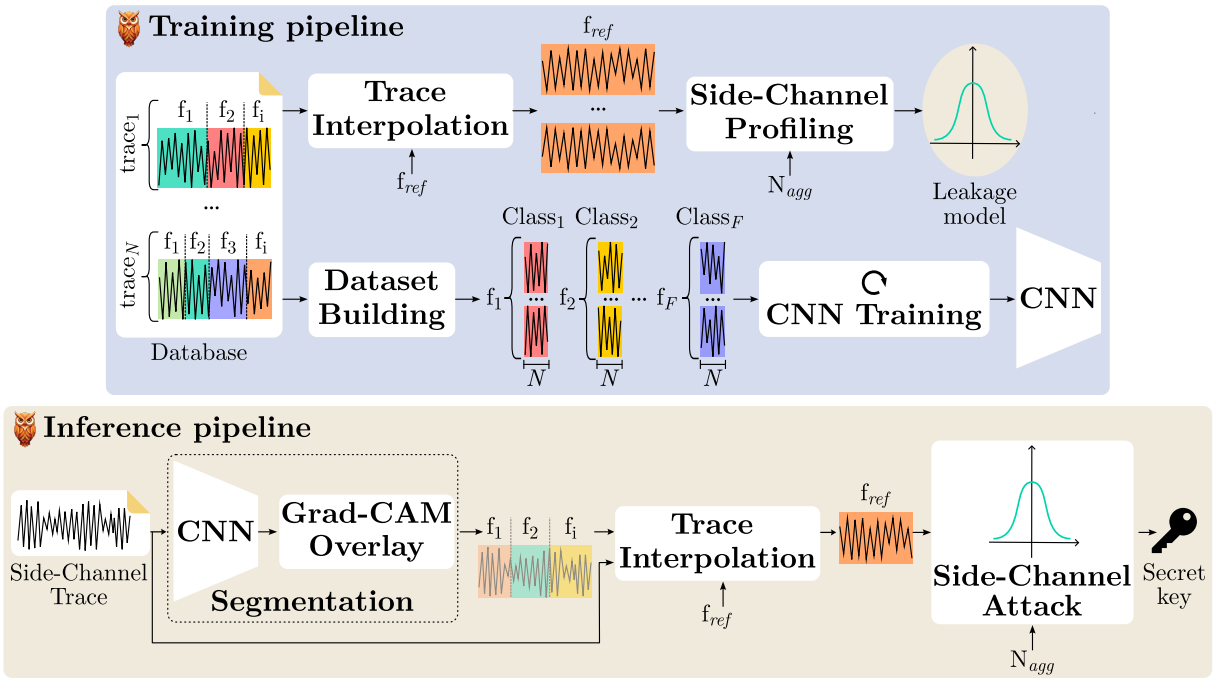


Figure 4.1: Overview of the proposed Owl pipeline for resynchronizing DFS-desynchronized side-channel traces, divided into *training* and *inference* pipelines.

4.1. Owl: deep-learning-assisted side-channel attack against dynamic frequency scaling countermeasures

DFS is a widely available, cost-effective actuator for managing power-performance trade-offs in embedded systems. Its ability to select the operating frequency at run-time from a small set of supported values, together with its presence on many edge and IoT platforms [UEF22, Ras23], makes DFS a realistic and attractive hiding countermeasure, and, consequently, a practical attack surface. Only a few side-channel studies explicitly target DFS-based defenses. Common practice models desynchronization with simple data-augmentation techniques, such as rigid time shifts [BPS+20]. While augmentation can mitigate limited data availability, the chosen augmentation model must be proven to be representative of the physical phenomena that produce desynchronization. Moreover, the literature shows that attack complexity grows with the degree of temporal desynchronization and that attacks can fail beyond a certain desynchronization level. Thus, studying highly desynchronized traces produced by real, commercial-like DFS actuators remains a critical and open research problem.

To address this gap, we propose Owl, a deep-learning-assisted attack pipeline that resyn-

chronizes traces strongly desynchronized by physical DFS actuators. Owl applies a deep-learning pre-processing stage to resynchronize traces to a reference clock frequency and then performs a classical SCA on the resynchronized data. Notably, once temporal alignment is restored, any established side-channel analysis technique can be applied. In this work, a template attack [CRR03] is chosen due to its simplicity and proven effectiveness.

Owl overview Starting from highly desynchronized traces, the proposed methodology implements a pre-processing deep learning stage to resynchronize the traces to a reference clock frequency before validating the resynchronization via classical SCA, such as a template attack. Notably, the proposed methodology exploits trained stages, i.e., the deep learning and the leakage profiling stages, thus, a more natural presentation flow requires organizing the discussion into training and inference phases as depicted in Figure 4.1. In the training pipeline, a CNN-based frequency classifier and leakage models are trained using a dataset of power traces paired with clock frequency labels. The process involves resynchronizing traces to a reference frequency using interpolation, followed by minor aggregation to reduce errors. The CNN is trained on time windows with constant frequency, leveraging the availability of precise frequency labels. The inference pipeline implements the Owl approach, which uses a deep learning pre-processing pipeline to resynchronize highly desynchronized traces. This involves segmenting traces using a frequency classifier and Grad-CAM to assign frequency labels to each sample. The traces are then realigned to the reference frequency through interpolation and aggregation before performing a classical SCA.

This methodology enables effective handling of desynchronized traces by combining deep learning-based segmentation and classical side-channel analysis techniques.

The rest of this section delivers a detailed discussion targeting the dataset creation, the trace interpolation, and the CNN for frequency classification in Section 4.1.1, and targeting inference pipeline and the segmentation in Section 4.1.2. Conversely, in the next Section 4.2 we present a practical defense that targets the same threat model.

4.1.1. Training pipeline

The training phase is meant to train a CNN frequency classifier and profile the side-channel leakages starting from an input dataset of pairs. Each pair consists of a highly desynchronized power trace and the corresponding set of temporally ordered labels identifying the clock frequency of each interval in the trace. The leakages are profiled using a two-stage pipeline. First, the `Trace Interpolation` stage resynchronizes all the traces of the in-

put dataset to a common reference clock frequency. Notably, the `Trace Interpolation` stage leverages the clock frequency labels associated with each sample of the input traces to perform the resynchronization. Second, the leakages are profiled using the resynchronized traces. Leveraging sliding window alignment [CCD00], a minor aggregation of N_{agg} samples over time is performed to mitigate interpolation errors during the `Side-Channel Profiling` stage.

The CNN is trained using a two-stage pipeline with the final goal of predicting the clock frequency of a given input trace. To this end, we employed a `Dataset Building` stage to create the training dataset by selecting a set of time windows from each trace which exhibit a constant clock frequency. The availability of a database with clock frequency labels for each trace sample makes this `Dataset Building` procedure a trivial task. Finally, starting from the extracted constant clock frequency time windows, the `CNN Training` stage delivers the `Trained CNN`.

Dataset Building

Considering the training pipeline, the `Dataset Building` stage takes the side-channel traces in input and generates the dataset to train, test, and validate the CNN (see `Training Phase` in Figure 4.1). The CNN dataset is built starting from the available collection of side-channel measurements, consisting of power traces desynchronized through DFS and a frequency label for each sample. Thus, each initial power trace encloses multiple frequencies, with possible values from a predefined finite set $\mathcal{F}$. The neural network is trained to classify each power trace according to its clock frequencies, which necessitates training data with constant frequency. The `Dataset Building` procedure inputs a desynchronized side-channel trace and selects N-sample windows of constant frequency. Since the starting database provides the frequency value for each sample, selecting a constant frequency window is trivial. Padding is left before and after each frequency change to avoid border effects due to the physical actuator. The output CNN dataset consists of N-sample constant-frequency windows with relative frequency labels.

Trace Interpolation

The `Trace Interpolation` stage is used both during the training phase and the inference phase to resynchronize the traces to a reference clock frequency (f_{ref}). The procedure employed to interpolate the trace is reported in Algorithm 4.1. The algorithm takes as input the desynchronized trace t, the segmentation s, and the desired reference frequency f_{ref}. The input segmentation s consists of a vector of frequency values for each sample

Algorithm 4.1 Owl's interpolation algorithm.

Input t, s, f_{ref} ▷ Input trace, segmentation, reference frequency
Output t_{int} ▷ Interpolated trace

1: **procedure** TRACEINTERPOLATION(t, s, f_{ref})
2: $windows \leftarrow$ `getConstantFrequencyWindows`(t, s)
3: $t_{int} \leftarrow [\,]$
4: **for** win in $windows$ **do**
5: $f_{\text{win}} \leftarrow$ `getFrequency`(win)
6: **if** $f_{\text{win}} = f_{\text{ref}}$ **then** ▷ No interpolation required
7: $t_{int} \leftarrow$ `concat`(t_{int}, win)
8: **else** ▷ Interpolation required
9: $win_{int} \leftarrow$ `scaleWindow`$(win, f_{ref}/f_{win})$
10: $t_{int} \leftarrow$ `concat`(t_{int}, win_{int})
11: **return** t_{int}
12: **procedure** SCALEWINDOW($window$, $freqRatio$)
13: $samples \leftarrow$ `len`$(window)$
14: $numSample_{int} \leftarrow samples/freqRatio$
15: $samples_{int} \leftarrow$ `newList`$(0, numSample_{int})$ ▷ List from 0 to $numSample_{int}$
16: $window_{int} \leftarrow$ `interp1D`$(window, sample, sample_{int})$ ▷ Linear interpolation
17: **return** $window_{int}$

in t. Notably, during the training phase, s is known and provided directly as part of the available database. Instead, during the inference phase, s is estimated by the proposed segmentation method (see `Segmentation` stage in Figure 4.1). Given the input segmentation, all the intervals characterized by a single clock frequency, named *windows* in the algorithm, are extracted from the original trace (see line 2 in Algorithm 4.1). The algorithm considers then one window at a time and interpolates based on the window frequency (f_{win}). If f_{win} is the same as the reference frequency, then the interpolation is unnecessary since the window is already aligned with the reference frequency (see lines 7-9 in Algorithm 4.1). In that case, the samples within the original window are directly stored in the output array t_{int}. On the contrary, in case f_{win} and f_{ref} are not the same, a linear interpolation is performed, represented by the *scaleWindow* function (see lines 10-13 and 17-25 in Algorithm 4.1). The interpolation scales the considered window by a factor $f_{\text{ref}}/f_{\text{win}}$, thus increasing or decreasing the number of samples in the final trace. The obtained scaled window is then stored in the output array t_{int}. The procedure is repeated for each window until all the constant-frequency windows in t have been processed. Finally, the interpolated trace t_{int} is returned (see line 15 in Algorithm 4.1).

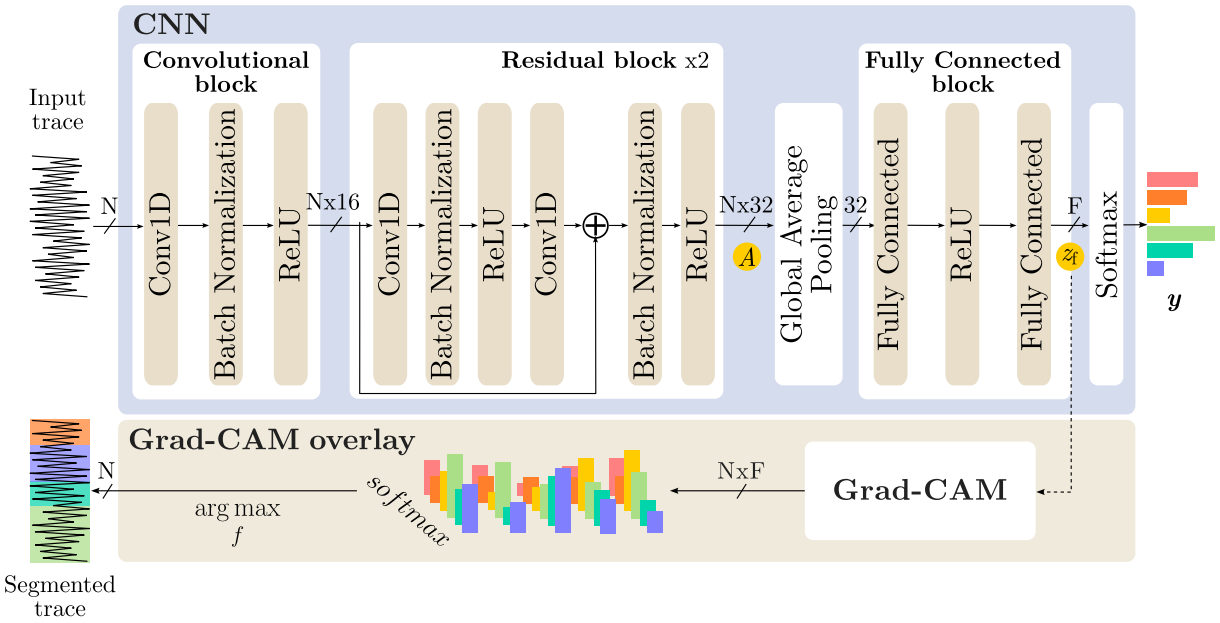


Figure 4.2: Owl's frequency classification CNN and Grad-CAM overlay architecture.

Convolutional Neural Network

The CNN follows the same architecture illustrated in Section 3.1.1, which is adapted from a 2D ResNet [HZRS16] and illustrated in Figure 4.2. The network aims to classify the operating frequency of the input side-channel trace. The input to the CNN is a power trace of N samples, while its output is a classification vector (y) with a probability for each frequency in the training label set $\mathcal{F}$. The CNN architecture consists of six consecutive blocks: a convolutional block, two residual blocks, a global average pooling layer, a fully connected block, and a softmax layer. Each convolutional block includes a 1D convolutional layer, a batch normalization layer, and a ReLU activation function. The residual blocks, which contain two convolutional blocks each, contain shortcut connections for element-wise feature summation. All convolutional layers employ a kernel size of 64, a stride of 1, and zero padding. The initial convolutional layer and the one in the first residual block use 16 filters, while the second residual block increases the filter count to 32. The global average pooling layer computes the average of the extracted features over the temporal dimension N. The resulting feature vector is then passed through two fully-connected layers, using a ReLU activation function, before being passed to the softmax layer, which generates a F-dimensional vector containing the classification scores for each frequency class. Notably, the structure of the global average pooling layer allows the potential use of different N values for the training and inference pipelines.

CNN training The CNN is trained on constant-frequency windows of N samples (see Section 4.1.1). Given an input window t, the network outputs a probability vector $y = g(t)$ over the F frequency classes. Training minimizes the cross-entropy loss between y and the true frequency label f (see Equation 3.1).

4.1.2. Inference pipeline

The inference pipeline consists of a pre-processing deep learning pipeline (see `Segmentation` and `Trace Interpolation` stage in Figure 4.1) to resynchronize the traces to a reference clock frequency (f_{ref}) and a classical side-channel attack (see `Side-Channel Attack` stage in Figure 4.1). The pre-processing deep learning pipeline consists of three stages. The first and second stages implement a frequency classifier and a network explanation tool used to identify the operating frequency associated with each time window of a desynchronized trace (see `Segmentation` stage in Figure 4.1). Notably, the frequency classifier has been trained using constant clock frequency traces, while in the Attack Phase, it is fed with highly desynchronized traces. Thus, the output vector is expected to highlight various clock frequencies associated with each desynchronized trace. The main novelty of the proposed approach is to avoid the direct use of such output vector. In contrast, we feed the output vector obtained from the frequency classifier into the Grad-CAM CNN explanation tool. Grad-CAM highlights the portions of a desynchronized trace that activated a specific frequency class in the output vector, thus allowing to associate a clock frequency label to each sample of the desynchronized traces. Starting from the desynchronized traces and the clock frequency labels identified in the `Segmentation` stage, the third stage of the deep learning pipeline applies the `Trace Interpolation` stage to realign each trace to the reference clock frequency before performing the actual attack. An N_{agg}-samples aggregation over time is used to fix minor misalignments due to the rough frequency segmentation and to mitigate interpolation errors [CCD00] (see `Side-Channel Attack` stage in Figure 4.1). Notably, in this work a template attack is performed as an example of a classical SCA technique that can be applied after the deep learning pre-processing pipeline.

Segmentation

Considering the inference pipeline, the trace segmentation block (see `Segmentation` stage in Figure 4.1) assigns a frequency value to each sample in the attacked desynchronized traces. Given an input trace t with N samples in the trace, the segmentation is the 1D vector s of N elements with values $s_i \in \{f_1, f_2, \ldots, f_F\}$, where f denotes a frequency label and F the number of possible frequencies synthesized by the DFS. We exploit the

Grad-CAM localization capability to approximate the segmentation s.

Grad-CAM based trace segmentation The Grad-CAM method consists of a sequence of steps, from the CNN's classification output to the trace segmentation (see the bottom of Figure 4.2). First, the CNN's classification output z is examined before the final Softmax layer. This output z is a vector containing scores for each of the F possible frequencies within the trace. Each element z_f reflects the network's confidence that the input trace corresponds to a specific frequency.

Next, a localization map *Grad-CAM* is generated for every frequency. These maps indicate the importance of each sample within the trace for predicting each frequency. The localization maps are constructed by combining the feature maps, denoted by A, extracted from the final residual block of the CNN (see layer A in Figure 4.2).

Finally, the localization maps segment the trace by assigning frequencies to individual samples. The segmentation is achieved by selecting the frequency with the highest localization score for each sample after applying a normalization step using the Softmax function. The output s indicates the frequency assigned to each sample i in the trace. Equation 4.1 denotes the segmentation s_i for each sample i.

$$s_i = \arg\max_f \text{Grad-CAM}_i^f \tag{4.1}$$

4.2. Hardware framework for dynamic voltage and frequency randomization

Previous Section 4.1 demonstrates that commercially available DFS implementations can be resynchronized using deep learning techniques, restoring side-channel vulnerabilities and enabling classical SCA techniques, thus motivating novel practical defenses. While recent literature indicates that the success rate of side-channel attacks decreases as desynchronization increases [BPS+20], there remains a lack of clarity regarding the optimal hyperparameters and configurations for maximizing side-channel resistance in practice.

This section presents a hardware framework for AMD Xilinx FPGAs that enables fast, fine-grained, run-time variation of operating points. The framework mimics process and board-to-board variability, providing an experimental platform to study how FPGA manufacturing differences and portability affect timing, power consumption, and side-channel leakage. By providing a flexible, configurable repository of operating points and low-latency actuation, the framework enables systematic exploration of which randomization

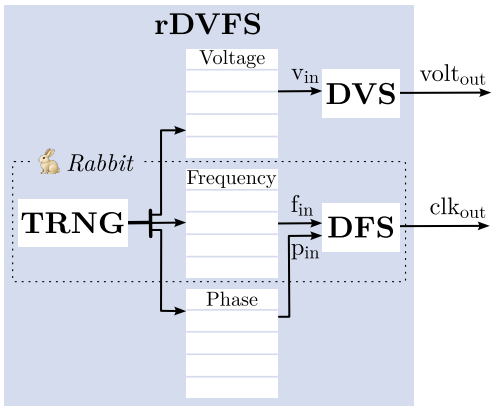


Figure 4.3: Architectural view of the proposed random dynamic voltage and frequency scaling.

strategies are most effective and offers a low-cost, deployable hiding countermeasure for real FPGA platforms.

Within this framework we introduce Rabbit, a lightweight hiding countermeasure based on randomized frequency scaling. Common DFS implementations for side-channel protection [HDL+20, DHL+21] typically offer a broad range of operating frequencies to maximize desynchronization effects. However, existing solutions continue to face challenges, including performance degradation, resource overhead, and incomplete protection against sophisticated deep learning attacks. Rabbit design pursues three goals: *i)* increase temporal uncertainty through fast, fine-grained frequency changes; *ii)* preserve uninterrupted system operation during actuations; and *iii)* offer a simple, flexible implementation on AMD Xilinx FPGA platforms with minimal PPA impact.

4.2.1. Hardware architecture

Figure 4.3 depicts the high-level view of the proposed random dynamic voltage and frequency scaling (rDVFS) architecture. The module implements a free-running true random number generator (TRNG) [GGFZ22] to address memories containing the configurations for each parameter, i.e., operating voltage, clock frequency, and clock phase. The operating points are then fed to the *DFS* and *DVS* actuators, which dynamically change the operating frequency and voltage of the FPGA, respectively. Finally, *rDVFS* outputs the generated clock (clk_{out}) and the operating voltage ($volt_{out}$) signals.

For clarity, Rabbit is a subsystem of the entire module which includes the DFS actuator, the memory containing the synthesizable frequencies, and the input TRNG.

The three configuration memories are independently accessed and are configured at de-

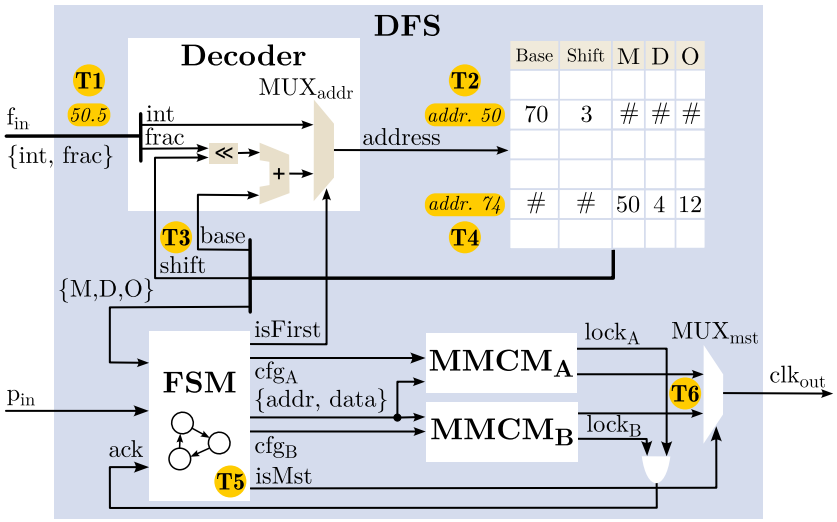


Figure 4.4: Dynamic frequency scaling module block diagram.

sign time according to the feasibility requirements documented for the target FPGA family [Xil20]. A design-time script allows users to specify boundaries and step increments for frequency and operating voltage, automatically generating the RTL description for the correctly instantiated memories. Although the architecture is generic, we allow up to 1024 frequency configurations, 8 clock phase configurations, and 128 operating voltage values. The limited number of phase configurations arises from the practical challenges of achieving rapid phase shifts for large phase values using the AMD Xilinx MMCM [AT]. To this end, we obtained large clock phase shifts by serializing multiple small phase shift actuations.

The rest of this section details the architecture of the *DFS* module, which represent the on-chip component of the random DVFS actuator. Current FPGAs do not offer on-chip DVS actuators, and thus, we exploit the DVS of the reference board leveraging the SPI/I2C protocol.

Dynamic Frequency Scaling actuator

The proposed DFS actuator leverages the ability of AMD Xilinx MMCMs to be dynamically reconfigured via Dynamic Reconfiguration Port [AT]. The relationship between the input (f_{in}) and output (clk_{out}) frequency of an MMCM is described by Equation 2.3, where M is the clock feedback multiplier, D and O are clock dividers. During reconfiguration, MMCM's scaling parameters (M, D, and O) are overwritten, delivering a new frequency.

Figure 4.4 provides an overview of the architecture of the *DFS* actuator. The module

Algorithm 4.2 DFS finite state machine algorithm.

1: $MMCM_{mst} \leftarrow MMCM_A$
2: $MMCM_{slv} \leftarrow MMCM_B$
3: **procedure** RECONFIGURE(f_{in}, p_{in}, ack)
4: $conf_f \leftarrow$ `retriveConf`(f_{in})
5: $conf_p \leftarrow p_{in}$
6: `loadConf`($conf_f$, $conf_p$)
7: `doReconf`($MMCM_{slv}$)
8: **while** $ack \neq 1$ **do**
9: $wait$
10: $MMCM_{mst}, MMCM_{slv} \leftarrow$ `flipMMCM`()

takes as inputs the clock frequency (f_{in}) and the clock phase (p_{in}), to independently enable the actuation of the two operating points. The *DFS* module outputs the generated clock signal (f_{out}). The architecture of *DFS* is organized in three parts: *i)* the storage element ($BRAM$) contains the configuration parameters for the frequency scaling (M, D, O), *ii)* the pair of Mixed-Mode Clock Modules, i.e., $MMCM_A$ and $MMCM_B$, represents the actual generators of the clock signal (f_{out}), and *iii)* the finite-state machine (FSM) and the related address decoder implement the protocol to perform the frequency and phase scaling.

Notably, AMD Xilinx MMCM allows synthesizing the required clock signal at run-time while its output remains low during the entire reconfiguration phase, thus provoking a clock-gating effect on the computing platform. The proposed architecture leverages two MMCMs to avoid the clock-gating effect during the reconfiguration of the clock signal. In particular, the master MMCM keeps generating the current clock signal while the slave MMCM is reconfigured. At the end of the reconfiguration, the role of the two MMCMs is flipped, i.e., the master MMCM becomes the slave one and vice versa, to offer a truly dynamic frequency scaling actuator.

The FSM module implements the protocol to reconfigure the clock frequency dynamically (see Algorithm 4.2). Algorithm 4.2 takes as input the required clock frequency (f_{in}) and clock phase (p_{in}) of the random operating point, and the signal that monitors the locked status of the two MMCMs (ack). The ack signal monitors the locking status ($lock_i$) of the two MMCMs to prevent triggering a new reconfiguration while the previous one is still in progress. Therefore, a new reconfiguration can start only if the ack signal is asserted, i.e., the previous reconfiguration has ended.

The algorithm 4.2 implements the configuration of the registers in the memory-mapped MMCM interfaces to dynamically change the clock frequency and phase. Notably, the

phase shift can be actuated by directly writing p_{in} into the correct MMCM memory-mapped register. In contrast, reconfiguring the clock frequency requires writing multiple MMCM registers for which the corresponding values are obtained from $BRAM$ (see line 4 in Algorithm 4.2). After retrieving the MMCM register values from $BRAM$, the procedure sequentially loads each configuration in the appropriate register (see line 6 in Algorithm 4.2). Once all register values have been updated, the actual MMCM reconfiguration can start (see line 7 in Algorithm 4.2), during which the FSM enters a wait state, waiting for the *ack* signal to rise (see lines 8-9 in Algorithm 4.2). At the end of the reconfiguration, the role of the two MMCMs is flipped, i.e., the master MMCM becomes the slave one and vice versa (see line 10 in Algorithm 4.2). The clk_{out} clock output is then selected through the $isMst$ 1-bit flag as the one output by the current master.

Such a protocol is meant to implement a flexible design that allows storing frequency configurations regardless of the step and the minimum and maximum frequencies. Considering the limitations of the AMD Xilinx MMCMs and the design timing constraints, the $BRAM$ can store up to 1024 clock frequency configurations within the $5-800\,\text{MHz}$ range where the minimum step is 0.125 MHz. For example, the user can configure the $BRAM$ with 80 frequency configurations where the slower and faster frequencies are 10 MHz and 30 MHz, respectively, with a step of 0.250 MHz.

The content of $BRAM$ is configured at design time according to the feasibility requirements described in the technical specification documents of the considered hardware components [Xil20]. In particular, the design-time configuration script allows the user to provide the boundaries and the step increment for the frequencies. It automatically generates the RTL description to instantiate the correctly configured memories, i.e., the BRAMs and the logic to access their content.

To further improve the understanding of the proposed solution, the following discusses the time steps to reconfigure the frequency of 50.5 MHz (see $T1$-$T6$ in Figure 4.4). Starting from the input frequency value $f_{in} = 50.5\,\text{MHz}$ (see $T1$ in Figure 4.4), the integer part is used to perform the first access to $BRAM$. This first memory access retrieves from line 50 a record containing the *Base* and *Shift* fields (see $T2$ in Figure 4.4). The *Base* value indicates the starting address of the actual configuration for each frequency range in the $BRAM$. *Shift* value mirrors the granularity of the frequency steps: it determines how many fractional frequencies are available between two integer frequencies. In the proposed example, the granularity is set to 0.125 MHz, allowing 2^3 fractional frequencies between 50 MHz and 51 MHz. Thus, the *Shift* value is set to 3. The retrieved *Base* and *Shift* fields, together with the fractional part of the input frequency, are used to compute the address for the second access to $BRAM$ containing the actual clock frequency configuration for f_{in}.

In particular, the configuration to obtain an operating frequency equal to 50.5 is stored at line 74, i.e., 70 (*Base*) plus 0.5 (*fractional part*) $\ll$ 3 (*Shift*) of $BRAM$ (see $T3$ and $T4$ in Figure 4.4). Notably, the two memory accesses create a level of indirection between the required frequency and the actual line of $BRAM$ that stores the configuration, thus allowing maximum flexibility in the design of the range and granularity of the frequency configurations within $BRAM$. Indeed, each frequency can have its own granularity. The configuration values M, D, and O at line 74, i.e., 50, 4, and 12, respectively, are driven to the FSM, which is in charge of configuring the slave MMCM registers through the $\{addr, data\}$ port (see $T5$ in Figure 4.4). Once the slave MMCM sets its $lock_i$ signal, the FSM drives $isMst$ to flip the role of the MMCMs and to propagate the new frequency through clk_{out} (see $T6$ in Figure 4.4).

5 | Setup and Benchmark for Methodology Evaluation

A requirement for the effortless validation and evaluation of side-channel analysis techniques lies in the availability of a controlled yet flexible experimental setup. Security laboratories rely on evaluation boards specificically desing to accurately collect side-channel signals. However, researchers cannot easily change such hardware to implement or test countermeasures. To this end, this chapter introduces a novel open framework named JARVIS to validate the robustness and efficacy of the proposed methodologies within realistic scenarios. JARVIS is a modern FPGA-based hardware-software framework that enables high-precision acquisitions while maintaining the flexibility to instantiate various countermeasures. Furthermore, two novel datasets, Chameleon and DFS_DESYNCH, are introduced to benchmark the proposed techniques under realistic hiding conditions. Both datasets are publicly released to encourage community reproduction of our results and, above all, to foster the development of new methodologies.

The rest of this chapter is organized as follows. Section 5.1 outlines the evaluation metrics adopted throughout the thesis. Section 5.2 introduces JARVIS, the hardware-software framework used for all experiments. Section 5.4 describes the DFS_DESYNCH dataset, which provides highly desynchronized power traces for evaluating attack strategies against DFS mechanisms. Finally, Section 5.3 details the Chameleon dataset, designed to benchmark segmentation and attack techniques under various hiding countermeasures.

5.1. Evaluation metrics

Evaluating the performance of a side-channel framework is a complex task that requires different metrics depending on the workflow's stage. This section describes the metrics used in this thesis to evaluate the performance of the different methodologies.

5.1.1. Segmentation metrics

Given a side-channel trace, segmentation is the task of dividing it into meaningful segments by classifying each feature, i.e., the time sample, into different classes. In this work, segmentation is used in two complementary ways. Hound leverages segmentation to locate and isolate individual COs in a side-channel trace, labeling each sample as belonging to a CO or not. Owl exploits segmentation to classify individual time samples by their frequency. The following metrics are used to evaluate the performance of the segmentation task.

Intersection over union A standard metric to evaluate the quality of a segmentation is the *intersection over union* (IoU). It measures the overlap between the predicted and the ground truth segmentation. It ranges from 0 to 1, where 1 is a perfect overlap. It is formally defined by Equation 5.1 as the ratio of the intersection between the predicted (P) and the ground truth (GT) classification and their union.

$$IoU(P, GT) = \frac{|P \cap GT|}{|P \cup GT|} \tag{5.1}$$

Notably, IoU is a meaningful metric only if the whole segmentation is available. In the context of COs identification, this means that both the start and end samples may be identified. Consequently, the IoU is applicable solely to Hound-v3, which is the only Hound's version that provides a complete segmentation.

False detection rates During COs identification, if their execution is considered as a single segment, *false positive* and *false negative rates* are two valuable metrics. A false positive occurs when a CO is located even though it is not present in the given trace. A false negative occurs when a CO is not located even though it is present in the given trace. False positive rate (FPR) and false negative rate (FNR) are defined as the proportion of false positives (false negatives, respectively) relative to the actual number of COs.

5.1.2. Deep-learning training metrics

Training a deep-learning model involves optimizing its parameters to minimize a loss function. The training process is typically monitored using a confusion matrix to evaluate the model's performance on a validation and test set.

Confusion matrix The confusion matrix (CM) is a table that summarizes the performance of a classification algorithm. The column indices represent the ground truth, i.e., the true target class, while the row indices represent the predicted ones. Thus, the cell CM_{ij} contains the percentage of traces whose true class is c_j, which the network predicted as c_i.

5.1.3. Side-channel attack metrics

A side-channel attack exploits information leaked during the execution of a CO to recover secret keys. To assess the attack effectiveness, different metrics are used to quantify its strength, such as: the guessing entropy, the guessing distance, the number of CO traces, and the CNN success rate.

Guessing entropy The guessing entropy (GE) is defined as the average rank position of the correct key among all possible key guesses, namely, the number of guesses required to find the correct key. It is formally defined by Equation 5.2, where $P_{\mathcal{T}}(k)$ is the probability of key k when attacking on set trace $\mathcal{T}$. The best possible GE value is 1, meaning the key is always precisely predicted.

$$GE(\mathcal{T}) = \mathbb{E}_{\mathcal{K}} \left[|k \in \mathcal{K} : P_T(k) \geq P_T(k^*)| \right]. \tag{5.2}$$

Guessing distance The guessing distance (GD) is a metric that quantifies the effectiveness of an attack by measuring how well the attack can isolate the correct key from the others. It is formally defined by Equation 5.3 as the distance between the correct key and the most likely incorrect key. $P_{\mathcal{T}}(k)$ is the probability of key k when attacking on set trace $\mathcal{T}$. It is normalized by the distance between the most likely and least likely keys and averaged over all possible keys. The higher the $GD(\cdot)$ value, the more effective the attack, and negative values indicate that the correct key is not the first guess.

$$GD(\mathcal{T}) = \mathbb{E}_{\mathcal{K}} \left[\frac{P_T(k^*) - \max P_T(\mathcal{K} \backslash k^*)}{\max P_T(\mathcal{K}) - \min P_T(\mathcal{K})} \right] \tag{5.3}$$

Number of CO traces The metric *number of CO traces* expresses the number of CO required to reach a guessing entropy of 1. The fewer CO executions are used to exfiltrate the secret key, the more powerful an attack methodology is considered. For CPA targeting multiple key bytes, the *number of CO traces* considers the COs needed to reach a GE of 1 for all attacked bytes. For CNN-based attacks, the *number of CO traces* is related to the best CNN model, i.e., the one that requires the fewest COs to recover the key.

CNN success rate The robustness of a countermeasure is inversely related to the number of CNNs that successfully recover the key. The more CNNs are able to achieve key recovery, the less resistant the countermeasure is considered to be. The metric *CNN success rate* refers to the percentage of tested CNNs that reached a guessing entropy of 1, highlighting how easily different neural network configurations can exploit the leakage. Notably, a *CNN success rate* greater than 0 is enough to recover the key.

5.1.4. Hardware design metrics

The implementation of a hardware design is evaluated using three main metrics: area, performance, and power consumption. This thesis focuses on FPGA implementations, and such metrics are measured using the AMD Xilinx Vivado Design Suite.

Area The area is a metric that quantifies the hardware resources used by a design. The area for FPGA designs is measured in terms of used look up tables (LUTs), flip-flops (FF), Digital Signal Processor (DSP), Block RAM (BRAM), MMCM, and Global Clock Buffer (BUFG).

Performance Performance defined the maximum clock frequency at which the design can operate. For evaluating the DFS actuator's performance, timing overhead required to reconfigure the actuator is considered. Since during this time the computing platform is gated, the performance is reduced and minimizing such overhead is crucial. Performance is defined in Equation 5.4 as the ratio between 1 and the timing overhead.

$$\text{Performance} = \frac{1}{1 + \text{Timing Overhead}} \tag{5.4}$$

Power consumption Power consumption is computed leveraging the AMD Xilinx Vivado Power Analysis tool.

5.2. JARVIS: FPGA-based hardware-software framework for side-channel research

This section introduces JARVIS, the hardware-software framework used throughout this thesis for evaluating SCA and countermeasures. JARVIS is an open-source platform that provides a comprehensive environment for SCA research on an FPGA-based IoT-class computing platform, specifically targeting modern RISC-V-based systems.

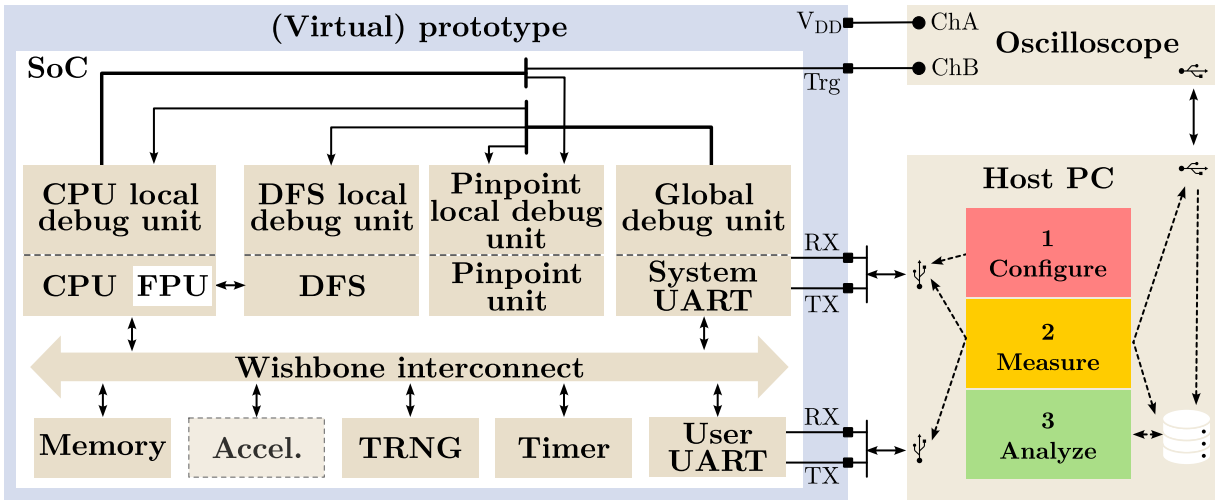


Figure 5.1: JARVIS' hardware-software infrastructure architecture.

The framework provides an IoT-class System-on-Chip (SoC) that includes a CPU based on the RISC-V ISA, dedicated hardware to enable the implementation of state-of-the-art SCA countermeasures, an ad-hoc debug infrastructure to maximize the observability and controllability of the computing platform and thus simplify the execution of SCA, and support for the open source FreeRTOS real-time operating system (RTOS).

The JARVIS framework is designed to address the lack of integrated, open-source solutions for side-channel research on general-purpose IoT platforms. The few RISC-V security-oriented solutions, e.g., OpenTitan [Low24], are not meant as general-purpose computing platforms and do not support the development of novel attack and defense methodologies, while others focus their research effort solely on the hardware side while disregarding the software framework [DCEM18] or vice versa.

JARVIS serves as the experimental foundation for the results and analyses presented in this thesis, offering a robust and extensible platform for advancing SCA research on IoT-class devices.

The rest of this Section is organized as follows: Section 5.2.1 provides an overview of the JARVIS framework, discussing its main components and their interactions; Section 5.2.2 details the microarchitecture of the SoC; finally Section 5.2.3 describes the implementation of the SoC on an FPGA prototype board and the software stack that runs on top of it.

5.2.1. Framework overview

The hardware-software infrastructure depicted in Figure 5.1 realizes the JARVIS flow by means of three main components, namely *1)* a (virtual) prototype, *2)* an oscilloscope, and

3) a host PC. This part discusses in detail the three components and how they interact with each other to deliver the JARVIS framework.

(Virtual) prototype

The computing platform at the foundation of the proposed SCA research framework implements an SoC architecture that can notably be instantiated as a prototype on FPGA, to collect the power trace measurements from an digital oscilloscope, as well as emulated as a *virtual* prototype in a RTL simulator that instantiates its post-place-and-route netlist in a testbench, to generate the matching VCD switching activity. The correspondence between the prototype and its virtual counterpart, in particular in how they execute a target application, is enforced by dedicated hardware mechanisms and it is the crucial aspect that enables employing our framework for SCA analysis.

The SoC architecture is built around a Wishbone interconnect. It comprises a CPU, and a global debug unit as its masters and a memory, a user UART, a true random number generator (TRNG), and a timer as its bus slaves. Moreover, it includes a pinpoint unit and a DFS actuator which takes its design by Rabbit [GMZ25] presented in Section 4.2.

The single-core, in-order, five-stage pipelined, 32-bit RISC-V CPU implements the base integer (I) and integer multiplication and division (M) RISC-V 32-bit ISA extensions [SZ19]. Supporting only I and M extensions, i.e., the bare minimum extensions for an IoT-class CPU, provides the simplest and most observable setup, making it easier to carry out the side-channel analysis. A floating-point unit (FPU) [ZGF21, ZG22, DGC+24] can be optionally instantiated as a functional unit of the CPU to expand the computational capabilities of the SoC, enabling the execution of floating-point-intensive applications without affecting the overall side-channel resistance of the computing platform. The interconnect consists of two separate 64-bit Wishbone data and instruction buses, both supporting single read and write transactions as well as burst ones, according to a modified Harvard architecture to minimize contention. The pinpoint unit is a hardware component that enables precise synchronization between the oscilloscope acquisition and the the execution of a target application recording the clock cycles elapsed from the start of the such application. The DFS actuator enables changing at run time the frequency of the clock signal fed to the CPU, whereas the rest of the SoC operates at a fixed clock frequency. A global debug unit and local debug units connected to the former in a point-to-point fashion compose the debug infrastructure of the SoC. The global debug unit interacts with the host PC through the system UART while two different channels support its communication with the rest of the SoC. Memory location of the memory-mapped bus slaves are accessed through Wishbone reads and writes, while dedicated lines connect the

global debug unit to the three local ones that manage the CPU, the DFS actuator, and the pinpoint unit, respectively.

The main memory, making use of the block RAM (BRAM) resources available on AMD Xilinx FPGAs, is instantiated as a slave on the Wishbone data bus. The bus slaves include a TRNG, that produces a sequence of random bits meant to be used in cryptography and SCA countermeasure tasks, a timer, which enables support for FreeRTOS, and a user UART, that provides an I/O interface for the application. Hardware accelerators exposing a Wishbone interface can also notably be added as memory-mapped bus slaves to the SoC to extend its capabilities.

The implementation phase of the proposed flow includes the selection of which parts of the SoC to instantiate, e.g., the optional the DFS actuator, pinpoint unit, TRNG, and timer, and the configuration of its parametric features, e.g., the baud rate of the UART interface, the width of the instruction and data buses, and the branch prediction scheme of the CPU.

In addition to the UART I/O interfaces, the prototype exposes the FPGA voltage ($\texttt{V}_{\texttt{DD}}$ in Figure 5.1), which is related to the power consumption of the whole FPGA chip, and a trigger signal (`Trg`), that is driven by the local debug unit of the CPU.

Oscilloscope

Power measurements from the prototype execution are carried out through an oscilloscope with two analog channels and a frequency bandwidth that is sufficient to collect samples from the target platform under measurement without aliasing. The oscilloscope is connected to the prototype board, with an analog channel measuring the voltage of the FPGA and the other one monitoring a signal meant to trigger the data acquisition on the former. It is managed through a USB interface by the host PC, which takes care of its configuration and receives the data samples measured from the board on both analog channels.

Host PC

The host PC drives the whole JARVIS framework through the flow software scripts, which manage the interaction with both hardware devices, namely the prototype board and the oscilloscope, and software tools, such as the EDA, compilation toolchains and state-of-the-art SCA scripts. We can identify three main top-level scripts that drive the corresponding phases of the framework.

Configure The configuration script leverages an EDA toolchain for the synthesis and place-and-route and bitstream generation of the SoC, previously configured according to the needs of the user, also taking as its inputs the XDC constraint files that include information about the internal clock signals as well the I/O mapping on the target FPGA chip. A RISC-V compiler toolchain is employed instead to compile the applications and, optionally, the RTOS. The compilation process produces an executable file, that can be loaded into the memory of the computing platform to be executed both in its simulation and on the prototype FPGA. Boundaries for a time window of interest are obtained from the executable to enable accurate matching, also from a temporal standpoint, the simulation of the computing platform with its prototype execution.

Measure The simulation leverages an RTL simulator to simulate the post-place-and-route netlist of the SoC inside a SystemVerilog testbench provided as part of the framework. The testbench loads the VMEM for the target application into the memory of the target SoC and drives the execution of the application according to the time window boundaries extracted from its ELF file at the previous phase. The simulation outputs a value change dump (VCD) that contains all the switching activity of the internal signals of the SoC corresponding to the execution of the application within the time window of interest. The prototype execution requires, after flashing the bitstream to the FPGA mounted on the prototype board, feeding the application binary to the SoC to be loaded into its memory. The flow script properly drives the board execution through the SoC debug interface, matching the behavior of the SystemVerilog testbench, and manages the oscilloscope connected to the board to collect the power trace of the execution. The VCD from the simulation and the power trace from the prototype execution are saved to a data storage device, so that they can be later retrieved for performing the various analyses.

Analyze The VCD obtained from the simulation and the power traces collected from the prototype execution, both corresponding to the exact same time window, are retrieved from the storage where they were saved to analyze them through a set of state-of-the-art SCA techniques, including CPA [CCD00], template [CRR03], and DL-based attacks [MPP16, BPS+20, ZBHV20], to detect whether there was any cryptographic information leakage and also identify the potential sources of the latter. In addition to the SCA security statistics, the analysis phase outputs the traditional PPA metrics leveraging the power consumption measured from the board execution, the cycle-granularity latency collected from simulation, and the resource utilization reports from netlist implementation. The outputs of the computation can also be obtained through the debug infrastructure to further check the correct functioning of the system.

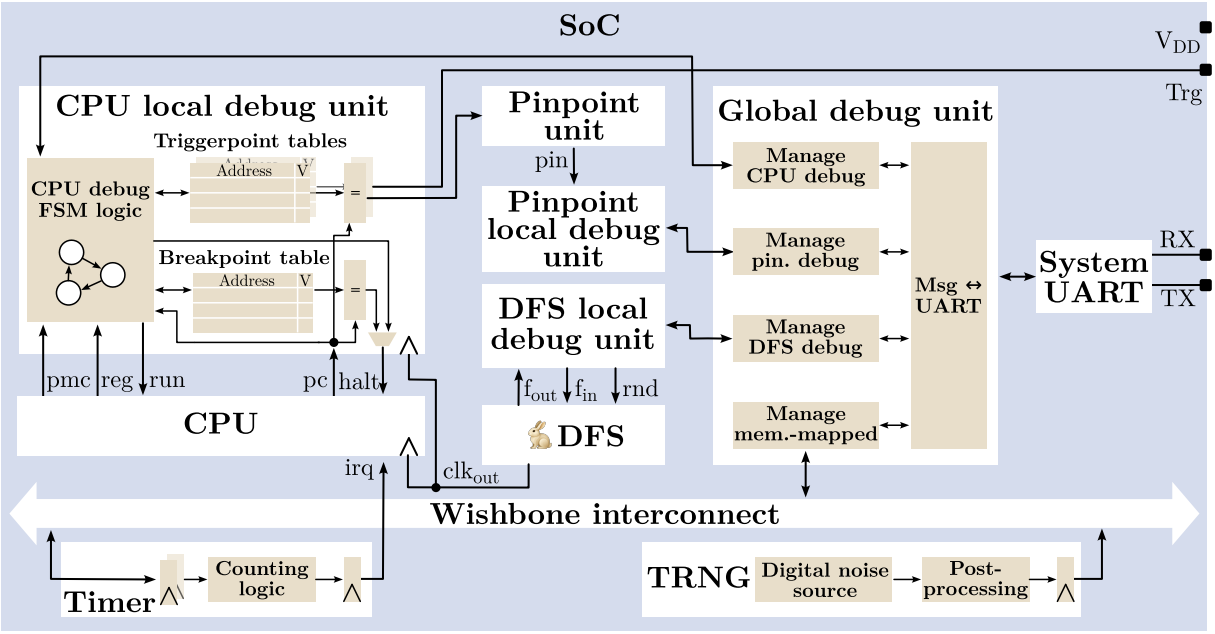


Figure 5.2: Detailed microarchitecture of the debug subsystem, timer, and TRNG.

5.2.2. SoC microarchitecture

Dedicated hardware and software mechanisms enable the side-channel analysis and countermeasure capabilities of the proposed framework. The debug subsystem, with its breakpointing and ad-hoc triggering mechanism, ensures maximum observability and controllability of the computing platform, thus allowing the collection of accurate side-channel information that can then be exploited by SCA of different kinds. A TRNG, a pinpoint unit, a DFS actuator, and a timer can instead be optionally instantiated in the SoC to enable support for a variety of state-of-the-art SCA countermeasures ranging from masking to hiding ones. The rest of this section delivers a detailed discussion of the microarchitecture of the debug subsystem, TRNG, pinpoint unit, DFS actuator, and timer.

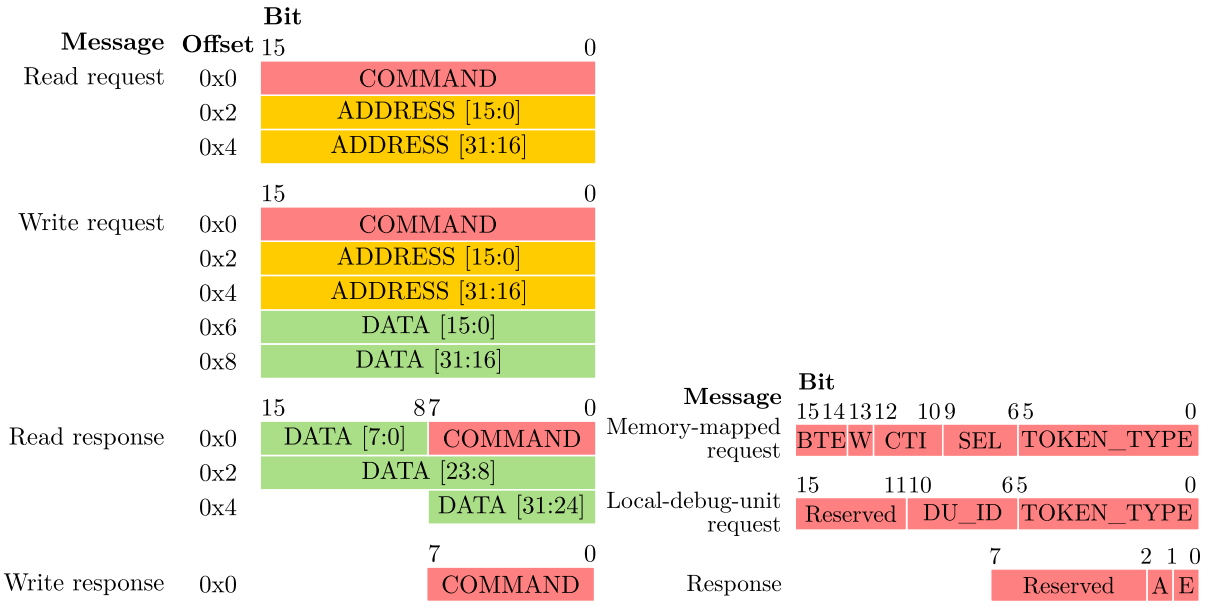
Debug subsystem

The framework provides hardware support for breakpoints and triggerpoints as well as a dedicated debug infrastructure that exposes a GDB-like debug interface, thus enabling the collection of accurate side-channel information that can then be targeted by attacks of different kinds. This part discusses first the global-local debug microarchitecture, then focuses on the breakpointing and triggering mechanisms, and finally provides an overview of the debug capabilities that can be exploited by the external host PC.

Global and local debug units The debug subsystem enables controlling and observing the whole SoC through a message-based protocol. It is composed of a global debug unit and of a number of local debug units. The global one, that acts as a master on the Wishbone bus, receives debug messages through a system UART interface and accordingly communicates both with the other bus masters, through point-to-point connections to local debug units that act as adapters to interface with such masters, and with the bus slaves, with whom it interacts directly through the bus by exploiting their memory-mapped nature.

Two local debug units are instantiated for the CPU core, for the pinpoint unit, and for the DFS respectively, to act as adapters between the global debug unit and the two bus masters. The local debug unit interface with the global one is common to all the local debug units, while the interface with the CPU, the pinpoint unit, or the DFS module is custom tailored to the specific interactions that are implemented with such module. The CPU local debug unit can access the program counter (PC), registers, and performance monitoring counters (PMCs) of the CPU, as well as halt, reset, and restart it and advance its execution by a single step. The pinpoint local debug unit stores the clock cycles elapsed between the start and the end of a target application during an oscilloscope's acquisition and then exposes allow the host PC to read such value and reset it to 0. The DFS one can set a new target clock frequency for the DFS actuator, get the frequency of the current clock signal generated and configure the DFS to randomly switch the frequency of the clock signal. Moreover, it can store the current clock frequency together with the elapsed clock cycles during an oscilloscope's acquisition and exposed such information to the host PC. Figure 5.2 depicts the microarchitecture of the debug subsystem of the SoC, highlighting the global debug unit, the CPU and DFS local debug units, and their interactions with each other as well as with the CPU and the DFS actuator.

Breakpoints and triggerpoints The CPU local debug unit supports, through its interaction with the CPU, a traditional breakpointing system coupled with an ad-hoc triggering one. Breakpoints enable halting the CPU when the PC matches their corresponding addresses, while triggerpoints, an ad-hoc variant of breakpoints, are meant to toggle a trigger signal that drives the data acquisition from the oscilloscope and do not instead halt the CPU, which is kept regularly running. The local debug unit for the CPU contains one table for the breakpoint and two tables for the triggerpoint addresses, respectively, as shown in Figure 5.2. All tables include a configurable number of entries, each composed of a 32-bit address that corresponds to an instruction in the target application and a 1-bit flag that signals its validity.



(a) Debug messages (b) Command field

Figure 5.3: Debug messages supported by the JARVIS framework: (a) structure and width of the debug messages, (b) encoding of the command field. Legend: **BTE** burst type extension, **W** write enable (0: read, 1: write), **CTI** cycle type identifier (for burst mode), **SEL** select, **TOKEN_TYPE** request message type, **Reserved** reserved for future use, **DU_ID** identifier for target local debug unit, **A** ack, **E** error; **BTE**, **W**, **CTI**, and **SEL** refer to Wishbone.

The valid addresses in the breakpoint table are constantly checked against the current PC of the CPU, whose value is passed to the CPU local debug unit. Whenever there is a match between a valid breakpoint address and the current PC, the CPU is halted until it is resumed through a dedicate debug command. The triggerpoint table entries are similarly compared against the PC, but a match between the latter and a triggerpoint address produces a notably different effect. Rather than halting the CPU as in a traditional breakpoint fashion, reaching a triggerpoint toggles a 1-bit signal that is mapped either on an I/O pin of the prototype board to be used as a trigger signal for an oscilloscope or on an pinpoint unit input.

Debug messages The debug infrastructure exposes a request-response protocol to interface with the SoC through the system UART. The communication is indeed carried out as a sequence of request and response messages. Notably, no new request message can be issued until the previous request has been completed and the corresponding response message has been sent back. Figure 5.3 summarizes the width, structure, and information encoded in the various messages depending on whether they are request or response and

read or write and on whether they are intended for memory-mapped recipients or local debug units.

Request messages can be of four types, depending on whether they correspond to read or write actions and whether their recipients are memory-mapped devices or bus masters. All request messages are composed of a 16-bit command and a 32-bit address, while write requests also comprise a 32-bit data, as shown in Figure 5.3a. The command field encodes different information depending on whether the debug message targets a memory-mapped peripheral or a local debug unit, as depicted in Figure 5.3b. Debug request messages that can be sent to the SoC through the system UART include, as listed in Table 5.1, *i)* memory-mapped reads and writes, *ii)* register reads and writes, *iii)* commands to reset, resume, and halt the CPU, advance its execution by a single step, and get its current state and program counter, *iv)* commands to retrieve performance monitoring counters *v)* commands to set, get, and remove both the breakpoints and the triggerpoints, and *vi)* commands to set and get the target clock frequency for the DFS actuator and configure it to randomly switch clock frequencies.

Response messages can be instead of only two types, i.e., read or write. The former include data as part of the response, while the latter only acknowledge the completion of the requested operation or the occurrence of an error.

TRNG

The TRNG is a Wishbone slave peripheral that exposes a set of 32 random bits through a memory-mapped register. The TRNG can be configured in the architecture of the digital noise sources and post-processing methods that compose it to obtain different results in terms of FPGA resource utilization, throughput, and security. A digital noise source produces the actual entropy underlying the random number generation, while a post-processing method improves the statistical and security properties of the TRNG. Three digital noise sources and three post-processing methods from the literature are provided with the proposed framework to implement the TRNG component. The digital noise sources that can be instantiated in the TRNG module are NLFIRO, PLL-based, and edge-sampling ones, and they can be coupled with XOR, Von Neumann, and LFSR post-processing methods [GGFZ22]. The random output produced by the TRNG component is periodically refreshed and exposed by a memory-mapped register that can be read through Wishbone. The randomness has been evaluated using the NIST SP 800-22 test suite [RSN+01] with all tests passing successfully.

Table 5.1: Debug message types supported by the JARVIS framework.

TOKEN_TYPE	R/W	Destination	Group
INVALID	Read	Memory-mapped	Invalid
MMAP_READ MMAP_WRITE	Read Write	Memory-mapped	Memory-mapped
GPR_INT32_READ GPR_INT32_WRITE GPR_FPU32_READ GPR_FPU32_WRITE	Read Write Read Write	CPU local debug unit	CPU registers
HALT_CPU RUN_CPU RST_CPU GET_DULOCAL_STATE GET_CPU_PC	Read Read Read Read Read	CPU local debug unit	CPU reset/ resume/halt
ADVANCE_ONE_STEP ECHO_FRONTEND	Read Write	CPU local debug unit	CPU stepping
GET_LOW_CYCLECNT GET_HIGH_CYCLECNT GET_LOW_INSTRCNT GET_HIGH_INSTRCNT	Read Read Read Read	CPU local debug unit	Performance counters
SET_BRKPNT_CPU GET_BRKPNT_CPU RM_BRKPNT_CPU GET_NUM_BRKPNT_CPU	Write Read Write Read	CPU local debug unit	Breakpoints configuration
SET_TRGPNT_CPU GET_TRGPNT_CPU RM_TRGPNT_CPU GET_NUM_TRGPNT_CPU	Write Read Write Read	CPU local debug unit	Triggerpoints configuration
SET_FREQ_DFS GET_FREQ_DFS RND_FREQ_DFS	Write Read Write	DFS local debug unit	DFS configuration
GET_PIN RST_PINS	Read Write	DFS/Pinpoint local debug unit	Dataset

Algorithm 5.1 Pinpoint Recording

```
    Inputs: trigger_gpio,
            trigger_pin
1:  sample ← 0
2:  while trigger_gpio do
3:      sample ← sample + 1
4:      if trigger_pin commutates then
5:          store(sample)
6:          sample ← 0
```

Algorithm 5.2 Frequency Recording

```
    Input: trigger_gpio
1:  sample ← 0
2:  while trigger_gpio do
3:      sample ← sample + 1
4:      if new frequency is stable then
5:          f ← getFrequencyDFS()
6:          store({sample, f})
7:          sample ← 0
```

Pinpoint unit

The pinpoint unit is an ad-hoc hardware module that enables precise synchronization between the oscilloscope's acquisition and the execution of a target application by recording the clock cycles elapsed from the start of the computation to its end. The module takes as input two trigger signals from the CPU local debug unit, namely `trigger_gpio` and `trigger_pin`. Following a host request, the pinpoint unit local debug unit returns the recorded `data`.

Algorithm 5.1 explains the behavior of the pinpoint unit to record the measured workload's first and last sample. When the computation is initiated, the CPU asserts the `trigger_gpio` signal. The `trigger_gpio` signal starts the acquisition process on the oscilloscope and simultaneously activates the pinpoint unit. As the process progresses, the `sample` variable is incremented with each clock cycle until the execution of interest application, at which point the computing platform sends a `trigger_pin` signal to the pinpoint unit. This signal marks the beginning of the monitored code section, storing the `sample` value. Once the execution completes, the `trigger_pin` signal is de-asserted, marking the end of the process. Thereby, the pinpoint unit records the first and last samples of the monitored application, ensuring that each *pin* is accurately aligned with the corresponding run. The pinpoint unit operates based on a loop that iterates once per clock cycle, with a constant clock frequency. Similarly, the oscilloscope's sampling frequency is maintained at a constant rate. This constancy ensures that the values recorded by the pinpointing unit are consistently correlated with the power trace samples captured by the oscilloscope, with any differences being a simple multiplication factor. Once the computation is complete, the host PC retrieves all the data collected by the oscilloscope and the pinpoint unit, ensuring that every side-channel trace is accurately aligned with the corresponding target application.

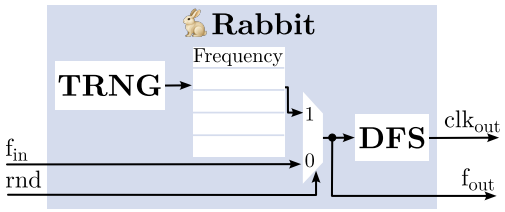


Figure 5.4: Detailed microarchitecture of the DFS actuator.

Rabbit for JARVIS

The DFS design is based on the Rabbit microarchitecture [GMZ25] presented in Section 4.2, tuned to the JARVIS framework for interfacing with the local debug unit. The DFS actuator leverages two mixed-mode clock manager (MMCM) components to provide a glitch- and latency-free switching between different clock frequencies at run time. Figure 5.4 depicted Rabbit's architecture, which receives as its inputs a target clock frequency $\texttt{f}_{\texttt{in}}$ and a 1-bit flag to enable random DFS `rnd`, and it outputs the generated clock signal $\texttt{clk}_{\texttt{out}}$ and the current clock frequency$\texttt{f}_{\texttt{out}}$. The `rnd` flag selects the actual target clock frequency for the DFS reconfiguration between the one received through the $\texttt{f}_{\texttt{in}}$ input by the DFS local debug unit and a random value $\texttt{f}_{\texttt{rnd}}$ output by an internal TRNG with similar to the one in Section 5.2.2. Such target clock frequency, registered to be output to the DFS local debug unit and readable through a dedicated debug command, is decoded and used as an address to select a corresponding set of MMCM configuration parameters from the `BRAM` memory. Random reconfiguration is activated through a debug request (`RND_FREQ_DFS` in Table 5.1), which is dispatched to the DFS local debug unit that in turn asserts the `rnd` input flag. When `rnd` is set to 1, the MMCMs keep indeed reconfiguring to new clock frequencies as soon as the previous reconfiguration has been completed, i.e., the `ack` signal to the FSM is asserted.

As additional metadata, the DFS local debug unit provides the frequency value for each sample of a side-channel trace during an oscilloscope's acquisition. Algorithm 5.2 describes the behavior of the frequency recording unit. The `sample` variable is incremented at each clock cycle until the `trigger` signal stops the recording process. Notably, the clock cycle applied to the frequency monitoring module is kept fixed, i.e., it is not affected by the frequency changes. When the DFS actuator synthesizes a new frequency and becomes stable, the unit reads the current frequency from the DFS actuator and stores it along with the `sample` value. The `sample` variable is cleared. Notably, a set of frequency resynchronizers is used to allow the coexistence of two operating frequencies, i.e., one dynamic for the CPU and the other fixed for the monitoring infrastructure.

Table 5.2: Breakdown of the SoC's FPGA resource utilization.

	FPGA resource utilization			
Component	**LUT**	**FF**	**DSP**	**BRAM**
CPU	4386	3192	4	0
DFS	822	225	0	1.5
Global debug unit	286	248	0	0
CPU local debug unit	875	587	0	0
DFS local debug unit	19	29	0	0
Memory	206	138	0	64
TRNG	2486	1252	0	0.5
Timer	144	163	0	0
System UART	365	292	0	0
User UART	359	284	0	0
Overall SoC	10667	7563	4	66

Timer and FreeRTOS support

The timer is a memory-mapped peripheral that exposes two 64-bit registers which can be accessed through read and write requests on the Wishbone interconnect. The two registers correspond to the current time, which gets increased by 1 at each clock cycle, and to the time threshold. Once the current time is greater than the time threshold, an interrupt request to the CPU is raised by setting to 1 the content of a 1-bit register.

Instantiating a timer in the SoC notably enables executing the FreeRTOS real-time operating system and thus providing support for implementing coarse-grained multithreading.

5.2.3. Deployment

The JARVIS framework has been deployed on the NewAE Technology CW305 [New18] Artix FPGA Target board, specifically designed for power analysis and fault injection attacks against hardware cryptographic functions. The CW305 board mounts a BGA socket that can accommodate any chip from the AMD Xilinx Artix-7 mid-range FPGA family, which is the most widely adopted in academic and industrial research, including the security and cryptography fields [Nat22].

Power measurements are carried out through an oscilloscope from the Pico Technology PicoScope 5000 Series family [Pic21], which features a 200MHz maximum bandwidth and can collect up to 1 billion samples per second at a resolution ranging from 8 to 16 bits. The oscilloscope has its A analog channel connected to the board through a coaxial SMA

connector to measure the FPGA voltage, while its B analog channel is connected to a general-purpose I/O pin that outputs the trigger signal which marks the start and the end of the computation.

The host PC must be able to execute the AMD Xilinx Vivado toolchain for the synthesis, place-and-route, and bitstream generation targeting AMD Xilinx Artix-7 FPGAs, as well as for the simulation and switching activity VCD collection. The software framework is run in Ubuntu 22.04.3 LTS on a host PC that features an Intel i7-10700 CPU and a 64GB DDR4 memory. The host PC includes ChipWhisperer 5.1.0 software and PicoScope 7 software and drivers to manage the CW305 board and the oscilloscope, respectively. Applications are compiled with GCC 11.4.0 to be executed on the SoC either bare-metal or on top of FreeRTOS 11.0.1. AMD Xilinx Vivado ML 2023.1 is employed for the RTL synthesis, place-and-route, bitstream generation, and simulation.

The SoC to be prototyped on FPGA and simulated is configured with a CPU that implements the RV32IM instruction set, a 256kB main memory, and a TRNG, a pinpoint unit, a DFS actuator, and a timer that enable support for state-of-the-art SCA countermeasures. Table 5.2 lists the FPGA resource utilization resulting from synthesis and place-and-route targeting 100 MHz and 50MHz clock frequencies for the CPU, driven by the DFS actuator, and the rest of the SoC.

5.3. Chameleon: dataset for segmenting and attacking obfuscated power traces

Chameleon is a novel dataset that aims to complement the available open datasets by offering real-world, obfuscated power traces to the broader research community. While available datasets focus only on the attack phase, Chameleon has also been specifically designed to develop segmentation techniques. State-of-the-art datasets [BPS+20] usually employ mathematical models for desynchronization via data augmentation. Even if data augmentation is a viable technique to face the lack of data and improve model robustness, the mathematical function used to augment data must be proved to properly model the physical phenomena inducing the desynchronization. However, Chameleon's side-channel traces leverage physical obfuscation processes, providing more realistic scenarios where the timing variations change unpredictably throughout the trace. Additionally, the obfuscation mechanisms generate genuine noise, particularly during transitions between different operating conditions, such as frequency shifts or delays, further amplifying this noise. The dataset is available at `https://github.com/hardware-fab/chameleon`.

The acquisition setup and the labeling methodology are discussed in Section 5.3.1. Section 5.3.3 details the dataset structure, and Section 5.3.4 provides statistics and Signal-to-Noise Ratio on the dataset. Finally, Section 5.3.5 presents the results of the dataset leakage validation through successful attacks.

5.3.1. Data acquisition

JARVIS [ZGG25] represents the hardware-software infrastructure employed to acquire the dataset. The setup comprises a host PC, a Picoscope 5244d digital sampling oscilloscope (DSO) [Pic21], and a NewAE Technology CW305 [New18] board which hosts an AMD Xilinx Artix-7 FPGA [Xil20]. The RISC-V CPU has been modified to implement the analyzed hiding methods. In particular, we provide a hardware implementation for the random delay and DFS actuators, while morphing and chaffing techniques are software implemented. Within each measured side-channel trace, we perform multiple executions of the COs. For each execution of the CO, the computing platform takes as input the `plaintext` and the secret `key` from the host and sends back the computed `ciphertext`, which is checked for functional correctness by the host. To avoid polluting the measured side-channel signal with external noise, the plaintexts and the secret key are pre-loaded into the computing platform's memory before acquiring each side-channel trace. During this phase, the host PC also configures the DSO to ensure it is ready to capture the power traces corresponding to the upcoming cryptographic operations.

The synchronization between the computing platform, the DSO, and pinpoint unit is meticulously designed to ensure precise timing across these components, which is critical for accurate data collection. At the beginning of the acquisition of a new side-channel trace, a hardware `trigger` from the computing platform signals to start the acquisition of the power trace to the DSO. In parallel, the pinpoint unit is responsible for recording the instants when the computing platform begins and ends the execution of each cryptographic operation. To synchronize the DSO acquisition with the pinpoint unit, the same trigger signal is fed to both the DSO and the pinpoint unit. The detailed description of the pinpoint unit is provided in Section 5.2.2.

As the cryptographic operation of choice, we selected the unprotected AES-128 cryptographic primitive from the OpenSSL [Tcl23] codebase. Notably, there are several other implementations of the AES algorithm. However, the wide adoption of the OpenSSL library in real applications allowed us to deliver a realistic dataset. The sampling rate of the DSO is set to 125 Msample/s with a resolution of 12 bits for the entire dataset. Notably, the CPU is clocked at 50 MHz for all acquisition campaigns, but for the DFS

one for which the DFS actuator is instructed to randomly change the operating frequency of the computing platform as its maximum speed.

As additional metadata, the Chameleon dataset provides the frequency value for each sample of each side-channel trace. Using the DFS as a hiding technique, we designed a module that allows continuous and random changes in the CPU frequency during the computation. Then, we developed an additional hardware module to record the time of each frequency change and the actual frequency value. Algorithm 5.2 describes the behavior of the frequency recording unit.

The complete hardware infrastructure has been added to the SoC, and a set of frequency resynchronizers is used to allow the coexistence of two operating frequencies, i.e., one dynamic for the CPU and the other fixed for the monitoring infrastructure.

5.3.2. Implemented hiding methods

The Chameleon dataset provides power traces obfuscated by four different hiding countermeasures, i.e., dynamic frequency scaling (DFS), random delay (RD), morphing (MRP), and chaffing (CHF). Notably, the decision to implement each hiding technique in hardware or software is guided by the traditional computer architecture design approach with the final goal of creating a dataset reproducing realistic scenarios.

Dynamic frequency scaling A randomized DFS continuously varies, in a random fashion, the clock signal frequency fed to the CPU, producing a distortion in the power consumption trace and reducing the information leakage [GMZ25]. The DFS mechanism has been hardware implemented in the SoC following the Rabbit microarchitecture [GMZ25] presented in Section 4.2 and deployed on JARVIS SoC. We leverage a hardware true random number generator (TRNG) [GGFZ22] to provide a random number to the DFS actuator continuously. The random number selects a frequency from a pool of 760 available frequencies ranging from 5 MHz to 100 MHz, with a step of 125 kHz, thus enabling a large variability. Notably, DFS remained enabled throughout the entire acquisition phase. Consequently, a new frequency is requested as soon as a configuration is locked. As a result, each cryptographic primitive experiences a variable number of clock frequency reconfigurations, leading to significant trace deformation.

Random delay The random delay countermeasure introduces random delays in the execution of the cryptosystem to hinder the attacker's ability to correlate the power consumption with the executed instructions [CK09]. The random delay mechanism has been hardware implemented in the SoC. The CPU has been modified to implement the random

Table 5.3: XOR equivalent implementations for morphing countermeasure [ABPS15b].

$a \oplus b$
$(\sim a \wedge b) \vee (a \wedge \sim b)$
$(a \wedge b) \oplus (a \vee b)$
$(a \wedge b) \wedge (\sim a \vee \sim b)$
$(a \vee b) \wedge \sim (a \wedge b)$
$\sim \big((\sim a \vee b) \wedge (a \vee \sim b)\big)$
$\sim \big((\sim a \wedge \sim b) \vee (a \wedge b)\big)$
$(a \wedge \sim b) \oplus (\sim a \wedge b)$

delay mechanism at hardware level by leveraging a TRNG [GGFZ22] in the computing platform. At run-time, the TRNG keeps generating random numbers to determine the actual number (up to 2) of random instructions to be inserted between each pair of consecutive program instructions.

Morphing The morphing countermeasure leverages the TRNG [GGFZ22] to randomly modify the executed instructions of a cryptographic operation [ABPS15b]. The morphing mechanism has been software implemented. Morphing targets the AddRoundKey and SubBytes steps of the AES cryptosystem. Each AddRoundKey execution uses a randomly selected version of the XOR operation, chosen among the eight equivalent implementations from Table 5.3 with different power consumption profiles. SubBytes executions are instead morphed by employing two *S-Boxes* that are masked by random values and periodically refreshed.

Chaffing Chaffing countermeasure introduces a set of chaff keys, i.e., secret keys that are different but correlated with the actual secret key used to encrypt the plaintext, to confuse the attacker [ABPS15a]. Algorithm 5.3 describes the chaffing pseudocode. The chaff mechanism spawns multiple threads executing the same CO. Only one CO employs the correct secret key (see line 1 of Algorithm 5.3), while the others use the chaff keys (see line 3-5). The chaffing hiding mechanism is software implemented and leverages the TRNG [GGFZ22] embedded in the computing platform to support a randomized multi-threading scheduler at the operating system level. In particular, we implemented the randomized thread scheduler within the FreeRTOS real-time operating system [Fre24]. All the threads are started simultaneously (see line 6), until the thread performing the actual CO has completed (see lines 7-12), and their execution is interleaved according to

Algorithm 5.3 Chaffing countermeasure algorithm [ABPS15a].

Inputs: AES, key, ptx, numChaff

```
1: mainThread ← Create (AES, key, ptx)
2: chaffKey ← GenerateChaffKeys (key, numChaff)
3: for i in 1 to numChaff do
4:     chaffThreads[i] ← Create (AES, chaffKey[i], ptx)
5: StartRandomScheduler (mainThread, chaffThreads)
6: while IsRunning (mainThread) do
7:     Wait ()
8: for i in 1 to numChaff do
9:     Kill (chaffThreads[i])
```

a random scheduler. The Chameleon dataset contains side-channel traces using a chaffing implementation employing three threads, i.e., using two chaff secret keys, to constrain acquisition time.

5.3.3. Dataset structure

The Chameleon dataset is organized into five subdatasets, i.e., `BASE`, `DFS`, `RD`, `MRP`, `CHF`, one for each hiding countermeasure implemented. `BASE` collects unprotected side-channel traces, while the other subdatasets are protected with a different hiding technique.

Figure 5.5 depicts the organization of the files in the dataset. The structure is the same for all the subdatasets, except for the `frequencies` folder, which is available only for the `DFS` part of the dataset. The dataset is organized in two folders: metadata and data.

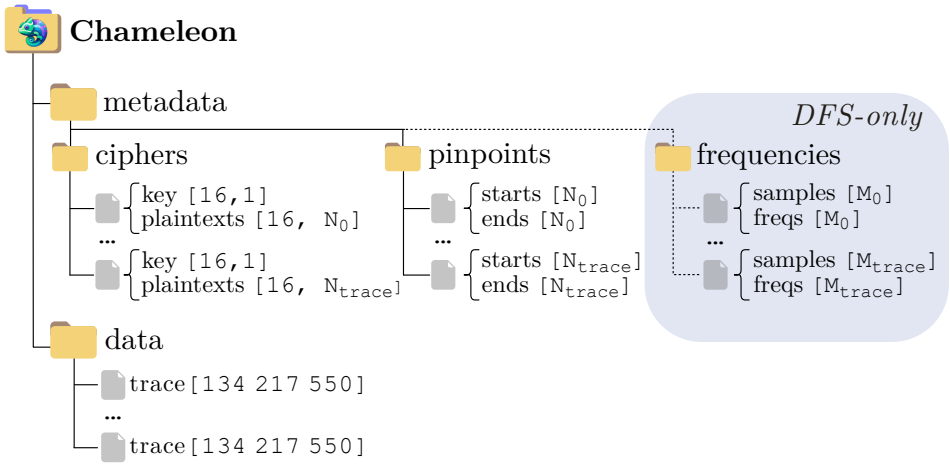


Figure 5.5: Chameleon dataset's file system.

Metadata For each hiding technique, the folder contains the information related to each execution of the COs within each trace. The metadata is divided into three sub-folders:

- `Ciphers` contains the plaintext and key of the AES executions in each trace file. The key is fixed for all the AES executions in the same trace, while the i-th plaintext is the output of the $(i-1)$-th encryptions to assure a random exploration of the plaintext space. The exact number of AES executions, thus plaintexts, in each trace i is different due to the random nature of the hiding mechanisms. We explored 256 different keys, namely all the possible variations of the first key byte keeping the last 15 bytes fixed, and the number of encryptions is balanced among the keys.
- `Pinpoints` contains the start and end time samples of each AES execution in each trace file. The value ranges from 0 to the trace length, i.e., 134 217 550 samples, and it is expressed in time samples. If an AES execution spans over the end of the trace, the end sample is set to the trace length, thus the last encryptions may be incomplete.
- `Frequencies`, only available for the `DFS` part of the dataset, contains the frequency changes due to the DFS actuations in each trace file. Each trace undergoes a different number of frequency changes due to the random nature of the DFS actuation, and the actual number of changes is given by M_i. The metadata comprises two fields: *i) sample* marks the time sample when a frequency change occurs and ranges from 0 to the trace length; *ii) frequency* specifies the new operating frequency starting from the corresponding sample and takes a value from 5 MHz to 100 MHz.

Data The folder contains N_{trace} raw measurements dumped directly from the oscilloscope. Each trace consists of 134 217 550 time samples equivalent to the storage capacity of Picoscope 5244d DSO [Pic21] operating at 125 MSample/s using a resolution of 12 bits. The side-channel trace is obtained from the SoC execution of the COs interleaved with general-purpose applications [CFG+20], i.e., an application different from a cryptographic one. The exact number of AES executions in each trace i is different due to the random nature of the hiding mechanisms, and it is given by parameter N_i in the metadata folder.

5.3.4. Statistics

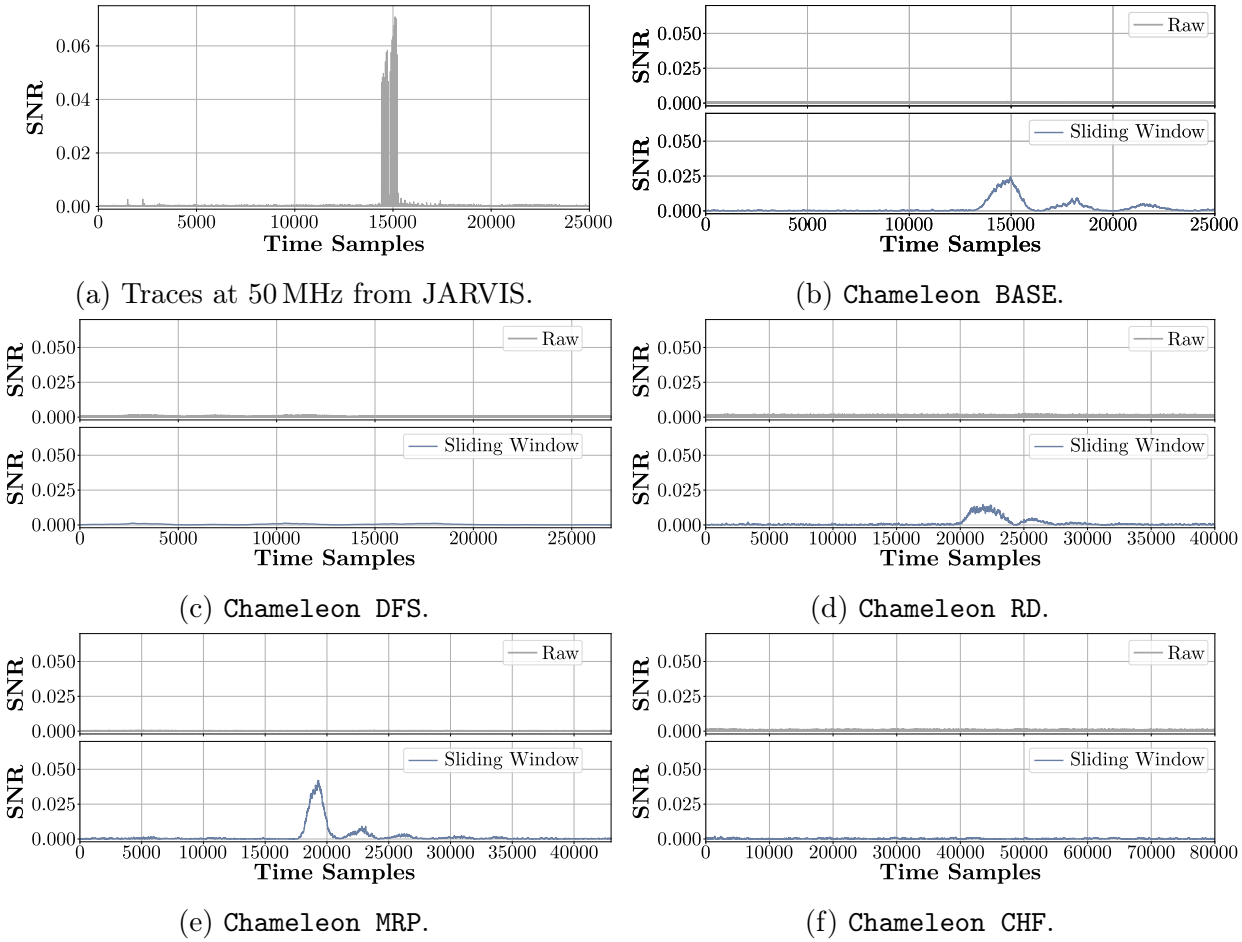
Table 5.4 reports the statistics of the Chameleon dataset for each hiding method. The acquisition time is the total time required to collect the data for each hiding method. The `DFS` subset required the longest acquisition time of 43 hours and 20 minutes due to the labeling of the frequency changes. The number of data is the total number of traces

Table 5.4: Chameleon dataset's statistics for each hiding countermeasure implemented. Execution lengths and frequency durations are expressed in time samples.

		BASE	DFS	RD	MRP	CHF
Acquisition time		3h 31m	43h 20m	4h 17m	4h 15m	5h 28m
# Traces		256	256	512	512	1 024
# AES (per trace)	mean	821	898	536	575	262
	std	6.89	7.84	7.47	3.87	2.53
AES executions length	mean	148 192	135 621	220 120	195 599	398 274
	std	0.00	13 420.55	5 061.38	4 434.78	88 546.58
General purpose applications length	mean	15 276	13 695	30 096	37 789	112 878
	std	36 578.06	33 403.41	76 200.47	36 529.91	32 683.79
# Frequencies (per trace)	mean	1	31 000	1	1	1
	std	0	74.36	0	0	0
Frequencies duration	mean	134 217 550	4 330	134 217 550	134 217 550	134 217 550
	std	0	10.37	0	0	0

collected and available in the dataset. Since we balanced the number of AES executions among the hiding methods, the number of traces, i.e., data, is different for each subset with a minimum of 256 traces for the `DFS` subset and a maximum of 1 024 traces for the `CHF` subset. The AES executions length is related to the implemented hiding mechanisms, with a mean ranging from 135 621 to 398 274 samples. The standard deviation is higher for the `CHF` subset due to the variable multithreading execution. The length of general-purpose applications is also reported, with mean and standard deviation. The executed general-purpose applications are randomly chosen from the same pool for every hiding mechanism except for chaffing. In `CHF` subset, the general-purpose applications consider the FreeRTOS routines of the scheduler and context switch.

Notably, each acquisition has been performed with a fixed CPU frequency of 50 MHz, except for the `DFS` subset. Therefore, the number of frequency changes is always one for each trace and the duration is the trace length, i.e., 134 217 550 time samples, except for the `DFS` subset. The DFS actuator performs a frequency change every 4 330 samples on average, with a standard deviation of 10.37 samples. Thus, each DFS trace has a different number of frequency changes, with a mean of 31 000 changes.



(a) Traces at 50 MHz from JARVIS.

(b) `Chameleon BASE`.

(c) `Chameleon DFS`.

(d) `Chameleon RD`.

(e) `Chameleon MRP`.

(f) `Chameleon CHF`.

Figure 5.6: SNRs and SW-SNRs related to the intermediate `sbox(p[1] ⊕ k[1])` in the first AES round interval on Chameleon's raw traces and traces aligned with sliding window.

Signal-to-Noise Ratio

A Signal-to-Noise Ratio (SNR) has been computed to quantify the noise level in the measurements [Man04]. The analysis has been performed on raw traces and traces aligned using a sliding window of 100 samples [CCD00], aiming to mitigate the impact of hiding countermeasures. Since key byte 0 can vary across traces in Chameleon, the SNR is calculated for the intermediate `sbox(p[1] ⊕ k[1])` to identify the leakage samples associated with the *S-Box* computation of byte 1 during the first encryption round. Each SNR is computed with a total of 200 000 COs randomly sampled from Chameleon and correctly segmented using the metadata available in the dataset. As a reference, the SNR for unprotected AES traces, i.e., with a static frequency of 50 MHz and no countermeasures applied, and aligned leveraging JARVIS' triggering infrastructure [ZGG25], is reported in Figure 5.8a and referenced as JARVIS's SNR. Figure 5.5 plots each SNR in the first AES

round interval. In each plot, the upper panel presents the SNR for raw traces, while the lower panel displays the SNR following sliding window alignment (SW-SNR). Since the length of each AES is random and different for each subdataset, we plotted the first fifth of the AES execution to ensure it always includes the first round's *S-Box* execution.

Comparing the JARVIS's SNR and the SNR on the raw `BASE` subdataset (see Figures 5.8a-5.6b), we can observe that the first-order leakages are significantly reduced in the Chameleon traces even without any countermeasure. The SNR reduction is likely due to Chameleon's long, single DSO acquisitions, capturing multiple consecutive COs. This prolonged capture increases vulnerability to environmental noise, such as clock drift, thermal fluctuations, and low-frequency phase noise, which accumulates over time. In contrast, CW305's short, triggered acquisitions limit noise exposure, leading to a higher SNR. For the subdatasets that apply countermeasures (see upper panels in Figures 5.6c-5.6f for `DFS`, `RD`, `MRP`, and `CHF`), the SNR values on raw data are similarly low, without any peak. Each technique introduces variability and noise into the traces, effectively obfuscating the leakage and making the power traces harder to analyze.

After applying the sliding window alignment (see lower panels in Figure 5.6), the SW-SNR values increase significantly for `BASE`, `RD`, and `MRP` subdatasets, reaching a peak value of 0.42 for `MRP`, thus mitigating environmental noise and lighter hiding countermeasures. However, there are no significant differences between raw SNR and SW-SNR for `DFS` and `CHF` subdatasets.

Notably, the SNR analysis aims to quantify the noise and leakage obfuscation in side-channel traces. For a thorough comparison of hiding techniques and dataset validation, please refer to next Section 5.3.5, where the effectiveness of side-channel attacks confirms the Chameleon dataset's integrity.

5.3.5. Leakage validation

To assess the quality of the Chameleon dataset, we aligned and successfully attacked the collected COs, grouped by hiding method. The attack phase was performed in isolation from the segmentation phase to evaluate a comprehensive dataset validation and assess the correct acquisition of side-channel information. Specifically, we conduct a CPA attack [BCO04] with a sliding window alignment [CCD00] (SW-CPA) and a deep-learning-based attack employing the tool introduced in [RWPP21] to generate high-performance CNN architectures automatically. The [RWPP21] methodology uses reinforcement learning to sequentially select CNN layers and hyperparameters guided by a reward function optimized for SCA performance. This reward function evaluates the network based on its

guessing entropy performance, including how quickly the guessing entropy converges to 1 and considering network validation accuracy. The reward function prioritizes models that can extract the secret key using fewer traces, optimizing efficiency and attack success. The algorithm is customized to focus on neural architectures commonly effective in SCA, with the search space consisting of convolutional, average pooling, batch normalization, or global average pooling layers. Each CNN ends with a softmax layer preceded by flattened or fully connected layers using the SeLU activation function.

Configurations

All the attacks exploit the leakage due to the *S-Box* substitution on the first AES round. Since we want to assess the dataset's quality, we leverage the Chameleon metadata to perform the COs' localization step. Each encryption has been truncated to include the first AES round and to fix the input size for the attack. Specifically, DFS, random delay, and morphing traces have been truncated after 100 000 samples, while chaffing traces after 200 000 to take into consideration chaffing higher length variability.

The SW-CPA attack uses a sliding window of 100 samples, and it is computed with a total of 200 000 COs randomly sampled from Chameleon. The attack is performed on the last 15 bytes of the AES key, as the first byte is not fixed.

In the CNN-based attacks, we preprocess the raw power traces with a highpass filter with a cutoff frequency of 125 KHz to remove any low-frequency noise captured during the acquisition. Moreover, every trace underwent sub-sampling by averaging 100 consecutive samples to reduce the input size and perform faster training without compromising the side-channel information [PWP22]. We target the first byte during the profiling and attack phase. As in standard deep-learning attacks, we split the dataset in training, validation, and test sets with a ratio of 80%, 10%, and 10%, respectively. The test set is used to evaluate the performance of the attack on the aligned traces. Key values are balanced from the set of possible keys for training and testing, and the COs are shuffled to avoid bias in the model learning process. We train 500 models for each hiding method on an Intel Core i5-12400 with 64 GB of RAM and an NVIDIA GeForce GTX 1080Ti GPU. We set the number of epochs to 50, the batch size to 256, and a one-cycle learning rate scheduler applied to the Adam optimizer.

Results

Table 5.5 shows the attack's results on the aligned AES executions across the five different Chameleon subdatasets. For the SW-CPA attack, results are reported in terms of the

Table 5.5: Results of the SW-CPA and CNN-based attacks on the aligned AES executions.

		BASE	DFS	RD	MRP	CHF
SW-CPA	**GE**	1	121.76	1	1	118.20
	# CO traces	$4.7k$	–	$37.5k$	$14k$	–
CNN-based	**GE**	1	1	1	1	1
	# CO traces	14	79	100	28	104
	CNN success rate (%)	48	0.7	1.2	3.0	0.19

guessing entropy (GE) and the number of COs required to recover the secret key. The results demonstrate that the `BASE` subdataset, which lacks any countermeasures, allows efficient key recovery with a GE of 1 achieved after only $4.7k$ COs. The `MRP` and `RD` subdatasets also allow key recovery, requiring $14k$ and $37.5k$ COs, respectively, indicating that these countermeasures increase resistance to SW-CPA but do not entirely prevent it. In contrast, for the `DFS` and `CHF` subdatasets, the attack does not succeed within the attacked dataset size, highlighting their effectiveness in obfuscating side-channel leakage.

For the CNN-based attack, the performance of the attack is evaluated by averaging 256 independent attacks, i.e., one for each key byte value. Along with GE and number of CO traces, we introduce a third metric, the percentage of effective CNN models trained, to a more robust comparison between countermeasures. The GE is 1 across all tested scenarios, but the number of CO traces needed to achieve it varies significantly between subdatasets. Notably, the unprotected `BASE` scenario requires only 14 COs to recover the key, demonstrating its vulnerability. When the chaffing countermeasure was applied, the number of CO traces rose sharply to 104, making it appear the most resilient. However, we stopped the CNN generator after training 500 models for each scenario, leaving potentially better-performing models unexplored. For this reason, the number of COs required to recover the key should not be solely used to compare the different hiding techniques. The *CNN success rate* reveals a clearer picture of the relative robustness of the countermeasures. As seen in the last row of Table 5.5, for the `BASE` scenario, 48% of the trained CNNs were able to recover the key, confirming that unprotected AES implementations require little effort in model tuning to exploit the leakage. The effectiveness drops dramatically for subdatasets with countermeasures. Specifically, `DFS` and `CHF` show success rates of only 0.7% and 0.19%, respectively, demonstrating their ability to reduce leakage significantly. The `RD` and `MRP` subdatasets exhibit higher success rates, at 1.2% and 3.0%, respectively, reflecting the SW-CPA results.

Overall, the results confirm the high quality of the Chameleon dataset. Successful key

recovery demonstrates that the COs are correctly located in the traces as indicated by the metadata and that the computations align with the provided plaintexts and keys. Furthermore, the ability to perform both SW-CPA and CNN-based attacks validates the accuracy of the side-channel trace acquisition, ensuring the dataset can effectively support the development of new attack methodologies.

5.4. DFS_DESYNCH: dataset for evaluating side-channel attacks against dynamic frequency scaling

The `DFS_DESYNCH` dataset consists of AES-128 power traces highly desynchronized via random DFS paired with frequency encoding. By publishing this dataset, we aim to offer real-world, highly desynchronized power traces to the broader research community to facilitate the development of more effective SCAs. While available open datasets [BPS+20] employ a mathematical model for desynchronization, our traces leverage an actual DFS process. This approach provides a more realistic desynchronization scenario, where the desynchronization is not constant throughout the trace but varies with each frequency change. Additionally, the DFS mechanism introduces real noise affecting all traces, with the transition between frequencies further amplifying this noise. The dataset is available at `https://github.com/hardware-fab/DLaTA`.

The acquisition setup and the labeling methodology are discussed in Section 5.4.1. Section 5.4.2 details the dataset structure, and Section 5.4.3 provides statistics and Signal-to-Noise Ratio on the dataset.

5.4.1. Data acquisition

The data acquisition is performed leveraging the JARVIS framework [ZG22] deployed on the CW305 board [New18]. Exploiting the debug infrastructure, the computing platform pre-loads `plaintext` and `key` inputs sent by the host PC and awaits the `ciphertext`, i.e., the result of the AES encryption. The host sends a command to start the AES encryption and the DSO waits for the `trigger` signal to start the data acquisition. The `trigger` signal is asserted by the CPU local debug unit when the AES encryption starts and is used to synchronize the DSO's acquisition with the execution of the target application. At the end of the computation, the `ciphertext` from the computing platform is sent back to the host, which checks the correctness of the encryption. The DFS actuator is configured to randomly synthesize a new clock frequency between six available ones, ranging from

35 MHz to 60 MHz with a step of 5 MHz. Notably, the DFS is instructed to continuously synthesize a new frequency randomly selected between the available ones as its maximum speed.

For the cryptographic benchmark, we employed the constant-time, unprotected AES-128 implementation from the OpenSSL [Tcl23] library. While alternative AES implementations exist, OpenSSL's prevalence in real-world deployments ensures the dataset's practical relevance. All acquisitions were performed at a sampling rate of 125 Msample/s with 12-bit resolution using a Picoscope 5244d DSO [Pic21].

Each power trace records one whole AES encryption, regardless of the operating frequency. Notably, traces with an average faster frequency have the last part of the acquisition in idle, as they have already finished computation. Moreover, several hundred random operations were performed before each encryption to further desynchronize the side-channel traces. The collected power traces have been highpass filtered with a cut-off frequency of 125 kHz to remove any low-frequency signal component added by the physical DFS actuator.

Hardware digital clock labeling

As additional metadata, the dataset provides the clock frequency at each time sample of the collected traces. From the theoretical standpoint, the clock frequency variations can be back-annotated on the collected traces by calculating the length of each period of the clock signal measured by an oscilloscope. Despite its theoretical simplicity, such a task requires recording a reasonably good square wave of the clock signal, thus imposing a high-performance oscilloscope with a sampling rate at least 20 times faster than the clock frequency generated by the DFS module. To this end, we exploited the JARVIS' frequency recording unit of the DFS local debug unit, which records the clock frequency at each sample of the measured trace during the AES acquisition (see Algorithm 5.2).

5.4.2. Dataset structure

The `DFS_DESYNCH` dataset is organized into two subsets, namely `profiling` and `attack`. We collected two batches of traces on two separate days and randomly split them between the profiling and attack group. Figure 5.7 depicts the `DFS_DESYNCH` dataset architecture. Both subdatasets share the same organization as follows:

Traces Each subdataset contains 128 000 power traces, each made of 200 000 time samples recording the whole AES encryption preceded by hundreds of random instructions.

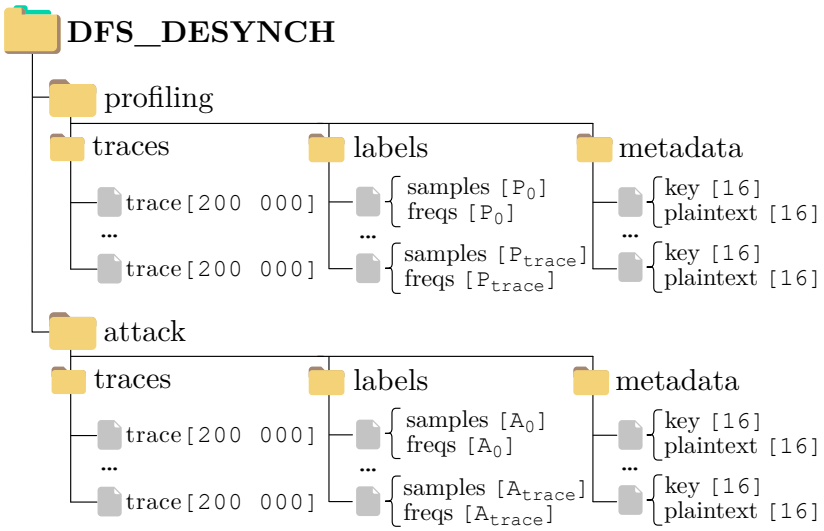


Figure 5.7: `DFS_DESYNCH` dataset's file system.

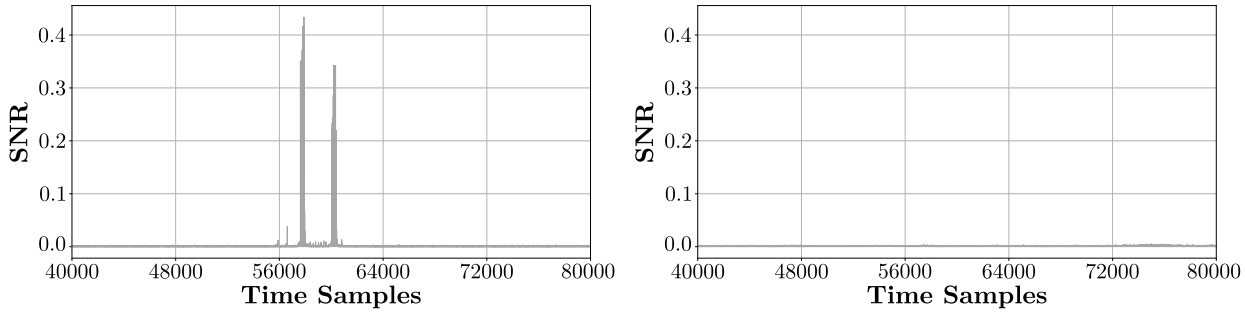
We explored 256 different keys, corresponding to all possible values of the first key byte while keeping the remaining 15 bytes fixed, and ensured that the number of encryptions is balanced across all keys. The power traces are post-processed using a highpass filter with a cut-off frequency of 125 kHz.

Labels For each power trace, the labels indicate the frequency changes occurring during the AES encryption. The label comprises two members: *i) Sample* marks the time sample when occurs a frequency change and ranges from 0 to 200 000; *ii) Frequency* specifies the new operating frequency starting from the corresponding sample, and it takes value from the set {35, 40, 45, 50, 55, 60}. The first sample is always 0, providing the starting frequency. The number of labels per trace is not fixed since a random number of frequency changes can occur during its execution.

Metadata Each subdataset contains one metadata for each trace, including *i) key* and *ii) plaintext*. Only the first key byte changes every 500 traces. Intermediate values to perform the attack can be derived using the metadata, such as `sbox(p[0]` $\oplus$ `k[0])`.

5.4.3. Statistics

The `DFS_DESYNCH` dataset comprises a total of 256 000 power traces, with 1 000 traces collected for each possible value of the first key byte. Each trace contains 200 000 time samples, capturing the entire AES-128 encryption process under dynamic frequency scaling. To support profiling and attack scenarios, the dataset is split into two equal subsets: 128 000 traces for profiling and 128 000 for attack, corresponding to 500 traces per key



(a) Traces at 45 MHz from JARVIS.

(b) `DFS_DESYNCH`.

Figure 5.8: SNRs related to the intermediate `sbox(p[0] ⊕ k[0])` in the interval [40 000, 80 000] including the first AES round.

byte in each subset. The traces exhibit a high degree of desynchronization due to the randomized frequency changes introduced by the DFS actuator. On average, each trace experiences 41 frequency changes, with a variance of 2.60. The average desynchronization, measured as the sample offset between the fastest and slowest traces, is 64 536 samples, with a standard deviation of 15 053.17. The maximum observed deviation between traces is 140 833 samples.

These statistics highlight the challenging nature of the dataset for side-channel analysis, providing a realistic benchmark for evaluating the effectiveness of desynchronization countermeasures and attack methodologies.

Signal-to-Noise Ratio

The Signal-to-Noise Ratio (SNR) [Man04] has been computed to quantify the noise introduced by the DFS actuator and to locate leakage samples related to the first S-Box computation in the first AES round, i.e., `sbox(p[0] ⊕ k[0])`. The SNR was estimated using 100 000 traces randomly sampled from the dataset. Figure 5.8 reports the SNR for unprotected traces recorded at a static 45 MHz and for traces from the `DFS_DESYNCH` dataset.

The results show that DFS substantially reduces observable leakage. The SNR for `DFS_DESYNCH` traces exhibits no prominent peak and its maximum value is about 94 times smaller than that of the static traces. In addition to high desynchronization, the reduction is due to the random timing of frequency transitions, which can occur during the `S-Box` execution and further suppress leakage. Even when the `S-Box` and nearby samples are captured at a single instantaneous frequency within a trace, that frequency may vary across traces, degrading alignment and lowering the measured SNR.

6 | Experimental Results and Comparative Evaluation

This chapter presents a comprehensive experimental evaluation of the methodological contributions presented in this thesis. All experiments were conducted using JARVIS, a hardware-software platform explicitly designed for side-channel security research, deployed on AMD Xilinx Artix-7 FPGAs. Results demonstrate that Hound can reliably segment COs in heavily obfuscated traces, achieving perfect identification rates across multiple hiding mechanisms. Owl's resynchronization pipeline enables effective attacks on frequency-desynchronized signals, recovering secret keys from traces where classical and deep-learning-based attacks fail. Finally, Rabbit achieves zero performance overhead while increasing side-channel resistance with no detectable first-order leakage.

The chapter is structured around three main methodological contributions. Section 6.1 discusses segmentation performance using the Hound framework; Section 6.2 reports Owl attack results; and finally Section 6.3 analyzes the effectiveness and efficiency of the proposed defense mechanisms.

6.1. Hound segmentation results

This section presents the results of Hound, a deep-learning framework designed to segment COs in obfuscated side-channel traces. Three versions of Hound have been developed, each improving upon the previous one. Hound-v1 [CGL+24] and Hound-v2 [GCZ24] targets specific hiding mechanisms, i.e., random delay and DFS respectively. They have been originally tested on an ad-hoc dataset, collected specifically to evaluated their segmentation capabilities on various COs. Initial testing on controlled problems highlighted the necessity of developing a more robust and general method, i.e., Hound-v3 [GCZ25]. Hound-v3 was evaluated on the Chameleon dataset [GCZ25], designed to benchmark side-channel segmentation frameworks on four common hiding mechanisms. The results of these tests are presented in this section, along with a discussion of Hound's limitations.

Table 6.1: Dataset sizes and parameters for each Hound-v1 stage over all the tested ciphers.

		AES	AES mask	Clefia	Camellia	Simon
Parameters	**Mean length**	220k	50k	108k	6k	10k
	$\boldsymbol{N}_{train}$	22k	4.8k	6k	1.4k	2k
	$\boldsymbol{N}_{inf}$	20k	5k	6k	1k	2k
	s	1k	100	500	100	100
Dataset size (N. windows)	**CO starts**	65 536	131 072	65 536	32 768	65 536
	CO spares	65 536	65 536	32 768	65 536	32 768
	Noises	32 768	65 536	32 768	32 768	32 768

6.1.1. Hound-v1 evaluation

Configuration

Data acquisition All data has been collected on the JARVIS framework sampling at 125Msamples/s with a resolution of 12 bits. The CPU has been modified to implement the random delay mechanism at hardware level by leveraging a true random number generator (TRNG) [GGFZ22]. At run-time, the TRNG keeps generating random numbers to determine the actual number of random instructions to be inserted between each pair of consecutive program instructions. The reported results consider two random delay configurations, i.e., RD-2 and RD-4. RD-2 and RD-4 limit the maximum number of inserted random instructions between two consecutive instructions in program order to 2 and 4, respectively. As the cryptographic operations (COs) of choice, we selected the constant-time, unprotected version of four ciphers, i.e., AES-128, Camellia-128, Clefia-128, and Simon-128, from the OpenSSL software codebase [Tcl23], and a masked version of Tiny-AES-128 [MEL20].

CNN training The training of the CNN leverages an NVIDIA GeForce GTX 1080Ti GPU employing the PyTorch software framework. The training datasets have been collected following the procedure of Section 3.1.1. A brief experimental campaign was carried out to find the right balance between the three window cases, i.e., *CO starts*, *CO spares*, and *noises*. Table 6.1 reports the dataset sizes as well as the windows sizes N_{train}. Windows belonging to the ciphers, i.e., *CO starts* and *CO spares*, are taken balanced between the key bytes. As in standard deep learning models, we divided the collected datasets into training, validation, and testing, respectively 80%, 15%, and 5% of the total. Each network was trained for 2 epochs using Adam [KB15] to minimize the cross-entropy loss (see

Table 6.2: Test confusion matrices for Hound-v1 on an ad-hoc random-delay dataset. Cryptographic operations of choice are AES, AES mask, Camellia, Clefia, and Simon.

(a) `AES`

Predicted class \ True class	0	1
0	**99.56%**	0.44%
2	2.70%	**97.30%**

(b) `AES mask`

Predicted class \ True class	0	1
0	**99.87%**	0.13%
2	0.07%	**99.93%**

(c) `Camellia`

Predicted class \ True class	0	1
0	**99.92%**	0.08%
2	0%	**100%**

(d) `Clefia`

Predicted class \ True class	0	1
0	**88.08%**	11.92%
2	0.03%	**99.97%**

(e) `Simon`

Predicted class \ True class	0	1
0	**94.30%**	5.70%
2	7.90%	**92.10%**

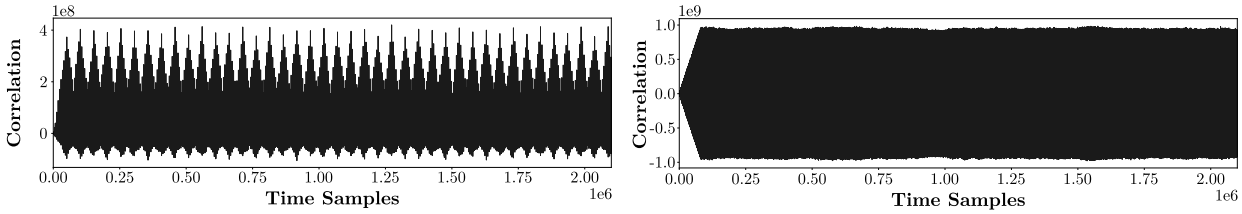
Equation 3.1). The mini-batches size was set to 64 with a learning rate of 0.001. The validation error was evaluated after each epoch, and the network with the lowest error was selected.

Segmentation parameters The pipeline parameters vary depending on the characteristics of the cipher that we want to classify, e.g., the average execution time. To this end, we experimentally determined all the required parameters. Table 6.1 shows the different values of the inference window sizes N_{inf} and the strides s, for each cipher. The use of the global average pooling layer allowed a smaller window size N_{inf} for the inference phase than the one N_{train} used for the training phase.

For each cipher, we tested the inference pipeline considering *i)* consecutive cipher executions and *ii)* encryptions interleaved with random general-purpose applications. The execution of a sequence of consecutive COs assesses the robustness of the proposed method in locating the COs when they are executed one after the other. The execution of the COs mixed with noisy applications assesses the effectiveness of the methodology in locating the COs within a heterogeneous application scenario.

Results

As an evaluation of the goodness of the trained CNNs, their confusion matrices are shown in Table 6.2, where for each cipher is reported the score on the RD-4 configuration. The



(a) Power trace without random delay, containing 40 AES encryptions.

(b) Power trace with random delay, containing 18 AES encryptions.

Figure 6.1: Output values from the matched filter [BFP22] applied to traces containing AES computations.

column indices represent the true classes, while the row indices represent the predicted ones. Notably, the trained classifiers can discriminate well between the two classes, as highlighted by the high percentages on the main diagonal of each matrix.

The segmentation false positive and negative rates are 0% for every cryptographic algorithm in both scenarios, i.e., consecutive encryption and interleaved with noisy applications, always managing to find all 512 executions without errors. The same results are achieved for both random delay configurations, i.e., RD-2 and RD-4. This demonstrates the generalizability of our approach, which, in addition to working with varying random delay configurations, also works on different encryption algorithms. We also show how our methodology suits protected ciphers, such as masked AES, whose side-channel traces have great variability.

We benchmarked our approach against two state-of-the-art methods, namely [BFP22] and [TBW$^+$21], on obfuscated AES-128 traces with RD-2 and RD-4 random delay configurations. In all tested scenarios, Hound successfully detected the start of every CO in the side-channel traces. Conversely, both reference methods were unable to locate any COs due to desynchronization caused by random delay, resulting in a 100% FNR. Notably, the authors in [TBW$^+$21] highlighted the proposed methodology's limitation to locating the COs in the presence of an effective random delay desynchronization countermeasure. In contrast, the authors in [BFP22] do not explicitly highlight such limitations for their methodology, thus, we perform a more in-depth investigation. Figure 6.1 depicts the results of the methodology in [BFP22] considering a side-channel trace containing the consecutive execution of 40 AES encryptions with (see Figure 6.1b) and without (see Figure 6.1a) random delays. As expected, the methodology in [BFP22] correctly identifies the forty AES executions in the trace without random delay, while the desynchronization due to the random delay destroys the pattern-matching information used by [BFP22] to locate the COs (see Figure 6.1b).

Table 6.3: Parameters for each Hound-v2 stage over all the tested ciphers.

	AES		AES mask		Clefia		Camellia	
General-Purpose Applications	✓	✗	✓	✗	✓	✗	✓	✗
avgCO	145k	120k	50k	50k	80k	80k	4.4k	4.1k
k	150	150	10	150	150	150	80	63
N	10k	10k	5k	5k	3k	3k	1.1k	1.1k
s	62	62	50	50	80	80	50	50
batch size	256		256		256		128	
lr	0.01		0.007		0.007		0.007	
dropout	0.2		0.35		0.3		0.4	

6.1.2. Hound-v2 evaluation

Configuration

Data acquisition The power traces used to train and evaluate Hound-v2 were collected on the JARVIS framework operating at a sampling rate of 125 Msamples/s and a resolution of 12 bits. The DFS actuator (see Section 4.2) has been configured to randomly synthesize a new frequency from a pool of 760 available ones ranging from 5 MHz to 100 MHz, with a step of 125 kHz. Notably, DFS remained enabled throughout the entire experimental phase. Consequently, a new frequency is requested as soon as a configuration is locked. As a result, each cryptographic primitive experiences a variable number of clock frequency reconfigurations, e.g., an average of 41 reconfigurations per AES encryption, leading to significant trace deformation. As the cryptographic primitives of choice, we selected the constant-time, unprotected version of three ciphers, i.e., AES-128, Clefia-128, and Camellia-128, from the OpenSSL software codebase [Tcl23], and a masked version of Tiny-AES-128 [MEL20].

CNN training The training of the CNN leverages a NVIDIA GeForce GTX 1080Ti GPU employing the PyTorch software programming framework. The training datasets were built following the methodology outlined in Section 3.1.1, starting from a collection of 262 144 power traces. The three classes, namely *CO starts*, *CO spares*, and *noises*, were balanced with equal representation of 33% components each. Table 6.3 reports the window sizes N for each cipher. As a general guideline, the window size N is adjusted to approximate one round of the targeted CO. For fast encryption algorithms, like Camellia, the window size is increased to capture a more significant portion of the CO without compromising the quality of results. The windows corresponding to the ciphers, i.e., *CO*

starts and *CO spares*, are evenly distributed across the key bytes. Following standard deep-learning practices, the collected datasets were divided into training, validation, and testing sets, constituting 80%, 10%, and 10% of the total, respectively.

Each neural network underwent training for 25 epochs, where an epoch encompasses a complete iteration over the training set. The Adam optimizer [KB15] was employed to minimize the cross-entropy loss, paired with the one-cycle learning rate scheduler. The validation error was assessed after each epoch, and the network with the lowest error was chosen for further evaluation. Subsequently, this network was employed to assess the performance of unseen traces during the inference phase. Since DFS introduces significant trace deformation with high variability, we noticed how CNNs that did not reach a validation accuracy of at least 99% were not always able to classify the COs in the sliding windows procedure correctly. Hyperparameters such as training batch size, learning rate (lr), and dropout were carefully chosen to address this issue (see last rows in Table 6.3).

Segmentation parameters A brief experimental campaign was conducted to evaluate the best pipeline parameters. Table 6.3 displays the values of the average cipher length $avgCO$, the strides s, and the initial kernel size k for each cipher.

For each cipher, we evaluated the pipeline on traces with consecutive CO executions and traces with COs interleaved with general-purpose applications, to assess robustness in both clean and noisy scenarios.

Results

A distinct CNN model has been trained using a custom dataset for each cipher under examination. As an indicator of the effectiveness of the trained CNNs, their confusion matrices are presented in Table 6.4. Notably, the trained classifiers can discriminate excellently between the three classes, as highlighted by the high percentages on the main diagonal of each matrix.

The segmentation false positive and false negative rates were consistently 0% across all tested cryptographic algorithms, both for consecutive CO executions and for traces interleaved with noisy general-purpose applications. This confirms that the proposed methodology reliably identifies all CO executions with high accuracy, maintaining robustness even under significant trace deformation caused by DFS.

We compare our approach against two state-of-the-art proposals, i.e., [BFP22], [TBW+21], targeting obfuscated AES-128 encryptions. For each analyzed scenario, Hound correctly

Table 6.4: Test confusion matrices for Hound-v2 on an ad-hoc DFS dataset. Cryptographic operations of choice are AES, AES mask, Clefia, and Camellia.

(a) `AES`

Predicted class \ True class	0	1	2
0	**100%**	0%	0%
1	0%	**100%**	0%
2	0%	0%	**100%**

(b) `AES mask`

Predicted class \ True class	0	1	2
0	**100%**	0%	0%
1	0%	**100%**	0%
2	0%	0%	**100%**

(c) `Clefia`

Predicted class \ True class	0	1	2
0	**99.97%**	0%	0.03%
1	0.01%	**99.99%**	0%
2	0%	0%	**100%**

(d) `Camellia`

Predicted class \ True class	0	1	2
0	**100%**	0%	0%
1	0%	**100%**	0%
2	0%	0%	**100%**

identifies the beginning of all the COs in the side-channel trace. In contrast, the two state-of-the-art methodologies fail to locate the COs in the side-channel trace due to the high DFS obfuscation, with a 100% FNR, indicating they cannot locate any COs in the trace.

6.1.3. Hound-v3 evaluation

Configuration

Data acquisition Hound-v3 has been evaluated on the Chameleon dataset [GCZ25], which is a benchmark dataset designed to assess the performance of side-channel segmentation frameworks. The Chameleon dataset consists of side-channel traces collected from the JARVIS framework operating at a sampling rate of 125 Msamples/s and a resolution of 12 bits. The dataset includes AES-128 power traces from five different hiding mechanisms, namely *BASE*, *DFS*, *RD*, *MRP*, and *CHF*. Please refer to Section 5.3 for a detailed description of the Chameleon dataset.

CNN training A distinct CNN model has been trained using a custom dataset for each Chameleon subdataset. The training of the CNN leverages an Intel Core i5-12400

Table 6.5: Configuration parameters for Hound-v3 training and inference phase.

	BASE	DFS	RD	MRP	CHF
Window size	10k	10k	20k	20k	30k
Learning rate	0.007	0.007	0.007	0.007	0.007
Batch size	256	128	128	256	128
Drop out	0.30	0.45	0.4	0.30	0.35
Weight decay	$3e^{-5}$	$8e^{-5}$	$5e^{-5}$	$3e^{-5}$	$5e^{-5}$
Stride	50	50	100	100	100
Screening filter	150	400	300	150	160

with 64 GB of RAM and an NVIDIA GeForce GTX 1080Ti GPU, employing the PyTorch software programming framework. The training datasets were built by leveraging the pinpoint metadata, windowing the side-channel traces into N-sample windows, and labeling them according to three classes, namely start of a CO, middle of a CO, and noise. The window sizes were adjusted to approximate one round of the targeted CO, i.e., AES-128, as reported in Table 6.5. To mitigate bias during training, an effort was made to balance the three classes. However, there were insufficient N-sample windows for the noise class in some instances. In those cases, we opted to unbalance the classes rather than significantly reduce the dataset size. To address the imbalance, class weights were adjusted during loss minimization to compensate for the potential bias. We preprocess the power traces with a highpass filter with a cutoff frequency of 125 KHz to remove any low-frequency noise captured during the acquisition and with data standardization. Only 90% of the traces were used for training, while the remaining 10% were used for testing the segmentation. The training datasets were split again into training, validation, and testing sets, constituting 80%, 10%, and 10% of the total. Each CNN network underwent training for 75 epochs, leveraging the Adam optimizer paired with the one-cycle learning rate scheduler.

A comprehensive hyperparameter search has been performed, inlcuding the learning rate, the batch size, and the regularization values, using a random search strategy [BB12]. Table 6.5 reports the chosen CNN hyperparameters used for training and the input window size.

Segmentation parameters The inference parameters, namely, the stride used in the `Sliding Window Classification` and the screening filter for polishing the segmentation, are also reported in Table 6.5.

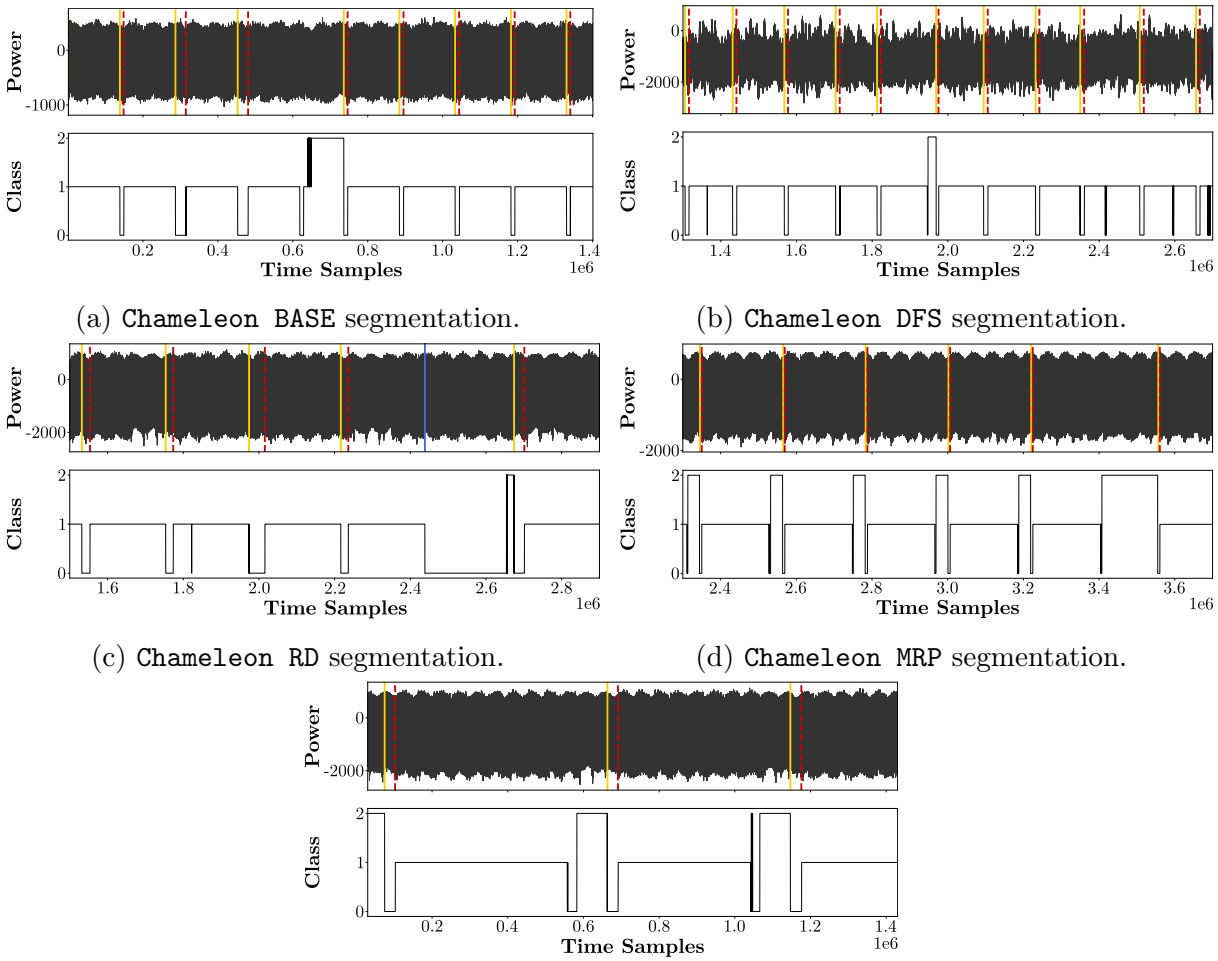
Table 6.6: Results of side-channel segmentation with Hound-v1 [CGL+24], Hound-v2 [GCZ24], and Hound-v3 [GCZ25] on the Chameleon dataset.

		BASE	DFS	RD	MRP	CHF
Hound-v1	**IoU** (%)	–	–	–	–	–
	FNR (%)	0.22	4.40	3.08	0.34	1.22
	FPR (%)	0.59	6.76	0.00	12.44	0.54
Hound-v2	**IoU** (%)	–	–	–	–	–
	FNR (%)	0.01	0.06	1.15	0.00	0.67
	FPR (%)	0.11	0.73	0.13	0.00	0.48
Hound-v3	**IoU** (%)	89.47	88.23	84.19	91.26	87.96
	FNR (%)	0.00	0.00	0.00	0.00	0.00
	FPR (%)	0.00	0.00	0.00	0.00	0.00

Results

Last rows of Table 6.6 report the segmentation results for the Chameleon dataset. The results indicate that all segmentation techniques achieved a high IoU, ranging from 84.19% to 91.26%. The morphing technique consistently outperformed others, suggesting it introduces less noise and variability into the traces. The IoU could be further improved by decreasing the stride in the `Sliding Window Classification`, albeit at the cost of increased computation time. The heuristics introduced to mitigate segmentation errors have been effective, consistently reporting 0% FNR and FPR across all techniques and demonstrating excellent performance in detecting all the present COs.

Figure 6.2 helps to better understand the segmentation results by showing the CNN models' output on the Chameleon dataset. All the plots show random portions of the traces but with the same number of samples to allow a comparison of the density and length of the AES operations for each hiding technique. For instance, Figure 6.2c shows the `RD` segmentation struggling with class 2, i.e., the noise class, often misclassifying it as class 0, i.e., the start of a CO. This misclassification can be noticeable by looking at the classifications starting with sample 2.41e6, which are classified as class 0 without having a ground truth reference. This classification error leads to the generation of a false positive, highlighted by the blue vertical line and correctly fixed by the heuristics. Notably, the segmentation towards the end of the COs is rougher, as clearly visible in Figure 6.2e. This behavior is expected, as the CNN model has been trained to detect the start of the COs, not their end. However, the end of the COs can be easily identified by looking at the segmentation plots, as the COs are always followed by a period of noise or the start of a new CO. By comparing the position of the ground truth and the position of the



(a) `Chameleon BASE` segmentation.

(b) `Chameleon DFS` segmentation.

(c) `Chameleon RD` segmentation.

(d) `Chameleon MRP` segmentation.

(e) `Chameleon CHF` segmentation.

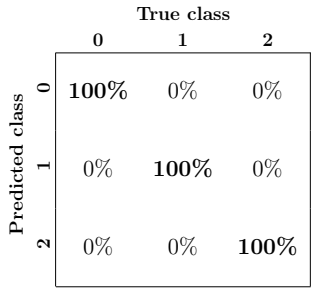
Figure 6.2: Segmentation results for Hound-v3 over the Chameleon dataset in random sample interval. In the top panels, red vertical lines represent the start of the COs' ground truth, while yellow vertical lines represent the predicted CO's starts. The bottom panels show the output classification.

CO found, it can be seen that the latter is always a few hundred samples earlier. This conservative approach was taken to ensure the inclusion of all samples belonging to a CO at the expense of a lower IoU in the segmentation.

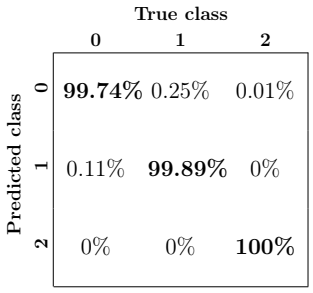
Table 6.7 provides the confusion matrices over the test set for each CNN model, highlighting high classification accuracy on the main diagonal. Nevertheless, the excellent classification in the training phase is not always reflected in the inference phase. For instance, `RD` achieves over 99.99% accuracy in training but exhibits misclassification for classes 2 and 0 during inference in Figure 6.2c. The lower inference performance is probably due to the sliding window procedure adopted, which increases the input variability to the CNN, challenging the classification.

Table 6.7: Test confusion matrices for Hound-v3 on the Chameleon dataset.

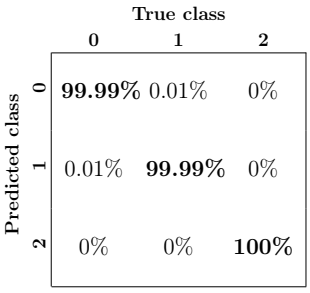
(a) `Chameleon BASE`



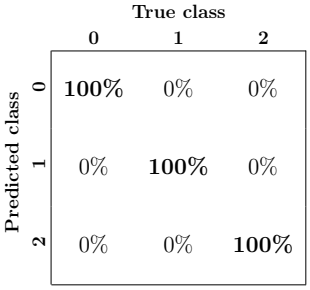
(b) `Chameleon DFS`



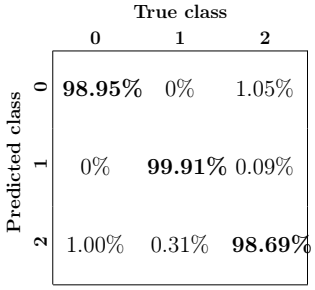
(c) `Chameleon RD`



(d) `Chameleon MRP`



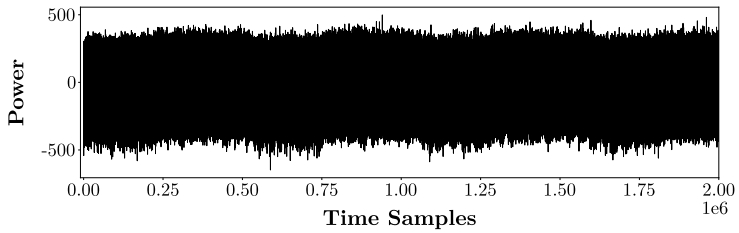
(e) `Chameleon CHF`



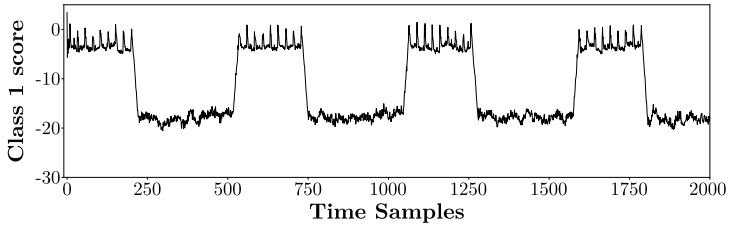
These results confirm the feasibility of developing effective segmentation techniques on the Chameleon dataset. The high IoU values achieved on various hiding techniques demonstrate that it is possible to accurately identify COs within obfuscated traces, even in the presence of significant noise and variability.

Comparison with previous Hound versions Hound-v3 builds upon and enhances seminal Hound versions by accurately identifying both the start and end samples of CO executions, yielding two significant advantages. First, it enables the calculation of the IoU metric, a critical measure of segmentation accuracy. In contrast, prior works only detect the start of encryption, precluding a precise computation of IoU. Second, by delineating both start and end samples, Chameleon enhances the robustness of the proposed segmentation methodology, enabling a more accurate identification of false positives and false negatives and using two heuristics to mitigate such errors.

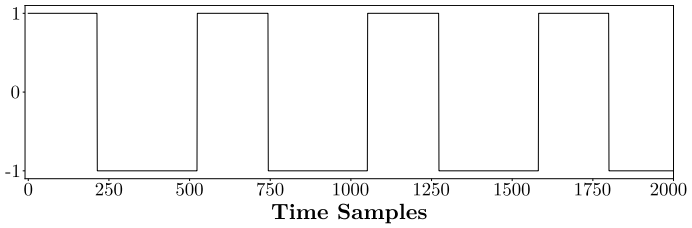
Table 6.6 presents a comparative analysis of our side-channel segmentation performance across all three Hound versions, using the Chameleon dataset. FNR and FPR are reported for all techniques, while IoU values are exclusive to this study due to its ability to pinpoint both start and end samples, a distinction that precludes fair IoU comparisons with earlier works. Although [CGL+24] has been developed to tackle only random



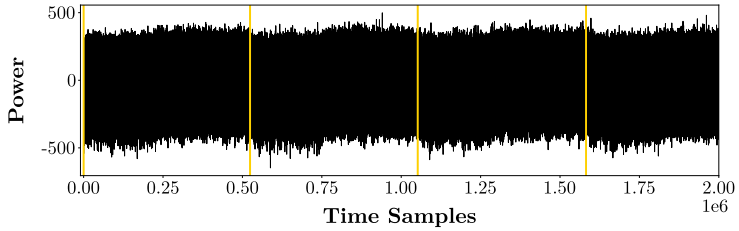
(a) A portion of the input trace containing 4 AES executions.



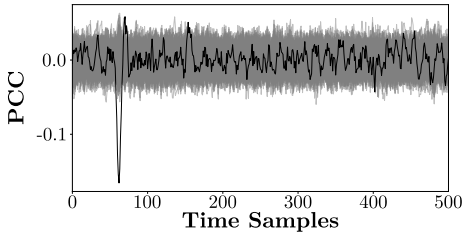
(b) Output of Hound-v1's `Sliding Window Classification` stage.



(c) Output of Hound-v1's `Screening` stage.



(d) AES locations on the portion of the inference trace.



(e) CPA results for the first key byte. The correct key is in black color.

Figure 6.3: Example of the attack pipeline to extract the secret key from a side-channel trace affected by RD-4 random delay that contains 4 AES executions mixed with noise applications.

Table 6.8: Hound-v1 segmentation and SW-CPA results targeting AES-128. Results consider RD-2 and RD-4 settings, and the presence (or not) of general-purpose applications interleaved with the COs.

Random Delay Configuration	General-Purpose Applications	FPR (%)	FNR (%)	SW-CPA (N. COs)
RD-2	✓	0	0	3 695
	✗	0	0	1 125
RD-4	✓	0	0	3 365
	✗	0	0	1 220

delay, it performs well across all hiding techniques, particularly on `BASE` and `CHF`. Similarly, [GCZ24] demonstrates promising results, achieving improved performance across all metrics. However, this study is the only one with perfect FNR and FPR scores for all hiding countermeasures. The excellent results are attributed to the novel tailored heuristics for error refinement, showcasing the robustness and precision of the proposed segmentation methodology.

6.1.4. A complete attack flow

We present a comprehensive attack flow that takes an unknown side-channel trace protected by random delay countermeasures and retrieves the secret key. The segmentation is performed leveraging Hound-v1 while the key is retrieved by CPA [BCO04] with a sliding window alignment [CCD00] (SW-CPA) as the effective side-channel attack. The CPA targets the *sub-byte* intermediate. A slight sliding window alignment is used to fix minor misalignment due to the rough estimation of the beginning of the COs and to mitigate the presence of random delay countermeasure. Figure 6.3 shows the results of applying each step of the proposed Hound-v1 inference pipeline to locate the COs into a portion of a trace containing four executions of AES-128, i.e., the CO. The entire side-channel trace contains 512 COs collected from a platform implementing a random delay mechanism. Starting from the portion of the side-channel trace in input, Figure 6.3b depicts the segmentation output of the `Sliding Window Classification` stage that correctly highlights the presence of four COs interleaved with random general-purpose applications. The number of samples in the x-axis of the plot is reduced by a factor of 1 000, equal to the value of stride s during the classification. The `Screening` stage allows cleaning the noise from the classification output to deliver a clear square wave locating the COs (see Figure 6.3c). The time instants locating the beginning of the four COs are applied to the original side-channel trace, thus allowing an easy time alignment (see Figure 6.3d). At

last, Figure 6.3e shows the results of the attack over the first key byte in terms of Pearson Correlation Coefficient (PCC).

Table 6.8 summarizes the results of the attack flow on AES-128, considering the two random delay configurations, i.e., RD-2 and RD-4, and the presence or absence of general-purpose applications interleaved with the COs. The successful segmentation in all the scenarios allows the SW-CPA to retrieve all the secret key's bytes with a minimum of 1 125 COs (see RD-2 without general-purpose applications).

6.1.5. Discussion and limitations

Although the segmentation technique and error-handling heuristics detailed in Section 3.2 demonstrates excellent results on the Chameleon dataset, achieving a 0% FNR and FPR in real-world scenarios is impractical. Such identification errors can significantly impact attack effectiveness by causing mismatches between COs and their corresponding plaintexts. In a naive matching approach, the position of the error significantly affects its impact. If a mismatch occurs early in the sequence, it causes a cascading misalignment for all subsequent CO executions. On the other hand, mismatches that occur later in the sequence are less disruptive since all preceding traces remain correctly matched with the corresponding plaintexts. However, for deep learning-based profiled attacks, the impact of identification errors is minimal, as such attacks typically require very few traces to succeed. For instance, even with an identification error in the 15th CO, the correct key may already have been recovered using the first 14 CO executions. Classical unprofiled attacks are more susceptible, as they generally depend on a higher number of correctly matched COs.

To quantitatively assess the impact of identification errors on Chameleon's attack performance, we considered a 0.05% error rate. For the most challenging subdataset, i.e., `CHF`, which requires 104 traces for a successful CNN-based attack, the probability of attack failure is 5.07%. In comparison, for the `BASE` subdataset, requiring only 14 traces, the failure probability decreases to 0.70%.

The proposed framework uses a window-based classification to segment COs in side-channel traces. A window-based classification means that the CNN classifies each window independently, resulting in a sequence of classifications rather than a proper segmentation. While our methodology effectively detects COs with high IoU and low false detection rates, future work will focus on developing a more accurate segmentation technique to classify each individual sample within a side-channel trace as part of a CO or not. By achieving a more granular segmentation, we can refine attack strategies and improve the overall

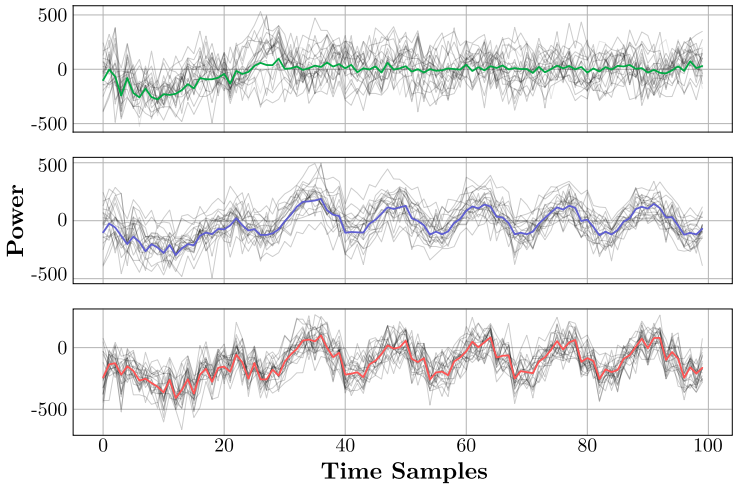


Figure 6.4: Qualitative comparison between the `DFS_DESYNCH` database (first row), the corresponding interpolated traces (second row), and power traces acquired with no DFS enabled (third row), over a range of 100 samples. Each colored line is the sample-wise average.

effectiveness of real-world side-channel analysis.

6.2. Owl side-channel attack results

This section presents the results of the Owl methodology presented in Section 4.1, which includes the evaluation of the trace interpolation, segmentation, and classification steps, as well as the effectiveness of a classical SCA methodology, i.e., a template attack, after the Owl's resynchronization.

6.2.1. Trace interpolation

The interpolation block is designed to recover information lost due to trace desynchronization. The goal is to evaluate the quality of the information recovered from the desynchronized traces by computing the similarity between interpolated and static traces. We opted for the Pearson Correlation Coefficient (PCC) to quantify the similarity between the interpolated and static traces (see Equation 2.1). The coefficient value ranges from -1 up to $+1$, and an absolute value of 1 denotes a perfect linear relationship.

Configuration We compared subsets of traces from two sources: *i)* power traces from the `DFS_DESYNCH` database and *ii)* static reference traces acquired from JARVIS with no desynchronization. Both subsets included 500 traces with the same key value, and their reference frequency was set to 45 MHz. For this comparison, we used the true labels of the

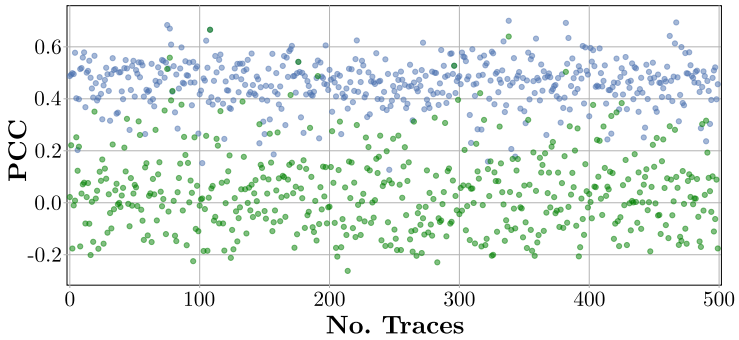


Figure 6.5: Pearson Correlation Coefficient computed between static-desynchronized traces (green points) and static-interpolated traces (blue points).

frequencies instead of the learned segmentation model to focus only on the interpolation performance. For each of the 500 static traces, we computed the PCC with both the corresponding interpolated trace and the corresponding desynchronized trace.

Results After applying interpolation, the evaluation results demonstrate a significant improvement in the correlation between the traces. Figure 6.4 shows a qualitative result of this evaluation procedure. The first row plots the raw desynchronized traces, the second row shows the interpolated traces at 45 MHz, and the third row plots reference static traces. It can be clearly noticed how the interpolation produces visibly similar signals to the reference static ones. Notably, the power reported in Figure 6.4 refers to the voltage drop rather than the actual power consumption. Considering side-channel analysis, the use of decoupled power rails is common to measure a single quantity, i.e., voltage drop, rather than both voltage and current.

Figure 6.5 illustrates how the average PCC between the desynchronized traces and the static ones (green points) increases from around 0 to 0.5 after interpolation (blue points). Similarly, the standard deviation decreases from 0.2 to 0.1. These results indicate that the interpolation process successfully recovers information lost due to desynchronization, making the interpolated traces more consistent and similar to the original static traces.

6.2.2. Convolutional neural network

The CNN is evaluated based on its ability to classify side-channel traces according to their frequency. Here are detailed the parameters used for building the training dataset as described in Section 4.1.1.

Table 6.9: Test confusion matrix of the trained CNN.

		True frequency					
		35 MHz	**40 MHz**	**45 MHz**	**50 MHz**	**55 MHz**	**60 MHz**
Predicted frequency	**35 MHz**	**99.64%**	0.11%	0.08%	0.05%	0.10%	0.08%
	40 MHz	0.08%	**99.67%**	0.09%	0.07%	0.15%	0.06%
	45 MHz	0.03%	0.07%	**99.69%**	0.05%	0.13%	0.06%
	50 MHz	0.06%	0.03%	0.03%	**99.76%**	0.10%	0.06%
	55 MHz	0.07%	0.07%	0.06%	0.05%	**99.46%**	0.06%
	60 MHz	0.11%	0.05%	0.05%	0.04%	0.06%	**99.68%**

Configuration The training dataset has been created starting from two 500-sample windows randomly extracted from each profiling trace in the `DFS_DESYNCH` database, resulting in a total of 256 000 windows. Intervals of at least 1 000 samples were considered to minimize border effects, and the center 500 samples were chosen for each window.

To avoid border effects near the frequency change, we considered intervals of at least 1 000 samples and selected the 500 samples in the center. As in standard neural networks training, we divided the collected dataset into training, validation, and test sets. Specifically, the 60% of the collected windows is used for training, while the remaining 40% is equally divided into validation and test sets.

By defining an epoch as a complete optimization step over the training set, we trained the network for a total of 8 epochs on an NVIDIA RTX™ 6000 GPU using Adam [KB15] to minimize the cross-entropy loss (see Equation 3.1). The mini-batch size was set to 128 with a learning rate of 0.001. Starting from the architecture in [WYO17], we performed a comprehensive hyperparameter search, such as the number of layers, kernel sizes, and activation functions, using a random search strategy [BB12]. A separate validation set was used to evaluate the different configurations and identify the optimal combination for our specific dataset and task. The error on the validation set is evaluated after each epoch, and the final network is chosen as the one with the lowest validation error.

Results Table 6.9 reports the test CM at the end of the CNN training. The high concentration and diagonal dominance of the values in the confusion matrix indicate that the trained CNN effectively discriminates between different frequencies. This suggests high accuracy in the network's frequency classification task.

Table 6.10: Intersection over Union (IoU) mean and standard deviation computed over the entire attack set of the DFS_DESYNCH database. Table reports both the averaged IoU value and the IoU values aggregated by frequency.

	Per-Frequency IoU (%)						Average
	35 MHz	**40 MHz**	**45 MHz**	**50 MHz**	**55 MHz**	**60 MHz**	IoU
DFT	90.28 ± 6.62	90.06 ± 6.79	90.09 ± 6.61	92.34 ± 5.32	89.31 ± 7.42	60.56 ± 14.94	73.40 ± 3.95
Grad-CAM	99.44 ± 0.08	99.60 ± 0.09	99.37 ± 0.08	99.57 ± 0.08	99.58 ± 0.09	99.50 ± 0.08	99.51 ± 0.06

6.2.3. Segmentation

The core of the Owl methodology relies on the trace segmentation based on Grad-CAM which classifies each sample according to its frequency (see Section 4.1.2). The analysis focuses on the attack set of the DFS_DESYNCH dataset.

Configuration The proposed Grad-CAM-based segmentation is compared against a segmentation method based on the analytical Discrete Fourier Transform (DFT). Both methods are evaluated by calculating the total IoU and the IoU for each frequency over the attack set of the DFS_DESYNCH dataset. The experiment considers all the 256 keys of the attack set of the database by aggregating the segmentations of the 500 traces.

Results The proposed Grad-CAM-based segmentation performs better than the analytical DFT method across all frequencies. It also demonstrates greater stability in performance between different classes, unlike the DFT segmentation, whose accuracy varied depending on the frequency.

A significant advantage of the Grad-CAM approach is its ability to focus on the relevant frequency segments within the traces, excluding noise or irrelevant frequency components. In contrast, the DFT method is more susceptible to capturing frequencies outside the range of interest, leading to an inflated overall error rate in the IoU metric.

The results presented in Table 6.10 quantify this advantage. The Grad-CAM-based segmentation achieves a substantially higher average IoU of 99.40% with low variance, compared to only 73.40% for the DFT method.

A qualitative comparison between the Grad-CAM-based segmentation and the ground

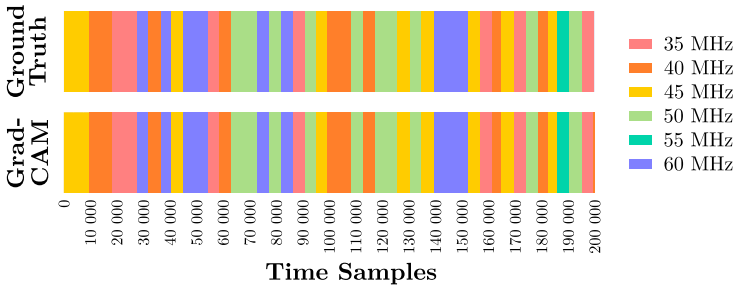


Figure 6.6: Segmentation qualitative comparison of a trace from the attack set of the `DFS_DESYNCH` database. The first row reports the ground-truth segmentation while the second row plots the Owl Grad-CAM-based segmentation.

truth for a single trace is visualized in Figure 6.6, further supporting its effectiveness for such a task.

6.2.4. Template attack

To further validate the effectiveness of the Owl resynchronization pipeline, we performed a template attack [CRR03] as last stage of the methodology. A template attack is chosen for its deployment simplicity and demonstrated effectiveness. The following presents attack configuration details and shows the template attack results obtained using the proposed Owl methodology.

Configuration Different templates were profiled on the `DFS_DESYNCH` traces of Section 5.4. All templates target the leakage produced by the `sbox` substitution in the first AES round, i.e., $\texttt{sbox(p[0]} \oplus \texttt{k[0])}$, where `p[0]` and `k[0]` are the first plaintext and key bytes respectively. For profiling and attack phases we used 500 traces per key value.

Two leakage models are considered. In the *Hamming Weight* model the leakage is assumed proportional to the number of set bits of the intermediate value, $\text{HW}(\texttt{sbox(p[0]} \oplus \texttt{k[0])})$, yielding nine classes. In the *Identity* model the leakage is assumed to reflect the full intermediate byte value, $\texttt{sbox(p[0]} \oplus \texttt{k[0])}$, yielding 256 classes.

Three different template pipelines were evaluated: *i)* Raw templates are built directly on the original, dynamically and randomly desynchronized traces; *ii)* DFT templates are built after frequency segmentation based on the Discrete Fourier Transform, followed by interpolation, i.e., resynchronization to 45 MHz; *iii)* Owl templates are built after resynchronization using the proposed Owl pipeline (see Section 4.1). All templates use a sliding window alignment procedure [CCD00], i.e., averaging consecutive samples, to reduce the input noise and size. For Raw and DFT templates we repeated experiments

Table 6.11: Guessing Entropy (GE) and Guessing Distance (GD) comparison over `DFS_DESYNCH` and `ASCAD` datasets using Hamming Weight and Identity as leakage models. Two State-of-the-Art DL-based SCA techniques [HGG19, BPS+20] are compared with the proposed Owl methodology. All Templates use minor aggregation over time and PCA as POI selection method.

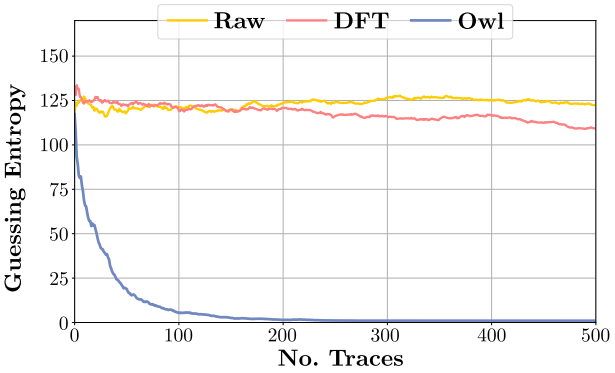
			Raw	**DFT**	**[HGG19]**	**[BPS+20]**	***Owl***
`DFS_DESYNCH`	**Hamming Weight**	GE	122.450	109.336	–	–	1.009
		GD	−0.460	−0.400	–	–	0.137
	Identity	GE	130.012	122.641	98.926	113.895	1.000
		GD	−0.497	−0.481	−0.436	−0.614	0.255
`ASCAD`	**Identity**	GE	1.000	1.000	1.050	2.400	1.000
		GD	0.617	0.617	0.279	0.421	0.617

with aggregation lengths of 50, 100, and 500 samples, for Owl we report results with $N_{agg} = 100$. POI selection was performed with PCA [WEG87], retaining the five principal components in all experiments.

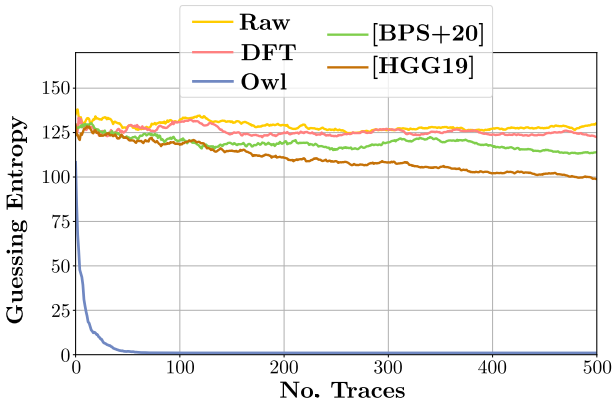
Results Table 6.11 presents the attack results divided by leakage model. The results clearly demonstrate the effectiveness of the proposed methodology. For raw traces, i.e., with dynamic frequencies, the Guessing Entropy (GE) is 122.450 and 130.012 for the *Hamming Weight* and *Identity* models, respectively, indicating that the templates perform no better than a random guess. Even though different aggregation values have been tested, no significant improvement has been observed and Table 6.11 reports only the best results. Templates built on DFT-segmented traces show similarly poor performance. Segmentation errors degrade the attack effectiveness and the sliding window alignment does not compensate, yielding GE values of 109.336 and 122.641 for the two models.

By contrast, traces resynchronized with the proposed Owl pipeline are effectively attackable. The *Identity* leakage model reports GE is 1.000, i.e., perfect success, with a GD of 0.255, which indicates strong isolation of the correct key. The *Hamming Weight* model also succeeds in the attack, with GE 1.009 and GD 0.137. The different results on the two models suggests that building templates on 256 classes delivers better performance.

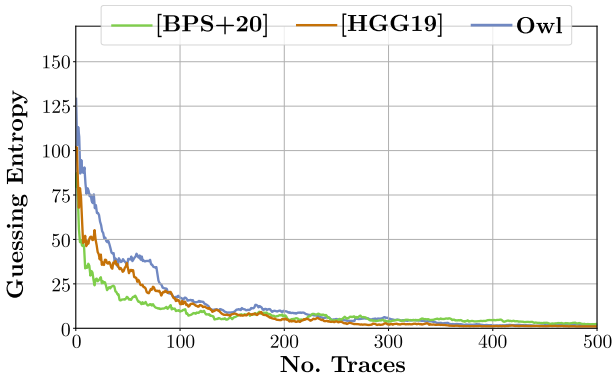
Figure 6.7 shows the GE as a function of the number of traces used for the attack. For raw traces the GE does not decrease as the number of traces increases. In contrast, for resynchronized traces the GE falls to values close to one in fewer than 300 traces for both leakage models.



(a) Hamming Weight as leakage model over `DFS_DESYNCH`.



(b) Identity as leakage model over `DFS_DESYNCH`.



(c) Identity as leakage model over `ASCAD`.

Figure 6.7: Guessing Entropy results over `DFS_DESYNCH` and `ASCAD` dataset. All Templates use minor aggregation over time and PCA as POI selection method.

6.2.5. Comparison with the state of the art

Table 6.11 also depicts a comparison between the proposed Owl methodology and two State-of-the-Art DL-based SCAs. All models were trained both on the `DFS_DESYNCH` and `ASCAD` [BPS+20] datasets, attacking as intermediate `sbox(p[0]` $\oplus$ `k[0])` with 256 classes. Traces from `ASCAD` have been desynchronized up to 100 samples, following the random desynchronization model of the dataset. On the `DFS_DESYNCH` dataset, the technique in [BPS+20], which uses a CNN model, achieves a GE of 113.895 and a GD of -0.614. These results show that this technique fails to recover the secret key under high desynchronization. The second technique [HGG19] uses a different approach that also relies on a CNN model. However, this technique also performs poorly, with a GE of 98.926 and a GD of -0.436. In contrast, our Owl methodology obtains a perfect Guessing Entropy of 1.000, consistently recovering the secret key. Therefore, given the same amount of information, we can conclude that our methodology is the most effective among the

three.

When we applied these methodologies to `ASCAD` [BPS+20], Owl methodology continued to outperform the others. A visual comparison of the GE over the number of traces used for the attack is reported in Figure 6.7c. The GE is close to one for all the techniques, meaning that the three methodologies are effective in recovering the secret key. However, the GD of 0.617 of Owl methodology is higher than the other two, showing that our methodology is more effective in isolating the correct key. Notably, `ASCAD` traces were collected from a computing platform operating at a single operating frequency. Therefore, Owl's segmentation and interpolation processes are straightforward.

6.2.6. Discussion and limitations

While the experimental results validate the effectiveness of the Owl methodology in mitigating frequency-based desynchronization, the continuous evolution of deep learning architectures suggests that its role is best defined as a specialized synchronization primitive. The primary contribution of Owl lies in its modular architecture providing the evaluator with a resynchronized leakage profile, which is essential for the application of standard leakage assessments. While it is possible to optimize a high-throughput CNN that map desynchronized traces directly to secret keys, such an approach would sacrifice the versatility offered by Owl, which remains agnostic to the specific leakage analysis applied in the second phase. Therefore, the inherent computational overhead due to the resynchronization is considered a trade-off to ensure a high-fidelity, interpretable security evaluation. Rather than competing with cutting edge end-to-end models, Owl complements them by providing a foundational layer for multi-countermeasure scenarios.

The proposed methodology demonstrated high precision when targeting a commercial DFS module with six operating points, representative of standard low-power IoT protection schemes. We acknowledge that as the granularity of frequency scaling increases toward hundreds of operating points, the Grad-CAM-based segmentation may encounter precision limits. The integration of higher-resolution segmentation techniques and attention mechanisms will be interest in a future work.

6.3. Dynamic voltage and frequency scaling countermeasure results

This section discusses the influence of dynamic voltage and frequency scaling on the side-channel security of FPGA-based cryptosystems. First, we present an extensive ex-

Table 6.12: Values in terms of operating voltage, clock frequency, clock phase shift, and used chip, for each experimental configuration.

	ID	Num. of steps	Voltage [V]	Frequency [MHz]	Phase [°]	Chip Train (Attack)
Chip	C1	1	1	50	0	Artix-7 100 (Artix-7 35)
	C2	1	1	50	0	Artix-7 35 (Artix-7 100)
DVS	V1	3	[0.99; 1.01] step 0.01	5	0	Artix-7 100 (Artix-7 100)
	V2	2	{0.75, 1.05}	5	0	
	V3	11	[0.75; 1.05] step 0.03	5	0	
DPS	P1	9	1	50	[0; 30] step 3.75	
DFS	F1	9	1	[38.375; 39.5] step 0.125	0	
	F2	7	1	[30; 65] step 5.0	0	
	F3	6	1	[25; 75] step 10.0	0	
	$F3_{125}$	401	1	[25; 75] step 0.125	0	

perimental evaluation of the impact of run-time variability in Section 6.3.1. Then, we present the experimental results of the *Rabbit* countermeasure in Section 6.3.2.

6.3.1. The impact of run-time variability on side-channel attacks targeting FPGAs

The impact of run-time variability on side-channel security lacks a comprehensive experimental evaluation. The following presents an analysis of variability as a potential side-channel countermeasure by analyzing changes in clock frequency, clock phase, voltage, and different FPGA instances. The analysis leverages the rDVFS module illustrated in Section 4.2, designed to mimic process variability through small and rapid variations

in operating parameters.

Configurations

Deployment The proposed random DVFS has been deployed into the JARVIS framework [ZGG25] leveraging the NewAE Technology CW305 board [New18], which features an FPGA socket that allows changing the mounted FPGA chip. The power traces were collected from the 20 dB-amplified output of the CW305 board, sampling at 250 Msamples/s with a resolution of 12 bits. As cryptographic operation, we used the constant-time AES-128 encryption from the OpenSSL library [Tcl23].

Design space Our analysis considered the trace desynchronization achieved by scaling each parameter, i.e., clock frequency, clock phase, operating voltage, and different FPGA chips, in isolation. Notably, our *rDVFS* is enabled for the entire encryption. Thus, a new configuration is requested as soon as one is locked. To this end, each encryption observes multiple clock frequency and voltage reconfigurations.

For each configuration, Table 6.12 reports the boundary values and the step of the considered parameter.

Attack methodology To assess the side-channel vulnerability, we utilized CPA and template attack, focusing on the S-Box operation in the initial AES round. CPA is performed up to 100k traces on the *train* chip of Table 6.12, while the template has been trained considering 1k traces per key value.

Multiple attacks were executed for each scenario and method. The primary attack used unprocessed traces, while subsequent ones incorporated post-processing, specifically sample aggregation [CCD00] and a high-pass filter (HPF) with a 125 kHz cutoff to eliminate low-frequency signal components.

Experimental results

Table 6.13 summarizes the experimental results. The *Synch* configuration reports the quality metrics for the reference design with no desynchronization. For each scenario (*ID*), results are reported in terms of *PCC* values, number of traces to recover all 16 bytes using *CPA*, and effectiveness of *template attack*. Notably, Pearson Correlation Coefficient (PCC) between the power traces and the power estimation is used as an equivalent representation of SNR. For each scenario, results are reported with and without the HPF and different levels of aggregation. Notably, the aggregation can resynchronize the traces by averaging

continuous samples at the risk of a decreased SNR.

Inter-chip variability and clock phase The inter-chip process variability on AMD Xilinx Artix-7 FPGAs is too small to prevent an attack using a model trained on one FPGA instance to target another (see $C1$ and $C2$ in Table 6.13). Notably, such results contrast those reported from analyzing the process variability targeting ASIC [RSVC+11], thus motivating the need for separate investigations. Similarly, dynamic phase shift provokes a negligible degradation in the side-channel attacks' success (see *DPS* in Table 6.13).

Operating voltage The attack's success rate diminishes as the operating voltage range increases. Considering the *Raw* traces, PCC lowers from 0.428 in $V1$ to 0.102 in $V3$, thus highlighting the benefit of the voltage scaling. Conversely, using a high-pass filter on the DVS collected traces shows a non-negligible improvement of PCC, as well the efficacy of *CPA* and *template attack*. The motivation behind such behavior is the low-frequency component over-imposed by the actuators on the power consumption. The scaling actuations provoke severe oscillations in the side-channel signal, thus reducing PCC values and the effectiveness of the various SCAs. However, the low dynamic of the over-imposed signal can be removed by applying a high-pass filter to the measured traces.

Clock frequency Experimental results confirm the effectiveness of dynamic frequency scaling as desynchronization countermeasures. Similarly to DVS, the side-channel attacks degrade with the increase in the range of the DFS actuator. For example, $F1$ and $F3$ employ a frequency range of $[38.375 - 39.5]$ MHz and $[25 - 75]$ MHz, respectively. The *CPA* and *template attack* results confirm that $F3$ makes the attacks harder than $F1$.

Considering the employed post-processing techniques, the high-pass filter (*HPF*) fails to improve the SCA. In contrast, aggregation severally improves the side-channel attacks. For example, aggregating 100 samples allows *CPA* to succeed for all key bytes with $5k$ traces in the $F1$ scenario (see Table 6.13). Notably, none of the considered attacks succeed in the $F2$ and $F3$ scenarios regardless of the employed post-processing technique, thus highlighting the effectiveness of severe trace desynchronization against SCA.

At last, we analyzed the effect of using a large variety of dynamic frequencies to improve the side-channel resistance as proposed in [DHL+21]. The frequency scenario $F3_{125}$ considers the step between two consecutive operating frequencies equal to 125 kHz. In addition, $F1$ mimics the effect of clock jitter. Our experimental results demonstrate that using a high number of frequencies offers negligible improvements to the side-channel resistance.

Table 6.13: PCC, CPA, and template attack comparison between all the configurations, both with and without filtering the traces and aggregating up to 1k samples (✓if GE = 1).

	ID	Aggregate n samples	PCC		CPA		Template attack	
			Raw	HPF	Raw	HPF	Raw	HPF
Synch		1	0.419	0.388	200	200	✓	✓
		100	0.587	0.555	200	200	✓	✓
		1k	0.557	0.360	200	500	✓	✓
Chip	C1	1	0.419	0.388	200	200	✓	✓
	C2	1	0.353	0.402	200	200	✓	✓
DVS	V1	1	0.428	0.435	200	200	✓	✓
	V2	1	0.089	0.326	10k	500	✗	✓
	V3	1	0.102	0.335	10k	500	✗	✓
DPS	P1	1	0.191	0.185	200	500	✓	✓
		100	0.220	0.202	200	500	✓	✓
		1k	0.170	0.089	500	1k	✓	✓
DFS	F1	1	0.046	0.033	✗	✗	✗	✗
		100	0.215	0.184	5k	5k	✓	✓
		1k	0.252	0.166	5k	5k	✓	✓
	F2	1	0.033	0.034	✗	✗	✗	✗
		100	0.032	0.031	✗	✗	✗	✗
		1k	0.031	0.018	✗	✗	✗	✗
	F3	1	0.040	0.044	✗	✗	✗	✗
		100	0.029	0.031	✗	✗	✗	✗
		1k	0.027	0.024	✗	✗	✗	✗
	$F3_{125}$	1	0.035	0.033	✗	✗	✗	✗
		100	0.031	0.028	✗	✗	✗	✗
		1k	0.028	0.024	✗	✗	✗	✗

6.3.2. Rabbit: dynamic clock randomization to protect against side-channel attacks

Motivated by the promising results of frequency scaling, we developed and implemented the *Rabbit* module (see Section 4.2) to rigorously evaluate it as a countermeasure. We pushed the synthesizable frequencies to their extreme, maximizing the operating range and the frequency granularity, and intensified the experimental campaign. In particular, CPA and TVLA experiments were scaled up to 10 million traces, and standard deep-learning-based SCAs were also evaluated.

Deployment The analysis considers the feasible dynamic of the proposed *Rabbit* and SoC, with a frequency range of [5, 75] MHz. We choose a 0.125 MHz step in the considered range, enabling the actuator to synthesize 561 different frequencies. Notably, *Rabbit* is enabled for the entire experimental campaign. Thus, a new one is requested as soon as a configuration is locked. To this end, each encryption observes multiple clock frequency reconfigurations, i.e., 41 on average.

The design has been implemented on an AMD Xilinx Artix-7 FPGA deployed on the NewAE Technology CW305 board [New18] leveraging the JARVIS framework. The power traces were collected sampling at 250 Msamples/s with a resolution of 12 bits and 20 dB amplification.

Attack methodology To assess the side-channel protection offered by the proposed *Rabbit*, we employed three classical side-channel attack methodologies and two DL-based SCAs (see *Attack Methods* columns in Table 6.14). As cryptographic operation, we used the constant-time AES-128 encryption from the OpenSSL library [Tcl23]. Each attack considers as intermediate the S-Box substitution on the first AES round as the attack model. The classical attacks include Correlation Power Analysis (CPA), a Template Attack using Principal Component Analysis (PCA) for Point of Interest selection (PCA-TA), and a Template Attack leveraging the Fast Fourier Transform (FFT-TA). The Fast Fourier Transform determines the operating frequency of the cryptosystem, enabling the alignment of measurements to a reference frequency through interpolation. Additionally, PCA is used prior to profiling the template. DL-based SCAs comprise the Convolution Neural Network (CNN) offered by [BPS+20], and the Layer-wise Relevance Propagation (LRP) technique [HGG19]. The template attacks and DL-based SCAs are trained on 256 000 power traces, namely 1 000 per key-byte value. In contrast, the CPA is evaluated up to 10 million side-channel measurements.

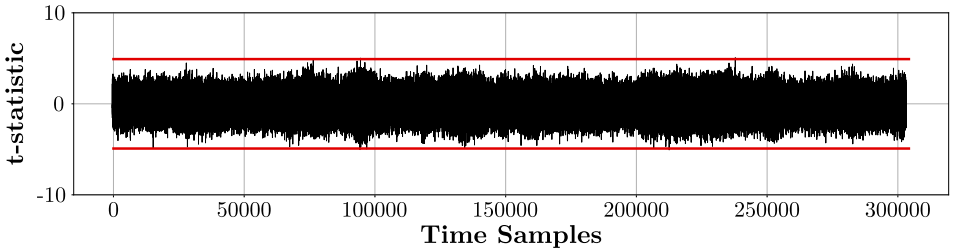


Figure 6.8: TVLA results on ten million traces measured from the protected SoC with *Rabbit*.

Experimental results

The unprotected AES, operating at a stable 50 MHz clock frequency, can reach GE of one with approximately 250 encryptions using CPA, 10 encryptions using PCA-TA and FFT-TA, and less than 3 encryptions using CNN and LRP. These results highlight the vulnerability of the reference design to side-channel attacks. In contrast, the proposed *Rabbit* countermeasure significantly enhances security. With *Rabbit* active, the GE remains above 102.40 across all attack techniques, demonstrating that none of the attack methods could effectively exploit the leaked information. CPA, evaluated with 10 million traces on the protected design, resulted in a GE of 110.00.

We collected 10 million power traces with randomly selected fixed or random plaintexts, covering the entire encryption process. The side-channel measurement has been post-processed on a high-pass filter with a cut-off frequency of 125 kHz to remove low-frequency components due to the actuator. The subsequent TVLA results in a t-statistics with no data points exceeding the ±4.5 threshold, as shown in Fig. 6.8, further affirming *Rabbit*'s strength against first-order leakage.

Table 6.14 summarized *Rabbit* compared with related state-of-the-art works. The notation *N/A* indicates metrics not specified in the original papers. Our design uses minimal resources, with only 82 LUT, 56 FF, and 1 BRAM. Only the design from [HDL+20] outperforms ours in terms of area, utilizing 36 LUTs, 9 FFs, and no BRAM. *Rabbit* maintains a remarkable performance of 100.00%, signifying no timing overhead, which stands out as the best result compared to other approaches, where performance losses range from 1.70% to 65.96% relative to unprotected platforms. Our null performance degradation has to be attributed to the dual-MMCM architecture, which enables the reconfiguration of one MMCM while the other continues to provide stable clocking to the computing platform. The security assessment has been performed with a wide coverage of attack methodologies, spacing from classical attacks to DL-bases SCA. *Rabbit* resists CPA, PCA-TA, FFT-TA, CNN, and LRP attacks. It achieved a TVLA of 10×10^6 traces, with

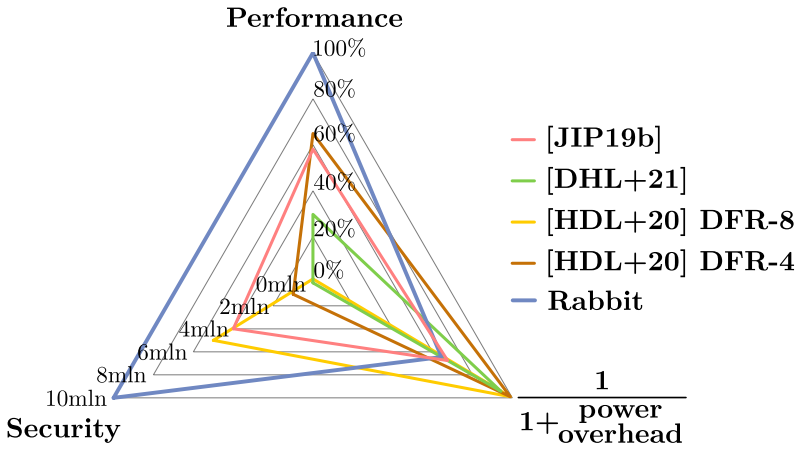


Figure 6.9: Performance, power, and security comparison with state-of-the-art proposals. Security is measured in millions of TVLA traces.

no leakage detected after analyzing ten million traces, marking the highest TVLA among all approaches, which range from 10×10^3 to 5×10^6 traces. However, the design incurs higher power consumption due to the dual-MMCM setup, with a power overhead of 1.55×, compared to 1.01× in the best alternative design [HDL$^+$20]. Fig. 6.9 illustrates *Rabbit*'s advantages in security and performance despite its slightly higher power consumption when compared with competing designs.

Overall, the proposed *Rabbit* countermeasure effectively strengthens AES security, offering robust protection against a wide range of side-channel attacks while maintaining resource efficiency and zero performance overhead.

Table 6.14: Rabbit's resources, performance, power, attack methods, and security comparison with State of the Art.

	Resources					Performance	Power	Attack Methods		Security
	LUT	FF	BRAM	MMCM	BUFG	(%)		Classical	DL-SCA	(TVLA Traces)
[LOM08]	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	60.98	4.11×	CPA	*N/A*	*N/A*
[Fri12]	2507	2245	10	1	*N/A*	18.83	3.43×	DPA	*N/A*	*N/A*
[GM11]	23	64	0	1	7	20.96	*N/A*	CPA	*N/A*	*N/A*
[RBBC18]	102	266	0	2	12	11.26	*N/A*	*N/A*	*N/A*	4×10^5
[JIP19b]	*N/A*	*N/A*	20	2	*N/A*	58.14	1.48×	PCA-CPA DTW-CPA	*N/A*	4×10^6
[HDL+20] DFR-4	30	9	0	1	12	65.94	1.01×	CPA SW-CPA FFT-CPA	FFT-MLP LRP	1×10^6
[HDL+20] DFR-8	36	9	0	1	23	1.70	1.01×	CPA SW-CPA FFT-CPA	FFT-MLP LRP	5×10^6
[DHL+21]	2058	218	0	1	*N/A*	29.76	1.06×	CPA	CNN	$< 10 \times 10^3$
Rabbit	82	56	1	2	5	100.00	1.55×	CPA PCA-TA FFT-TA	CNN LRP	10×10^6

7 | Conclusions and Future Directions

Side-channel attacks explore how unintentional physical emissions produced by computing devices can be exploited to recover secret information. These attacks represent a critical security threat since they bypass the mathematical security of cryptographic algorithms and often remain undetected by the target system. This thesis explicitly tackles the challenge of evaluating and strengthening the security of digital circuits against side-channel attacks, with an emphasis on realistic threat models and on reducing the cost and complexity of the analysis. The proposed methodology follows the entire side-channel analysis pipeline, supported by a common experimental infrastructure named JARVIS. As a modular hardware-software platform, JARVIS provided the necessary foundations for reproducible research on FPGA-based systems, enabling the streamlined acquisition and analysis of COs required to validate the core contributions of this work.

In this context, the longstanding challenge of segmenting COs in obfuscated and highly desynchronized traces is addressed by the Hound deep-learning pipeline, together with the Chameleon dataset, which provides robust resources for automatically locating and aligning COs, even in the presence of advanced hiding countermeasures. These resources enable systematic benchmarking and facilitate the development of new segmentation techniques.

On the attack side, the Owl methodology demonstrates that deep learning effectively preprocesses and realigns traces affected by commercial dynamic frequency scaling implementations, enabling successful key recovery under conditions previously considered highly resistant to analysis. The DFS_DESYNCH dataset further supports research in this area by providing real-world, highly desynchronized traces for benchmarking purposes.

In terms of defense, the thesis introduces Rabbit, a hardware module for dynamic and fine-grained frequency randomization, which has been shown to provide strong resistance against a wide range of side-channel attacks. Extensive experimental campaigns, conducted through the integrated experimental workflow, highlight the effectiveness of fre-

quency scaling as a hiding technique, while also revealing the limitations of other approaches, such as phase shift and voltage variability.

Overall, this thesis delivers a comprehensive workflow for side-channel analysis in realistic environments, bridging the gap between theoretical research and practical security evaluation. The contributions made support the development of secure digital systems and provide valuable resources for the research community, underscoring the symbiotic relationship between attack and defense. Effective attacks are necessary to identify vulnerabilities and assess the strength of countermeasures, driving an iterative cycle where each new attack informs the refinement of defenses. Despite ongoing advancements, information security remains an evolving field as no truly secure system can exist but only those that have not yet been compromised.

7.1. Current limitations and future directions

While the methodologies presented in this thesis demonstrate significant advancements in addressing the complexities of modern side-channel analysis, they are subject to specific constraints that define their current operational boundaries. Detailed discussion of the limitations inherent to the Hound and Owl frameworks have been previously discussed in Section 6.1.5 and Section 6.2.6, respectively.

This final section outlines common challenges open in the SCA community by acknowledging the broader limitations of this thesis. Future research directions build upon the foundational contributions of this work, aiming to enhance the robustness and portability of deep learning-driven side-channel evaluations.

7.1.1. Combined hiding and masking countermeasures

A current limitation of this thesis is its exclusive focus on hiding-based countermeasures, leaving the evaluation of masking techniques outside the current scope. Masking is a widely adopted countermeasure that aims to protect cryptographic implementations by randomizing the intermediate values processed during computation [GP99]. By splitting sensitive data into multiple shares, masking effectively obscures the correlation between the side-channel emissions and the secret information. However, recent studies have shown that advanced side-channel attacks, particularly those leveraging deep learning, can bypass masking countermeasures [BPS+20, RWPP21].

While this thesis provides insights into the resilience of state-of-the-art hiding techniques, the detailed analysis of these countermeasures in the context of masked ciphers is an im-

portant and complementary area of research and an open direction for future research. Integrating masked implementations into a new version of the Chameleon dataset would provide a valuable resource for advancing the understanding of the interplay between masking and hiding techniques. This extension would facilitate the development and benchmarking of novel attack methodologies specifically designed to target complex scenarios.

A fundamental step towards publishing this dataset is its validation, ensuring that the collected traces contain the leakage necessary for an effective attack. The validation process is non-trivial, as it requires careful consideration of the masking scheme, the hiding techniques employed, and the specific characteristics of the side-channel emissions. Preliminary experiments have been conducted to assess the feasibility of attacking masked implementations on the JARVIS platform, leveraging deep-learning techniques. These experiments involve a systematic exploration of various neural network architectures, hyperparameter tuning, and data augmentation strategies to identify configurations that can successfully recover the masked secret keys. However, the results highlight the challenges associated with overfitting and over-specialization, as well as the need for robust generalization.

7.1.2. Portability threat model

Profiled side-channel analysis has been extensively studied in the literature since 2003 [CRR03]. Most works, including this thesis, are limited to a simplified and unrealistic adversary model where the profiling and attack phases are performed on the same device instead of relying on a clone device for profiling. Such threat model is significantly biased, which could lead to highly inaccurate conclusions about the effectiveness of side-channel attacks. An actual portability threat model would be more difficult to analyze since the measurements from the two devices may not follow the same distribution, causing over-specialization effects where a neural network learns to generalize only for a specific dataset and cannot generalize for other datasets, i.e., devices [vdVPB20, BCH$^+$20].

To address this challenge, future work will develop segmentation and attack methods that remain effective across different device instances, microarchitectures, and instruction set architectures (ISA). Hound already shows robustness to variations in the target COs and to several hiding countermeasures. We believe that, thanks to appropriate learning-based techniques, Hound can be extended to tolerate changes in ISA and hardware platforms. The long-term objective is to make Hound generalize to unseen devices and architectures without requiring full retraining of the neural model. To pursue this goal, we will inves-

tigate transfer learning, domain adaptation, and ensemble strategies to improve model generalization across diverse environments and measurement conditions, enabling reliable CO segmentation even on devices and architectures not seen during training.

On the attack side, we aim to develop profiling methods that can recover keys from devices different from those used for profiling, i.e., under a perfect portability threat model. Preliminary experiments on unprotected AES-128 using template attacks indicate that portability is achievable, although it generally requires more attack traces than when profiling and attacking the same device. Section 6.3.1 and Table 6.13 present portability results for template attacks across different AMD Xilinx Artix-7 FPGA configurations deploying the same JARVIS platform and AES implementation. Such results demonstrate successful key recovery when the profiling and target devices differ. However, when it comes to deep-learning attacks designed to break modern countermeasures, we believe that portability is more challenging. Deep-learning models are in fact more prone to overfitting, and seminal studies in literature already highlight this tendency [PPM+23]. Therefore, building a model that can attack masking countermeasures regardless of the hardware platform remains a major challenge.

As a final future work, we would like to collect and release a dataset to boost the study of attack methodologies with a portability threat model. The dataset will include measurements from multiple acquisition boards and FPGA platforms, in addition to the possibility of switching between different microarchitectures and ISAs.

Copyright

Bibliography

[ABP21] Francesco Antognazza, Alessandro Barenghi, and Gerardo Pelosi. Metis: An integrated morphing engine cpu to protect against side channel attacks. *IEEE Access*, 9:69210–69225, 2021.

[ABPS15a] Giovanni Agosta, Alessandro Barenghi, Gerardo Pelosi, and Michele Scandale. Information leakage chaff: Feeding red herrings to side channel attackers. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

[ABPS15b] Giovanni Agosta, Alessandro Barenghi, Gerardo Pelosi, and Michele Scandale. The MEET Approach: Securing Cryptographic Embedded Software Against Side Channel Attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(8):1320–1333, 2015.

[ABuH+19] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. Port contention for fun and profit. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 870–887. IEEE, 2019.

[AHC+24] Pierre Ayoub, Aurélien Hernandez, Romain Cayre, Aurélien Francillon, and Clémentine Maurice. Phasesca: Exploiting phase-modulated emanations in side channels. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):392–419, Dec. 2024.

[ALAM19] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE symposium on security and privacy (sp)*, pages 1362–1380. IEEE, 2019.

[APSQ06] Cédric Archambeau, Eric Peeters, F-X Standaert, and J-J Quisquater. Template attacks in principal subspaces. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–14. Springer, 2006.

[AT] AMD Xilinx and Jim Tatsukawa. *MMCM and PLL Dynamic Reconfiguration*, XAPP888, 2019.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, feb 2012.

[BBB+21] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. Sok: Computer-aided cryptography. In *2021 IEEE symposium on security and privacy (SP)*, pages 777–795. IEEE, 2021.

[BBG+01] Arthur Beckers, Josep Balasch, Benedikt Gierlichs, Ingrid Verbauwhede, FX Standaert, and E Oswald. Design and implementation of a waveform-matching based triggering system, 2016-01-01.

[BBIP21] Alessandro Barenghi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. Exploring Cortex-M Microarchitectural Side Channel Information Leakage. *IEEE Access*, 9:156507–156527, 2021.

[BBYS22] Ileana Buhan, Lejla Batina, Yuval Yarom, and Patrick Schaumont. Sok: Design tools for side-channel-aware implementations. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 756–770, 2022.

[BCD+13] Georg T. Becker, Jim Cooper, Elizabeth K. DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, T. Kouzminov, Andrew J. Leiserson, Mark E. Marson, Pankaj Rohatgi, and Sami Saab. Test vector leakage assessment (tvla) methodology in practice. 2013.

[BCH+20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *NDSS 2020-Network and Distributed System Security Symposium*, pages 1–14, 2020.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, pages 16–29. Springer, 2004.

[BFP22] Alessandro Barenghi, Gioele Falcetti, and Gerardo Pelosi. Locating side channel leakage in time through matched filters. *Cryptography*, 6(2), 2022.

[BFPZ20] Alessandro Barenghi, William Fornaciari, Gerardo Pelosi, and Davide Zoni. Scramble suit: A profile differentiation countermeasure to prevent template

attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(9):1778–1791, 2020.

[BIK+24] Elie Bursztein, Luca Invernizzi, Karel Král, Daniel Moghimi, Jean-Michel Picod, and Marina Zhang. Generalized power attacks against crypto hardware using long-range deep learning. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(3):472–499, Jul. 2024.

[BIKP19] Elie Bursztein, Luca Invernizzi, Karel Král, and Jean-Michel Picod. SCAAML: Side Channel Attacks Assisted with Machine Learning, 2019.

[BJP20] Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. AES HD dataset - 500 000 traces. AISyLab repository, 2020. `https://github.com/AISyLab/AES_HD_2`.

[BLH+25] Fabian Buschkowski, Georg Land, Niklas Höher, Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. Hades: Automated hardware design exploration for cryptographic primitives. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):1–45, Sep. 2025.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ascad database. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020.

[BUS21] Davide Bellizia, Balazs Udvarhelyi, and François-Xavier Standaert. Towards a better understanding of side-channel analysis measurements setups. In *International Conference on Smart Card Research and Advanced Applications*, pages 64–79. Springer, 2021.

[CBT20] Lukasz Chmielewski, Ileana Buhan, and Karim Tobich. Reassure (h2020 731591) masked aes dataset, March 2020.

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In *Cryptographic Hardware and Embedded Systems—CHES 2000: Second International Workshop Worcester, MA, USA, August 17–18, 2000 Proceedings 2*, pages 252–263. Springer, 2000.

[CDGT23] Claude Carlet, Abderrahman Daif, Sylvain Guilley, and Cédric Tavernier. Quasi-linear masking against sca and fia, with cost amortization. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(1):398–432, Dec. 2023.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures: Profiling attacks without pre-processing. In *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 45–68. Springer, 2017.

[CFG+20] Luca Cremona, William Fornaciari, Andrea Galimberti, Andrea Romanoni, and Davide Zoni. Vgm-bench: Fpu benchmark suite for computer vision, computer graphics and machine learning applications. In Alex Orailoglu, Matthias Jung, and Marc Reichenbach, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 323–335, Cham, 2020. Springer International Publishing.

[CGL+24] Giuseppe Chiari, Davide Galli, Francesco Lattari, Matteo Matteucci, and Davide Zoni. A deep-learning technique to locate cryptographic operations in side-channel traces. In *2024 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6, 2024.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 156–170. Springer, 2009.

[CM24] Gaëtan Cassiers and Charles Momin. The smaesh dataset. *Cryptology ePrint Archive*, 2024.

[CPW24] Hao Cheng, Daniel Page, and Weijia Wang. eliminate: a leakage-focused ise for masked implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):329–358, Mar. 2024.

[CRR03] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic hardware and embedded systems-CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*, pages 13–28. Springer, 2003.

[CRZ18] Jean-Sébastien Coron, Franck Rondepierre, and Rina Zeitoun. High order masking of look-up tables with common shares. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):40–72, Feb. 2018.

[CS19] Michael Cooper and Kim Schaffer. Security requirements for cryptographic modules, 2019-03-22 2019.

[DCEM18] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware masking, revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(2):123–148, May 2018.

[DGC+24] Lev Denisov, Andrea Galimberti, Daniele Cattaneo, Giovanni Agosta, and Davide Zoni. Design-time methodology for optimizing mixed-precision CPU architectures on FPGA. *Journal of Systems Architecture*, 155:103257, 2024.

[DHL+21] Ba-Anh Dao, Trong-Thuc Hoang, Anh-Tien Le, Akira Tsukamoto, Kuniyasu Suzaki, and Cong-Kha Pham. Correlation power analysis attack resisted cryptographic risc-v soc with random dynamic frequency scaling countermeasure. *IEEE Access*, 9:151993–152014, 2021.

[dlSdSd21] Agence Nationale de la Sécurité des Systèmes d'Information. Ascadv2, 2021. `https://www.data.gouv.fr/en/datasets/ascadv2/`.

[DRS+13] François Durvaux, Mathieu Renauld, François-Xavier Standaert, Loïc Van Oldeneel Tot Oldenzeel, and Nicolas Veyrat-Charvillon. Efficient removal of random delays from embedded software implementations using hidden markov models. In *International Conference on Smart Card Research and Advanced Applications*, pages 123–140. Springer, 2013.

[Dwo15] Morris J. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions:, 2015-07-01 04:07:00 2015.

[FI16] Ibraheem Frieslaar and Barry Irwin. Investigating multi-thread utilization as a software defence mechanism against side channel attacks. In *Proceedings of the 8th International Conference on Signal Processing Systems*, ICSPS 2016, page 189–193, New York, NY, USA, 2016. Association for Computing Machinery.

[Fis38] Ronald A Fisher. The statistical utilization of multiple measurements. *Annals of eugenics*, 8(4):376–386, 1938.

[FRBSG24] Jakob Feldtkeller, Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. Combined threshold implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(4):307–334, Sep. 2024.

[Fre24] FreeRTOS.org. Freertos. `https://github.com/FreeRTOS/FreeRTOS`, 2024. Available under MIT License.

[Fri12] Austin W Fritzke. Obfuscating against side-channel power analysis using hiding techniques for aes. 2012.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual Information Analysis - A Generic Side-Channel Distinguisher. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442, Washington DC,US, 2008. Springer-Verlag.

[GCZ24] Davide Galli, Giuseppe Chiari, and Davide Zoni. Hound: Locating cryptographic primitives in desynchronized side-channel traces using deep-learning. In *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, pages 114–121, 2024.

[GCZ25] Davide Galli, Giuseppe Chiari, and Davide Zoni. Chameleon: A dataset for segmenting and attacking obfuscated power traces in side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(3):389–412, Jun. 2025.

[GD24] John Gaspoz and Siemen Dhooghe. Bit t-sni secure multiplication gadget for inner product masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):104–127, Dec. 2024.

[GGF+24] Davide Galli, Adriano Guarisco, William Fornaciari, Matteo Matteucci, and Davide Zoni. The impact of run-time variability on side-channel attacks targeting fpgas. In *2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4, 2024.

[GGFZ22] Davide Galli, Andrea Galimberti, William Fornaciari, and Davide Zoni. On the effectiveness of true random number generators implemented on fpgas. In Alex Orailoglu, Marc Reichenbach, and Matthias Jung, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 315–326, Cham, 2022. Springer International Publishing.

[GLBG23] Anna Guinet, Georg Land, Ioan Gabriel Bucur, and Tim Güneysu. A tale of snakes and horses: Amplifying correlation power analysis on quadratic maps. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(1):27–50, Dec. 2023.

[GLMZ25] Davide Galli, Francesco Lattari, Matteo Matteucci, and Davide Zoni. A deep learning-assisted template attack against dynamic frequency scaling countermeasures. *IEEE Transactions on Computers*, 74(1):293–306, 2025.

[GM11] Tim Güneysu and Amir Moradi. Generic side-channel countermeasures for reconfigurable devices. In *Cryptographic Hardware and Embedded Systems–*

CHES 2011: 13th International Workshop, Nara, Japan, September 28–October 1, 2011. Proceedings 13, pages 33–48. Springer, 2011.

[GMK16] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS '16, page 3, New York, NY, USA, 2016. Association for Computing Machinery.

[GMZ25] Davide Galli, Matteo Matteucci, and Davide Zoni. Rabbit: Dynamic clock randomization to protect against side-channel attacks. In *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2025.

[GP99] Louis Goubin and Jacques Patarin. Des and differential power analysis the "duplication" method. In *Cryptographic Hardware and Embedded Systems: First InternationalWorkshop, CHES'99 Worcester, MA, USA, August 12–13, 1999 Proceedings 1*, pages 158–172. Springer, 1999.

[GR24] Morgane Guerreau and Mélissa Rossi. A not so discrete sampler: Power analysis attacks on hawk signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(4):156–178, Sep. 2024.

[HDL+20] Benjamin Hettwer, Kallyan Das, Sebastien Leger, Stefan Gehrer, and Tim Güneysu. Lightweight side-channel protection using dynamic clock randomization. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 200–207. IEEE, 2020.

[HGC+24] Xiaoran Huang, Yiwen Gao, Wei Cheng, Yuejun Liu, Jingdian Ming, Yongbin Zhou, and Jian Weng. Towards high-quality electromagnetic leakage acquisition in side-channel analysis. In *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 370–377, 2024.

[HGG19] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In *International conference on selected areas in cryptography*, pages 645–666. Springer, 2019.

[HMH+12] Johann Heyszl, Stefan Mangard, Benedikt Heinz, Frederic Stumpf, and Georg Sigl. Localized electromagnetic analysis of cryptographic implementations. In *Topics in Cryptology–CT-RSA 2012: The Cryptographers' Track at the*

RSA Conference 2012, San Francisco, CA, USA, February 27–March 2, 2012. Proceedings, pages 231–244. Springer, 2012.

[HPGM17] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Side-channel analysis of lightweight ciphers: Does lightweight equal easy? Cryptology ePrint Archive, Paper 2017/261, 2017. `https://eprint.iacr.org/2017/261`.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2012.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[Inr15] Inrevium Inc. Sasebo-gii-32, 2015.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[JIP19a] Darshana Jayasinghe, Aleksandar Ignjatovic, and Sri Parameswaran. Rftc: Runtime frequency tuning countermeasure using fpga dynamic reconfiguration to mitigate power analysis attacks. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[JIP19b] Darshana Jayasinghe, Aleksandar Ignjatovic, and Sri Parameswaran. Scrip: Secure random clock execution on soft processor systems to mitigate power-based side channel attacks. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2019.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[Key20] Keysight Technologies Inc. DS1002A Pattern Based Trigger Generator. `https://www.keysight.com/us/en/product/DS1002A/`

`pattern-based-trigger-generator.html`, 2020. Accessed: 2025-07-04.

[KHF+19] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[Kiz20] Ilya Kizhvatov. AES RD dataset, 2020. `https://github.com/ikizhvatov/randomdelays-traces`.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.

[KMMS21] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated generation of masked hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):589–629, Nov. 2021.

[Koc96] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, pages 104–113. Springer, 1996.

[KPP24] Sengim Karayalcin, Guilherme Perin, and Stjepan Picek. Diffuse some noise: Diffusion models for measurement noise removal in side-channel analysis. *Cryptology ePrint Archive*, 2024.

[LKMM21] Oleksiy Lisovets, David Knichel, Thorben Moos, and Amir Moradi. Let's take it offline: Boosting brute-force attacks on iphone's user authentication through sca. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):496–519, Jul. 2021.

[LMDM25] Daniel Lammers, Nicolai Müller, Siemen Dhooghe, and Amir Moradi. Constant-cycle hardware private circuits. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):172–214, Sep. 2025.

[LOM08] Yingxi Lu, Maire P O'Neill, and John V McCanny. Fpga implementation and analysis of random delay insertion countermeasure against dpa. In *2008*

International Conference on Field-Programmable Technology, pages 201–208. IEEE, 2008.

[Low24] LowRISC. ot-sca - Side-Channel Analysis and Fault Injection Setup for OpenTitan. `https://github.com/lowRISC/ot-sca`, 2024. Accessed: 2024-11-05.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[LZC+21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):235–274, Jul. 2021.

[Man04] Stefan Mangard. Hardware countermeasures against dpa–a statistical analysis of their effectiveness. In *Cryptographers' Track at the RSA Conference*, pages 222–235. Springer, 2004.

[MEL20] MELITY. Masked aes implementation. `https://github.com/CENSUS/masked-aes-c`, 2020.

[MMH+25] Dev Mehta, Trey Marcantino, Mohammad Hashemi, Sam Karkache, Dillibabu Shanmugam, Patrick Schaumont, and Fatemeh Ganji. Scapegoat: Side-channel analysis library. In *2025 IEEE 43rd VLSI Test Symposium (VTS)*, pages 1–7. IEEE, 2025.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[MS23] Loïc Masure and Rémi Strullu. Side-channel analysis against anssi's protected aes implementation on arm: end-to-end attacks with multi-task learning. *Journal of Cryptographic Engineering*, 13(2):129–147, 2023.

[Nat22] National Institute of Standards and Technology (NIST) - U.S. Department of Commerce. NISTIR 8413, Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. `https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf`, 2022.

[New18] NewAE Technology Inc. Cw305 artix fpga target. `https://rtfm.newae.com/Targets/CW305%20Artix%20FPGA/`, 2018. Accessed: 2025-07-04.

[New20a] NewAE Technology Inc. Chipwhisperer pro. `https://chipwhisperer.readthedocs.io/en/latest/Capture/ChipWhisperer-Pro.html`, 2020. Accessed: 2025-07-15.

[New20b] NewAE Technology Inc. CW308T-FE310. `https://rtfm.newae.com/Targets/UFO%20Targets/CW308T-FE310-G002`, 2020. Accessed: 2025-07-04.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[NHI+07] Sei Nagashima, Naofumi Homma, Yuichi Imai, Takafumi Aoki, and Akashi Satoh. Dpa using phase-based waveform matching against random-delay countermeasure. In *2007 IEEE International Symposium on Circuits and Systems*, pages 1807–1810. IEEE, 2007.

[OP11] David Oswald and Christof Paar. Breaking mifare desfire mf3icd40: Power analysis and templates in the real world. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 207–222. Springer, 2011.

[oSND+23] National Institute of Standards, Technology (NIST), Morris J. Dworkin, Meltem Sonmez Turan, and Nicky Mouha. Advanced encryption standard (aes), 2023-05-09 04:05:00 2023.

[OST06] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Cryptographers' track at the RSA conference*, pages 1–20. Springer, 2006.

[PHJ+18] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):209–237, 2018.

[Pic21] Pico Technology Inc. PicoScope® 5000D Series Data Sheet. `https://www.picotech.com/download/datasheets/picoscope-5000d-series-data-sheet.pdf`, 2021. Accessed: 2025-07-04.

[PPM+23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Computing Surveys*, 55(11):1–35, 2023.

[PWP21] Guilherme Perin, Lichao Wu, and Stjepan Picek. AISY - Deep Learning-based Framework for Side-Channel Analysis. Cryptology ePrint Archive, Report 2021/357, 2021. `https://eprint.iacr.org/2021/357`.

[PWP22] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):828–861, 2022.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *Smart Card Programming and Security: International Conference on Research in Smart Cards, E-smart 2001 Cannes, France, September 19–21, 2001 Proceedings*, pages 200–210. Springer, 2001.

[QWY+24] Shipei Qu, Yuxuan Wang, Jintong Yu, Chi Zhang, and Dawu Gu. Trace copilot: Automatically locating cryptographic operations in side-channel traces by firmware binary instrumenting. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):128–159, Dec. 2024.

[Ras23] Raspberry Pi Foundation. Raspberry pi documentation, 2023. Accessed: 2024-05-20.

[RBBC18] Prasanna Ravi, Shivam Bhasin, Jakub Breier, and Anupam Chattopadhyay. Ppap and ippap: Pll-based protection against physical attacks. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 620–625. IEEE, 2018.

[RD08] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[rg14] TELECOM ParisTech SEN research group. Dpa contest (4th edition), 2013-1014. `https://dpacontest.telecom-paris.fr/v4/index.php/`.

[RSA78] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RSN+01] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert,

et al. *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, volume 22. US Department of Commerce, Technology Administration, National Institute of . . . , 2001.

[RSVC+11] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 109–128. Springer, 2011.

[RWPP21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):677–707, Jul. 2021. Available under MIT License.

[SBM21] Aein Rezaei Shahmirzadi, Dušan Božilov, and Amir Moradi. New first-order secure aes performance records. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):304–327, Feb. 2021.

[SCD+17] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017.

[SKP+24] Ioana Savu, Marina Krček, Guilherme Perin, Lichao Wu, and Stjepan Picek. The need for more: Unsupervised side-channel analysis with single network training and multi-output regression. In *Constructive Side-Channel Analysis and Secure Design: 15th International Workshop, COSADE 2024, Gardanne, France, April 9–10, 2024, Proceedings*, page 113–132, Berlin, Heidelberg, 2024. Springer-Verlag.

[SM23] Marvin Staib and Amir Moradi. Deep learning side-channel collision attack. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):422–444, Jun. 2023.

[SRH16] Sami Saab, Pankaj Rohatgi, and Craig Hampel. Side-channel protections for cryptographic instruction set extensions. *Cryptology ePrint Archive*, 2016.

[SZ19] Giovanni Scotti and Davide Zoni. A fresh view on the microarchitectural design of fpga-based risc cpus in the iot era. *Journal of Low Power Electronics and Applications*, 9(1), 2019.

[TBW+21] Jens Trautmann, Arthur Beckers, Lennert Wouters, Stefan Wildermann, Ingrid Verbauwhede, and Jürgen Teich. Semi-automatic locating of cryptographic operations in side-channel traces. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):345–366, Nov. 2021.

[TCG+25] Qi Tian, Hao Cheng, Chun Guo, Daniel Page, Meiqin Wang, and Weijia Wang. A code-based ise to protect boolean masking in software. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2):293–332, Mar. 2025.

[Tcl23] Tls/ssl and crypto library. Openssl. `https://github.com/openSSL/openSSL`, 2023. Available under Apache License 2.0.

[UEF22] UEFI Forum Inc. Advanced configuration and power interface (acpi) specification, 2022. Accessed: 2024-05-20.

[VCGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *ASIACRYPT 2014*, volume LNCS 8874 of *ASIACRYPT 2014*, pages 282 – 296, Kaoshiung, Taiwan, December 2014. Palash Sarkar, Tetsu Iwata, Springer.

[vdVPB20] Daan van der Valk, Stjepan Picek, and Shivam Bhasin. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 175–199. Springer, 2020.

[VTM21] Aurélien Vasselle, Hugues Thiebeauld, and Philippe Maurine. Spatial dependency analysis to extract information from side-channel mixtures. In *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, pages 73–84, 2021.

[vWWB11] Jasper GJ van Woudenberg, Marc F Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In *Cryptographers' Track at the RSA Conference*, pages 104–119. Springer, 2011.

[WEG87] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37–52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

[WFW+24] Lixuan Wu, Yanhong Fan, Weijia Wang, Bart Preneel, and Meiqin Wang. Extending randomness-free first-order masking schemes and applications to

masking-friendly s-boxes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):340–366, Dec. 2024.

[WP20] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 389–415, 2020.

[WPP22] Lichao Wu, Guilherme Perin, and Stjepan Picek. The best of two worlds: Deep learning-assisted template attack. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):413–437, Jun. 2022.

[WPP24] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. *IEEE Transactions on Emerging Topics in Computing*, 12(2):546–557, 2024.

[WYO17] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[Xil20] AMD Xilinx. DS180: 7 Series FPGAs Data Sheet: Overview. `https://docs.amd.com/v/u/en-US/ds180_7Series_Overview`, 2020. Access: 2025-07-04.

[YF14] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, l3 cache Side-Channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, San Diego, CA, aug 2014. USENIX Association.

[YL06] T. Ylonen and C. Lonvick. The secure shell (ssh) protocol architecture. `https://www.rfc-editor.org/rfc/rfc4251`, 2006. Request for Comments (RFC) 4251.

[YWV+05] Shengqi Yang, W. Wolf, N. Vijaykrishnan, D.N. Serpanos, and Yuan Xie. Power attack resistant cryptosystem design: a dynamic voltage and frequency switching approach. In *Design, Automation and Test in Europe*, pages 64–69 Vol. 3, 2005.

[ZBHV20] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 1–36, 2020.

[ZG22] Davide Zoni and Andrea Galimberti. Cost-effective fixed-point hardware support for RISC-V embedded systems. *Journal of Systems Architecture*, 126:102476, 2022.

[ZGF21] Davide Zoni, Andrea Galimberti, and William Fornaciari. An FPU design template to optimize the accuracy-efficiency-area trade-off. *Sustainable Computing: Informatics and Systems*, 29:100450, 2021.

[ZGG25] Davide Zoni, Andrea Galimberti, and Davide Galli. An fpga-based open-source hardware-software framework for side-channel security research. *IEEE Transactions on Computers*, 74(06):2087–2100, 2025.

List of Publications

This appendix lists in detail the scientific publications resulting from the research carried out during the PhD period, whose content was presented in the previous chapters.

- **Title:** Chameleon: A Dataset for Segmenting and Attacking Obfuscated Power Traces in Side-Channel Analysis.
  Authors: Davide Galli, Giuseppe Chiari, Davide Zoni.
  Journal: IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES).
  Year: 2025.
  Volume: 2025 (issue 3).
  Pages: 389–412.
  DOI: 10.46586/tches.v2025.i3.389-412.
  Copyright: © 2025 IACR
  Cited as: [GCZ25].
  Personal contribution: Ground-truth labelling architectures, dataset validation, experimental benchmarking and manuscript preparation.

- **Title:** Rabbit: Dynamic Clock Randomization to Protect against Side-Channel Attacks.
  Authors: Davide Galli, Matteo Matteucci, Davide Zoni.
  Conference: 2025 IEEE International Symposium on Circuits and Systems (ISCAS).
  Year: 2025.
  Pages: 1–5.
  DOI: 10.1109/ISCAS56072.2025.11043661.
  Copyright: © 2025 IEEE
  Cited as: [GMZ25].
  Personal contribution: DFS randomization, implementation and experimental evaluation, manuscript preparation.

- **Title:** An FPGA-Based Open-Source Hardware-Software Framework for Side-Channel

Security Research.
Authors: Davide Zoni, Andrea Galimberti, and Davide Galli.
Journal: IEEE Transactions on Computers.
Year: 2025.
Volume: 74.
Pages: 2087–2100.
DOI: 110.1109/TC.2025.3551936.

Cited as: [ZGG25].
Personal contribution: Side-channel countermeasures implementation, side-channel experimental evaluation, manuscript preparation.

- **Title:** A Deep Learning-Assisted Template Attack Against Dynamic Frequency Scaling Countermeasures.
  Authors: Davide Galli, Francesco Lattari, Matteo Matteucci, Davide Zoni.
  Journal: IEEE Transactions on Computers.
  Year: 2025.
  Volume: 74.
  Number: 1.
  Pages: 293–306.
  DOI: 10.1109/TC.2024.3477997.

  Cited as: [GLMZ25].
  Personal contribution: Dataset preparation and validation, side-channel attack validation and manuscript preparation.

- **Title:** Hound: Locating Cryptographic Primitives in Desynchronized Side-Channel Traces using Deep-Learning.
  Authors: Davide Galli, Giuseppe Chiari, Davide Zoni.
  Conference: 2024 IEEE 42nd International Conference on Computer Design (ICCD).
  Year: 2024.
  Pages: 114–121.
  DOI: 10.1109/ICCD63220.2024.00027.

  Cited as: [GCZ24].
  Personal contribution: Model design and validation, dataset preparation, experiments and manuscript preparation.

- **Title:** The Impact of Run-Time Variability on Side-Channel Attacks Targeting FPGAs.
  Authors: Davide Galli, Adriano Guarisco, William Fornaciari, Matteo Matteucci, Davide Zoni.
  Conference: 2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS).
  Year: 2024.
  Pages: 1–4.
  DOI: 10.1109/ICECS61496.2024.10848741.
  Copyright: © 2024 IEEE
  Cited as: [GGF+24].
  Personal contribution: Experimental campaign, data analysis and manuscript preparation.

- **Title:** A Deep-Learning Technique to Locate Cryptographic Operations in Side-Channel Traces.
  Authors: Giuseppe Chiari, Davide Galli, Francesco Lattari, Matteo Matteucci, and Davide Zoni.
  Conference: 2024 Design, Automation and Test in Europe Conference and Exhibition (DATE).
  Year: 2024.
  Pages: 1–6.
  DOI: 10.23919/DATE58400.2024.10546758.
  Copyright: © 2024 IEEE
  Cited as: [CGL+24].
  Personal contribution: Side-channel attack validation and manuscript preparation.

List of Figures

List of Tables