Analysis and Mitigation of Side-Channel Vulnerabilities in FPGA Based AES Design

A

report submitted in fulfillment for the award of the degree of

Bachelors of Technology

in

Information Technology

Bachelors of Technology Project

Final Evaluation Report

By

Karamthot Venkatesh: 2022IMG-028

Sushant: 2022IMG-062

Vankdoth Sahithi Sree: 2022IMG-065

Under the Supervision of

Dr. Vijaypal Singh Rathor

Department of Information Technology

ABV-INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT GWALIOR

DECLARATION

We, Karamthot Venkatesh, Sushant, Vankdoth Sahithi Sree, students of IMG, bearing Institute Roll No. 2022IMG- 028, 2022IMG-062, 2022IMG-065 of ABV-IIITM, Gwalior during the session 2022-2027, do hereby solemnly declare that we have fully completed my B.Tech Project from May 2025 to July 2025. We have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, and websites, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

Dated: **Signature of the candidates**

This is to certify that the above statement made by the candidates is correct to the best of our knowledge.

Dated: **Signature of supervisor**

Acknowledgements

We are highly obliged to **Dr. Vijaypal Singh Rathor**, for bestowing upon us the opportunity to work under him as student researchers and for immeasurable support and encouragement. Dr. Vijaypal encouraged us to look up and work on unique topics and also helped solve queries. He was always approachable.

We also extend our sincere gratitude to our friends and family who have been really supportive of me during the duration of the project.

We are also highly indebted to our institution for affording us the opportunity to embark on this project. This project allowed us to dive deep into the concepts of navigating the complexities of Side-channel Analysis and Hardware security and explore a field that was very new to me.

Karamthot Venkatesh [2022IMG-028]

Sushant [2022IMG-062]

Vankdoth Sahithi Sree [2022IMG-065]

Abstract

This research presents a comprehensive analysis of side-channel vulnerabilities in FPGA-based AES cryptographic implementations and demonstrates the development of effective hardware-level countermeasures. Using the ChipWhisperer Husky platform and CW305 target FPGA, we systematically evaluate the susceptibility of unprotected AES-128 cores to Correlation Power Analysis (CPA) attacks, successfully recovering complete secret keys from power consumption traces.

Our experimental methodology involves capturing 5,000 power traces from an unprotected baseline AES core, applying statistical correlation analysis to exploit data-dependent power variations during S-Box operations, and achieving complete key recovery with correlations exceeding 0.188. The baseline implementation demonstrates critical vulnerabilities where attackers can extract secret keys using as few as 800-1500 traces, highlighting the practical security risks in standard cryptographic hardware deployments.

To address these vulnerabilities, we design and implement a secure Verilog-based AES-128 core incorporating first-order Boolean masking countermeasures. The protected implementation randomizes intermediate values by XORing sensitive data with random masks, effectively decorrelating power consumption from actual cryptographic operations. Validation experiments confirm that the masking countermeasure successfully defeats first-order CPA attacks, with the protected core recovering only 1 out of 16 key bytes (consistent with random chance) compared to complete key extraction from the unprotected version.

The research contributes a practical framework for developing side-channel resistant cryptographic hardware, validated through rigorous power analysis using industry-standard tools. While the implemented first-order masking provides robust protection against CPA attacks, the work acknowledges theoretical vulnerabilities to higher-order attacks and identifies future research directions including advanced masking schemes, true random number generation, and multi-channel attack resistance. This work establishes foundational guidelines for secure hardware implementation in embedded systems and IoT devices where physical security is paramount.

Contents

List of Figures

List of Tables

1

Introduction

This research introduces the field of Side-Channel Analysis (SCA) and demonstrates the application of Correlation Power Analysis (CPA) on cryptographic systems. Specifically, it focuses on the AES encryption algorithm implemented on FPGAs and examines how power consumption can be exploited to extract secret keys. Furthermore, this work highlights the motivation for studying hardware-level security and addresses the challenge of designing and validating effective RTL countermeasures—such as masking—to mitigate these physical attacks.

1.1 Context

This report presents the research work carried out as part of our B.Tech Project, focused on the Advanced Encryption Standard (AES) and its hardware-level vulnerabilities. The project gave us hands-on experience in hardware security and FPGA-based AES implementations, showing how even a mathematically secure algorithm can leak information through its physical execution.

AES has been the global encryption standard and is widely used in communication, finance, and defense. While secure in theory, real-world implementations face threats from Side-Channel Analysis (SCA), which exploits leakages such as power consumption and timing variations [1, 2]. By measuring these leakages, attackers can recover secret keys without breaking the algorithm.

FPGAs are popular for AES due to their flexibility and performance but are highly susceptible to such leakages [3]. In our work, we implemented AES-128 on a CW305 FPGA and used the ChipWhisperer Husky to perform Correlation Power Analysis (CPA). With thousands of power traces, we successfully recovered the AES key, confirming the vulnerability of unprotected designs [4].

To counter this, we developed a masked AES core in Verilog. Masking randomizes intermediate values to break the correlation between power and data. Experiments showed that while the unprotected AES leaked enough to recover the key, the masked design significantly reduced leakage, making CPA ineffective.

Finally, we note that AES, though widely adopted, is not always practical for highly resource-constrained devices like RFID tags and tiny IoT sensors, where lightweight ciphers are better alternatives.

The major contributions of this project can be summarized as follows:

- A focused study and practical demonstration of side-channel vulnerabilities in FPGA-based AES implementations.
- Experimental validation of key recovery from an unprotected AES-128 core on a

CW305 FPGA using Correlation Power Analysis (CPA).

- The design and implementation of a masking-based AES core in Verilog as a countermeasure against side-channel attacks.
- A comparative evaluation of the unprotected and masked AES designs, using power traces and statistical correlation metrics to quantify the effectiveness of masking.
- A recognition of AES's limitations in ultra-constrained devices, motivating future exploration of lightweight cryptography.

1.2 Motivation

Cryptographic Security in Embedded Systems: The mathematical security of cryptographic algorithms such as AES is well-established, but their physical implementation often introduces serious vulnerabilities. In a world increasingly reliant on embedded systems and IoT devices, physical attacks like power analysis pose a significant threat. Successful demonstrations of such attacks on smart cards and microcontrollers have shown how easily sensitive information can be compromised [5]. This highlights the importance of secure RTL design practices that explicitly consider physical leakage alongside functionality and performance.

Market for Secure Hardware: With the growing adoption of IoT and smart devices, the demand for secure hardware has never been greater. While software-level protections have become relatively mature, hardware-level security against physical attacks is still an area with no universal methodology or standardized design flow. Our project contributes to this field by demonstrating a practical masking-based AES implementation on FPGA that raises the difficulty of side-channel key recovery and serves as a step toward more robust hardware security solutions.

1.3 Side-Channel Attacks and Hardware Security

1.3.1 Definition and Uses

Side-channel attacks (SCA) exploit physical leakages from a cryptographic device—such as variations in power consumption, electromagnetic radiation, or execution time—to extract secret information. Unlike traditional cryptanalysis, which targets mathematical weaknesses in the algorithm, SCA focuses on the physical implementation.

This field has significant implications for the security of smart cards, embedded systems, military hardware, and IoT devices. The key motivation behind our work is the understanding that a cryptographically secure algorithm is only as strong as its implementation. Even a mathematically flawless cipher like AES can be compromised if its hardware design leaks information through observable physical channels. Therefore, studying these leakages and developing effective countermeasures is critical for ensuring true end-to-end security in cryptographic hardware.

1.3.2 Nature of Features

In side-channel analysis, temporal features play the most critical role. Temporal features capture the variations in power consumption over time, reflecting the sequence of internal operations performed by the cryptographic device. For example, the power traces corresponding to each round of AES encryption often exhibit distinctive temporal patterns, which can be exploited by statistical techniques such as Correlation Power Analysis (CPA) to recover secret key bytes.

While some advanced attacks may also consider spatial features—such as simultaneous measurements from multiple sensors or pins—our work focuses on temporal analysis of single-point power traces. By carefully analyzing these time-domain features, we are able to link the observed power consumption to intermediate values within the AES computation and extract the secret key.

1.3.3 Methods for Side-Channel Analysis

Among the various approaches for side-channel analysis, Correlation Power Analysis (CPA) has proven to be one of the most effective and widely used techniques. CPA relies on the assumption that the power consumption of a device is related to the data being processed, often modeled using metrics such as Hamming weight or Hamming distance. By capturing multiple power traces during AES operations and statistically correlating them with hypothetical leakage models for different key guesses, the correct key can be identified.

While other methods such as Differential Power Analysis (DPA) and machine learning-based approaches also exist, our work specifically employs CPA due to its effectiveness, practicality, and relatively low trace requirements. This choice allows us to directly demonstrate the vulnerability of an unprotected AES core on FPGA and to evaluate the effectiveness of masking as a countermeasure.

1.3.4 Properties of a Power Trace

(i) **Chronological Fluctuations:** A power trace is a time-ordered sequence of measurements that reflects the progression of a cryptographic operation. This chronological nature makes it possible to isolate specific stages of the AES algorithm, such as SubBytes or MixColumns. Identifying these regions within the trace is essential for performing statistical attacks like CPA, since the correlation between hypothetical values and measured power depends on locating the leakage points accurately.

(ii) **Data-Dependent Variations:** The power consumed by a digital circuit depends on the data being processed, a property that forms the foundation of side-channel attacks. For example, the output of an AES S-box operation has different Hamming weights depending on the input, and these differences manifest as measurable variations in power consumption. CPA exploits this property by comparing predicted leakage values with the actual traces to recover the secret key.

(iii) **Signal-to-Noise Ratio (SNR):** A power trace always contains both useful leakage and noise. Leakage corresponds to the deterministic, data-dependent portion of the signal, while noise arises from measurement inaccuracies, environmental interference, or unrelated switching activity within the FPGA. The effectiveness of an attack strongly depends on the signal-to-noise ratio (SNR). A higher SNR makes correlations clearer, while a low SNR requires collecting more traces to achieve key recovery.

(iv) **Statistical Stability:** The statistical characteristics of power traces, such as their mean and variance, often change across different rounds of AES because each round processes different intermediate values. This instability creates distinguishable patterns that attackers can exploit. One of the goals of countermeasures like masking is to reduce this exploitable variability, ensuring that the traces no longer reveal consistent correlations with the secret key.

1.4 Problem Statement

Developing a secure Verilog implementation of the AES-128 algorithm to mitigate physical leakages against side-channel attacks. We specifically focus on implementing masking as a countermeasure to reduce data-dependent power variations and improve the physical security of the cryptographic design.

2

Literature Review

This section reviews existing research in side-channel analysis (SCA) and hardware-based cryptographic countermeasures. It highlights key milestones such as Correlation Power Analysis (CPA) and masking techniques, which form the foundation for our project's approach.

2.1 Background

2.1.1 Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) is a symmetric key block cipher that operates on 128-bit data blocks with key sizes of 128, 192, or 256 bits. AES follows a Substitution-Permutation Network (SPN) structure, where multiple rounds of substitution, permutation, and mixing transformations are applied to achieve confusion and diffusion [1]. AES-128, the most widely used variant, applies 10 such rounds, ensuring high resistance against conventional cryptanalysis [2].

2.1.2 AES Round Structure

Each AES round (except the final round) consists of four main transformations:

- **SubBytes:** A non-linear substitution step where each byte in the state matrix is replaced using a fixed substitution box (S-box). This introduces non-linearity into the cipher [4].

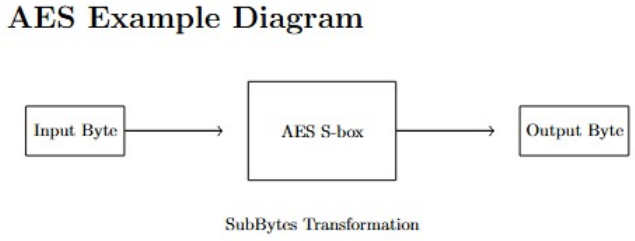


Figure 2.1: A diagram representing AES SubBytes step using the S-box.

- **ShiftRows:** A transposition step where rows of the state are cyclically shifted to the left, providing diffusion across columns.
- **MixColumns:** A mixing operation that applies a linear transformation on each column, ensuring further diffusion between bytes [6].
- **AddRoundKey:** The state is XORed with the round key, which is derived from the key schedule.

2.1.3 Key Scheduling

AES derives round keys from the original secret key using a key expansion algorithm. For AES-128, this generates 11 round keys (one for each of the 10 rounds plus the initial key). The key schedule ensures that fresh key material is used at every round, strengthening resistance against linear and differential attacks [5].

2.1.4 Security in Practice

Although AES is mathematically secure, practical hardware implementations must balance speed, area, and resistance to physical attacks [7]. Optimizations such as pipelining and loop unrolling improve throughput, but they can introduce vulnerabilities to leakage-based attacks [8]. This makes AES a prime target for Side-Channel Analysis (SCA).

2.2 Side-Channel Analysis methods for AES

2.2.1 Overview

Side-channel analysis (SCA) exploits physical leakages such as power consumption, electromagnetic emissions, or timing variations to recover secret information without directly breaking the mathematical security of AES [9]. These attacks reveal weaknesses in real-world implementations, especially when countermeasures are not applied.

2.2.1.1 Correlation Power Analysis (CPA)

CPA is a powerful technique that uses statistical correlation between predicted leakage and measured power traces. Leakage is modeled using Hamming weight or Hamming distance, and Pearson's correlation is applied across many traces. The key hypothesis showing the highest correlation is identified as correct. CPA is efficient, noise-resilient, and requires fewer traces compared to DPA, making it highly effective against AES implementations.

2.2.1.2 Differential Power Analysis (DPA)

DPA is an earlier technique that groups power traces based on guessed key values and compares their averages. Differences in leakage patterns reveal information about the correct key. Although conceptually simpler, DPA is less robust in noisy conditions and typically needs more traces. It introduced the foundation for exploiting statistical biases in side-channel data, upon which advanced methods like CPA are built.

2.2.2 Countermeasures

Masking is one of the most studied countermeasures against first-order CPA [10]. By randomizing intermediate values, masking breaks the correlation between power consumption and secret data. Higher-order masking and combined countermeasures offer additional protection. Studies show that masked implementations require far more traces for successful attacks, and in some cases resist them entirely, making masking a strong candidate for lightweight protection [11].

2.2.3 Platforms and Tools

Research in SCA has been accelerated by platforms such as the open-source ChipWhisperer framework [12]. Hardware like the CW305 FPGA target and ChipWhisperer Husky provide reproducible environments for attacks and defenses. These tools are widely used in both academia and industry to study leakage and validate countermeasures. Our project employs these platforms to demonstrate AES key recovery using CPA and to evaluate masking as a countermeasure [13].

2.3 Key Related Works on Side-Channel Analysis (SCA)

Side-channel analysis exploits physical leakages such as power consumption to reveal information about cryptographic states [5,11]. For AES, SBox operations are particularly vulnerable, as their power usage often correlates with the Hamming weight of processed

data [8].

Classical studies show that Correlation Power Analysis (CPA) is highly effective in recovering AES-128 keys on FPGAs [4, 14]. By modeling leakage with Hamming weight or Hamming distance and correlating with measured traces, attackers can retrieve full keys even without breaking AES mathematically [13].

A practical challenge in such attacks is trace misalignment due to noise or jitter. Preprocessing techniques such as filtering and alignment have been shown to greatly improve CPA efficiency, often reducing the number of traces needed for key recovery [10].

Among countermeasures, masking has emerged as a widely studied and effective defense. Boolean masking randomizes intermediate values, breaking the correlation between power and secret keys, and significantly increases the traces required for a successful attack [7, 15].

Recent works stress the importance of testing countermeasures under realistic experimental conditions [16]. Building on these insights, our project targets AES-128, demonstrating its vulnerability through CPA and validating masking as a lightweight yet effective defense using the CW305 FPGA and ChipWhisperer Husky.

2.4 Research Gaps

While our project demonstrates the feasibility of recovering AES-128 keys using CPA and highlights the effectiveness of masking as a countermeasure, several gaps remain that open directions for further work:

(i) **Extending Beyond First-Order Masking:** Our implementation focuses on first-order Boolean masking. While effective against standard CPA, it remains vulnerable to higher-order analysis. A clear research gap exists in developing lightweight higher-order masking schemes that remain practical for FPGA implementations [9].

(ii) **Comprehensive Attack Models:** This study uses CPA as the primary attack technique. However, side-channel attacks include a broader range of statistical

and machine learning-based approaches that may bypass simple protections. Future research should extend evaluation to these advanced methods to fully validate countermeasure strength [17].

(iii) **Impact of Noise and Environmental Factors:** Our experiments were conducted under controlled laboratory conditions with stable measurement setups. In practice, devices operate under varying environmental noise, voltage fluctuations, and clock instabilities. Further work is needed to analyze the robustness of masking under such real-world conditions [12].

(iv) **Integration with Automated Design Flows:** Our secure AES-128 Verilog model was manually designed and tested. A research gap exists in incorporating security features such as masking directly into RTL design automation tools, which would make secure hardware development more scalable and less error-prone [18].

(v) **Generality Across Platforms:** The evaluation in this project was limited to the CW305 FPGA platform. Future studies should investigate whether the same masking-based countermeasures retain their effectiveness across different FPGA families and hardware architectures, which may exhibit distinct leakage characteristics [19].

2.5 Objectives

The main objectives during the tenure of this project were:

- To study and demonstrate the relationship between a cryptographic device's power consumption and the secret key, through literature review and experimental validation of side-channel attacks.

- To design and implement an AES-128 encryption core in Verilog, incorporating masking as the primary hardware countermeasure.

- To apply Correlation Power Analysis (CPA) on both unprotected and masked AES implementations using the ChipWhisperer Husky and CW305 FPGA platform.
- To evaluate and compare the effectiveness of the protected and unprotected AES designs using statistical measures and side-channel security metrics.

2.6 Novelty

The novelty of our work lies in the way we demonstrate and address side-channel vulnerabilities in cryptographic hardware. The following points summarize the unique aspects of our project:

- We focus specifically on implementing and validating **Correlation Power Analysis (CPA)** on an AES-128 core using the ChipWhisperer CW305 FPGA platform, providing a practical demonstration of side-channel leakage on real hardware.
- Unlike many studies that remain theoretical, our work emphasizes a hands-on, experimental approach, bridging the gap between cryptographic theory and practical hardware security.
- We implement a secure **masked AES-128 Verilog design** and show its effectiveness against CPA, demonstrating how even lightweight countermeasures can significantly increase the difficulty of key recovery.
- The project highlights the critical role of **trace preprocessing and alignment** in successful side-channel attacks, reinforcing the importance of data quality in experimental SCA research.
- By comparing unprotected and masked AES cores on the same FPGA platform, we provide a clear, measurable evaluation of countermeasure effectiveness under controlled conditions.

Table 2.1: A concise portrayal of research literature and future scope derived from this work.

Name of Literature	**Feature**	**Future Scope Derived**
Correlation Power Analysis (CPA) [5]	Exploits statistical correlation between hypothetical leakage (e.g., Hamming weight) and measured traces to recover AES keys.	Extending evaluation to higher-order attacks and machine learning-based side-channel analysis for stronger validation.
Trace Preprocessing and Alignment [11]	Filtering and alignment methods improve the quality of traces, making CPA more effective under noisy or misaligned conditions.	Exploring automated and adaptive preprocessing techniques for real-world, noisy environments.
Boolean Masking [9]	Randomizes sensitive intermediate values, breaking first-order correlation between power consumption and the secret key.	Designing lightweight higher-order masking techniques suitable for FPGA implementations.
Secure RTL Practices [18]	Careful Verilog design helps minimize data-dependent switching activity and power leakage.	Integrating security features directly into automated RTL design flows to simplify secure hardware development.
ChipWhisperer-Based Evaluation [17]	Provides a reproducible hardware platform (CW305 + Husky) for demonstrating both attacks and countermeasures on AES.	Extending experiments across multiple FPGA families to study differences in leakage characteristics.
Comparative Analysis of AES Implementations [19]	Demonstrates the difference in leakage between unprotected and masked AES-128 cores under CPA.	Extending comparison to include multiple countermeasures and quantifying trade-offs between security, area, and performance.

3

Proposed methodology

The complete architecture and proposed methodology are discussed in this section, along with the implementation details. The discussion will revolve around the system architecture used for the analysis, the methodology proposed for attacking cryptographic implementations, and the working process and implementation details of the same.

3.1 System Analysis Framework and Experimental Setup

This section details the hardware and software components required to perform the side-channel analysis experiments discussed.

3.1.1 Hardware and Software Requirements

The primary hardware and software used in this methodology are as follows:

- **Hardware Components:**
  - **ChipWhisperer Husky:** The main instrument for capturing power traces and injecting faults [8].
  - **Target Boards:** FPGA-based targets such as the CW305 and CW312 UFO boards.
  - **Cables and Connectors:** A 20-pin connector for linking the target to the Husky, SMA cables for signal measurement, and USB cables for programming and communication.

We have connected the Chipwhisperer husky with our laptop via usb-c cable. A SMA connector from FPGA's measure to Husky's Pos terminal for Clock and a USB interface from the laptop to the FPGA for programming.

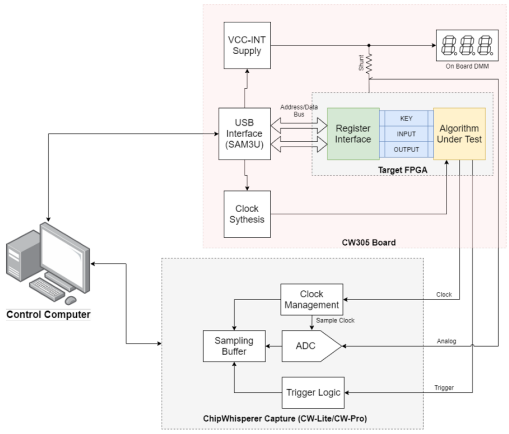


Figure 3.1: This is how a typical setup will look like for our experiments.

- **Software Tools:**
  - **Python Environment:** With libraries such as NumPy for data analysis and Matplotlib for plotting.
  - **ChipWhisperer Software:** The API and Jupyter Notebook environment for controlling the hardware and running attacks.
  - **Vivado Design Suite:** Used for synthesizing the Verilog/VHDL code and generating the bitstream to program the target FPGA.

3.1.2 FPGA Design and Synthesis Workflow

The process of preparing the FPGA for analysis involves a standard digital design flow using the Vivado IDE.

Vivado is an integrated development environment (IDE) provided by Xilinx (now AMD) for designing, simulating, and programming digital systems on their FPGA and SoC devices. It supports RTL design using Verilog or VHDL, allows simulation and debugging, and includes tools for generating bitstreams to program the FPGA hardware.

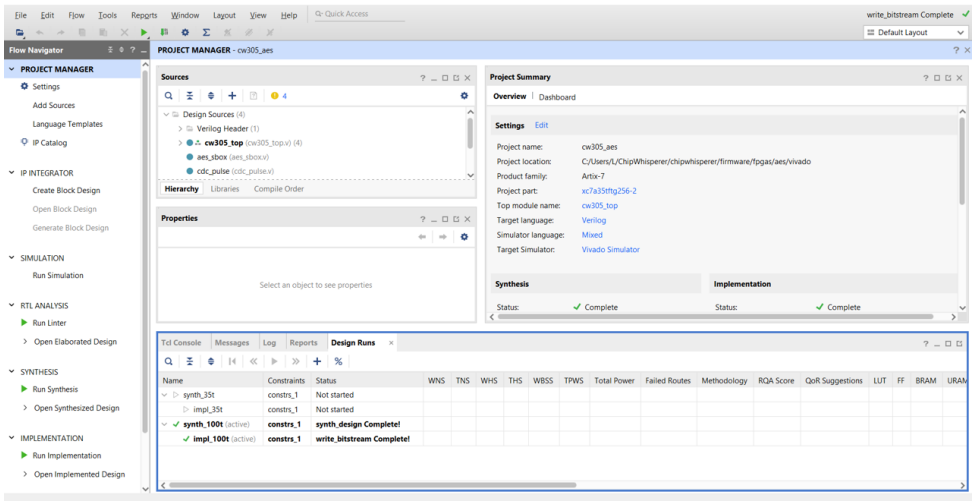


Figure 3.2: Synthesis in Vivado

Running synthesis in Vivado takes your Verilog/VHDL code and converts it into a gate-level netlist that maps your design to the actual logic components (like LUTs, flip-

flops, etc.) available on the target FPGA. It analyzes your code, resolves logic, and optimizes it to fit the FPGA's architecture, preparing it for placement and routing.

(i) **Synthesis:** During synthesis, Verilog or VHDL code is converted into a gate-level netlist, which maps the design to the actual logic components like LUTs and flip-flops available on the target FPGA. This process analyzes the code, resolves the logic, and optimizes it to fit the FPGA's architecture, preparing it for the next stages of placement and routing.

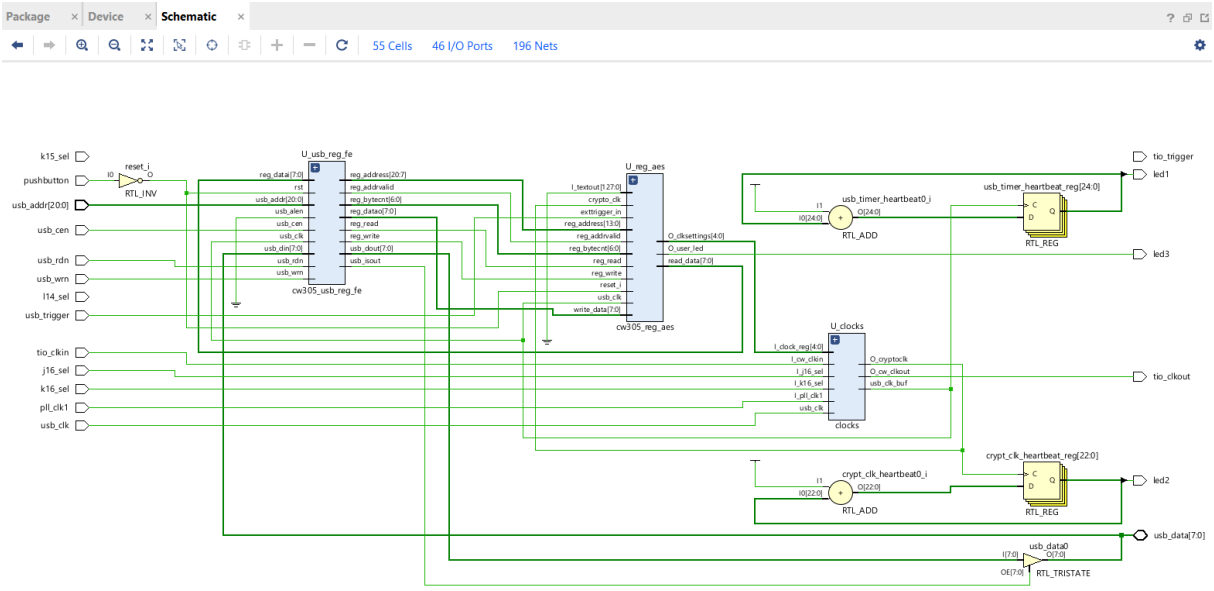


Figure 3.3: FPGA's architecture.

(ii) **Implementation:** The synthesized netlist is placed and routed. This step determines the physical location of each logic gate and the wiring paths that connect them on the FPGA fabric.Verilog is the chosen HDL, and the design is primarily for the CW305 FPGA board. The project's HDL folder contains numerous Verilog (.v) files that define the complete system.

(iii) **Bitstream Generation:** A binary file, known as a bitstream, is generated. This file contains the final configuration data that will be loaded onto the FPGA to

Table 3.1: Description of Verilog HDL Files

Filename	Description
clocks.v	Responsible for the generation of master clock signal
cw305_aes_defines.v	Holds constants and definitions used across AES logic
cw305_reg_aes.v	Manages AES data/control registers for communication
cw305_top.v	This is the top level module that connects everything together
cw305_usb_reg_fe.v	Handles USB Frontend for register communication
SimpleSerial.v	Manages SimpleSerial protocol command parsing
UART.v	Implements UART protocol communication

implement the AES circuit.

(iv) **Flashing the Bitstream:** The generated bitstream is programmed onto the target board (e.g., CW305) using the ChipWhisperer API, making it ready for testing and analysis.

```
TARGET_PLATFORM = 'CW305_100t'
target= cw.target(scope, cw.targets.CW305, bsfile=r"C:\Users\L\ChipWhisperer\chipwhisperer\firmware\fpgas\aes\vivado\cw305_aes.runs\impl_100t\cw305_top.
print("successfully dumped")
```

Figure 3.4: Bitstream is programmed(Screenshot from the software).

3.2 Side-Channel Analysis (SCA) Fundamentals

Side Channel Analysis (SCA) is a class of attacks where an attacker extracts sensitive information (like cryptographic keys) by observing the physical behavior of a device during its operation. These behaviors include power consumption, timing, electromagnetic emissions, and more. Attackers measure side-channel information (e.g., power traces while AES encryption runs) and use statistical methods like Correlation Power Analysis (CPA) or Differential Power Analysis (DPA) to infer the secret key bit-by-bit.

3.2.1 AES as a Target for SCA

While the Advanced Encryption Standard (AES) is mathematically secure, its physical implementations can leak information [11].Advanced Encryption Standard (AES) is a

symmetric block cipher widely used for securing data in embedded systems, communications, and storage. It performs a series of well-defined operations such as SubBytes, ShiftRows, MixColumns, and AddRoundKey on 128-bit blocks of data, using a secret key of 128, 192, or 256 bits The most common target for SCA is the initial round of encryption, specifically the non-linear 'SubBytes' operation performed via an S-Box. The attack typically models the power consumption of this operation. An intermediate value, v, is calculated based on a known plaintext byte and a guessed key byte:

$$v = \text{plaintext\_byte} \oplus \text{key\_guess}$$

The output of the S-Box is then:

$$\text{sbox\_out} = \text{SBOX}[v]$$

The power consumed during the computation of sbox_out is correlated with its Hamming Weight or Hamming Distance. By measuring this power consumption and comparing it against a hypothetical power model for all 256 possible key bytes, an attacker can determine the correct key byte.

3.2.2 Power Leakage in FPGAs

FPGAs are particularly susceptible to power analysis attacks due to their architecture. Power leakage arises from the data-dependent switching activity of the internal digital logic.

- **Data-Dependent Consumption:** The power consumed by logic gates (like LUTs and registers) depends on the data being processed. For instance, transitioning more bits from 0 to 1 generally consumes more power. This creates a relationship between the Hamming weight of the data and the power consumption.
- **Lack of Inherent Countermeasures:** Unlike secure ASICs, FPGAs are general-purpose and lack built-in side-channel resistance unless explicitly designed into the RTL.

- **Exploitable Patterns:** Shared resources and unbalanced routing can create consistent and predictable power patterns that an attacker can exploit to infer secret data.

3.3 Attack Procedure on Unprotected AES Core

The primary objective of this phase is to establish a baseline by demonstrating the vulnerability of the standard hardware AES implementation to side-channel attacks. The procedure is executed in four distinct steps: target preparation, data acquisition, statistical analysis, and key recovery.

3.3.1 Taget Preparation

The initial step in the attack procedure is to prepare the hardware target. This involves configuring a CW305 FPGA with a standard, unprotected AES encryption core, which serves as a baseline to demonstrate side-channel vulnerabilities. The process transforms the high-level hardware design code into a functional circuit on the FPGA.
The cryptographic core is a standard AES-128 implementation provided within the ChipWhisperer framework, described in the Verilog hardware description language (HDL). The design is contained within the Bitstream file, which includes all necessary source files for the AES logic and communication interfaces for the CW305 board.
After successfully flashing the bitstream, the target is fully prepared with the non-hardened AES core and is ready for the power trace acquisition phase of the attack.

3.3.2 Data Acquisition

The power trace acquisition phase is a critical step in which the physical side-channel leakage from the target device is measured and recorded. For hardware targets like FPGAs, this process requires careful consideration because of their inherent characteristics.

(i) **Rationale for High-Volume Data Capture:** A Hardware-based side-channel attacks on FPGAs are notably more challenging than those on microcontrollers.

This is primarily because two reasons: the encryption process is significantly faster, occurring in just a few clock cycles, and there is less observable power leakage because FPGAs do not use memory buses in the same way as CPUs. This results in a much lower signal-to-noise ratio (SNR) in the raw measurements. To overcome this challenge and isolate the faint, key-dependent signal from the background noise, it is necessary to capture a large number of power traces. For this experiment, **5,000 traces** were acquired to ensure that the statistical analysis in the next stage would be effective.

(ii) **Data Acquisition Setup and Process:** The ChipWhisperer Husky serves as the primary instrument for data capture. It is connected to the CW305 target board via a 20-pin ribbon cable for control and an SMA cable for the measurement itself. The SMA cable links the FPGA's dedicated power measurement output to the Husky's high-speed Analog-to-Digital Converter (ADC) input.The acquisition follows an automated, iterative process for each of the 5,000 traces:

- A unique, random plaintext is sent to the FPGA.
- The encryption process is triggered.
- As the FPGA executes the AES algorithm with the fixed secret key, the ChipWhisperer Husky precisely measures and records the device's instantaneous power consumption over the duration of the operation.

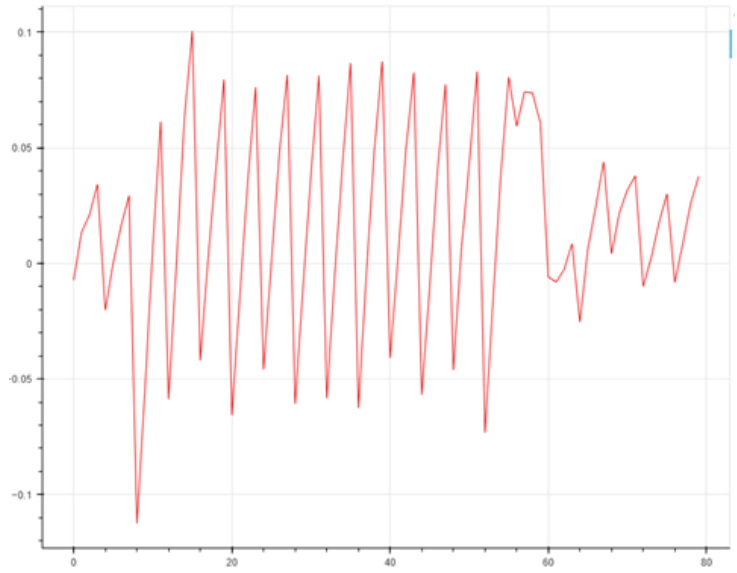


Figure 3.5: Capture Traces.

```
[13]: attack_results = attack.run(cb)
```

Finished traces 4975 to 5000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	D0 0.172	14 0.191	F9 0.193	A8 0.181	C9 0.176	EE 0.145	25 0.187	89 0.180	E1 0.179	3F 0.176	0C 0.217	C8 0.189	B6 0.185	63 0.230	0C 0.166	A6 0.194
1	7D 0.059	97 0.063	AC 0.063	2E 0.060	B6 0.065	0D 0.057	9A 0.058	9E 0.065	78 0.062	A0 0.062	40 0.056	87 0.054	52 0.068	B8 0.057	20 0.067	62 0.063
2	52 0.058	F1 0.054	F4 0.057	AE 0.057	F8 0.057	A0 0.055	EB 0.058	30 0.054	FA 0.061	A9 0.059	FE 0.055	38 0.054	C6 0.062	5A 0.052	75 0.065	C4 0.061
3	16 0.057	B6 0.052	7E 0.054	53 0.055	83 0.057	8E 0.054	96 0.054	2A 0.052	30 0.060	C3 0.057	D3 0.055	5E 0.053	55 0.058	CE 0.052	69 0.058	75 0.050
4	45 0.056	66 0.052	9B 0.051	DE 0.053	85 0.056	2A 0.053	AC 0.053	94 0.052	A2 0.054	0F 0.056	F2 0.052	1E 0.053	2B 0.056	58 0.051	81 0.055	F8 0.050

The attack results can be saved for later viewing or processing without having to repeat the attack:

Figure 3.6: .

Each complete recording is a single "power trace." Once captured, the collection of traces is stored in a NumPy array for efficient processing. Due to the significant time required for this step, the captured data is saved as a project file, allowing for repeated analysis without needing to re-run the lengthy acquisition process.

3.3.3 Correlation Power Analysis (CPA)

Once the power traces are acquired, a Correlation Power Analysis (CPA) is performed to extract the secret key. CPA is a sophisticated side-channel attack that uses statistical

correlation to compare a hypothetical power consumption model with the actual power traces measured from the cryptographic device [14].

- **The Power Model: Hamming Weight**

  CPA is a method where an attacker compares hypothetical power consumption with actual measured power traces from a device like an FPGA running AES. The attack relies on a leakage model, most commonly the Hamming weight, which is the number of '1's in a binary value. The core assumption is that the power consumed by a microcontroller's transistors is directly proportional to the Hamming weight of the data being processed.

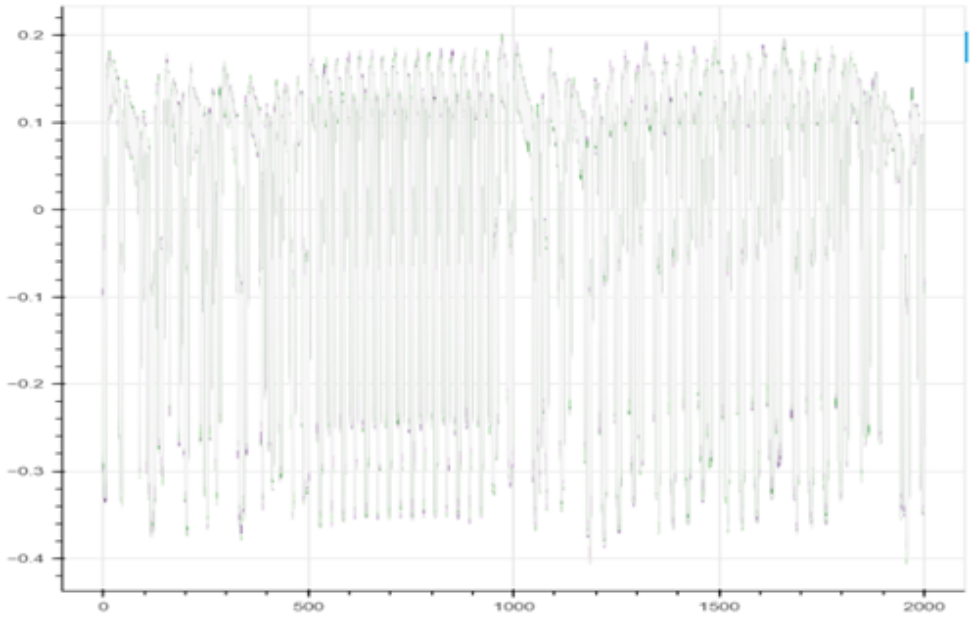


Figure 3.7: Grouping trace by Hamming Weight.

- **Isolating the S-Box Operation**

  In a raw power trace, noise makes it difficult to pinpoint the exact moment of a specific calculation, like the AES S-Box operation. To solve this, a signal processing technique is used.Traces are first grouped by the Hamming weight of the hypothetical S-Box output.The average of all traces is then subtracted from each group.

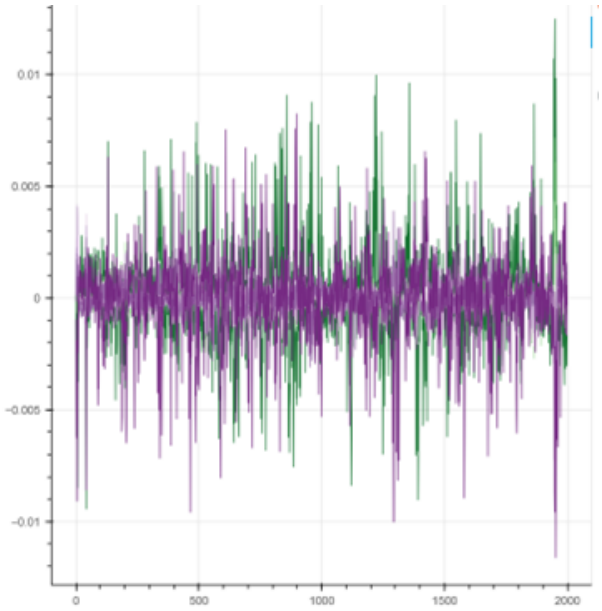


Figure 3.8: Hamming Weight Group Averages.

This procedure effectively removes the overall "shape" of the trace, reducing noise and isolating the leakage related to the S-Box. Plotting the result reveals a clear linear relationship between Hamming weight and the measured voltage, which confirms that a correlation-based attack is viable.

- **The Statistical Attack**

  CPA is highly efficient, requiring far fewer traces are sufficient for correlation. The analysis hinges on calculating the Pearson correlation coefficient, which measures the linear relationship between two datasets.

$$r = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

Figure 3.9: Formula For Correlation

$$cov(X,Y) = \sum_{n=1}^{N}[(Y_n - \bar{Y})(X_n - \bar{X})]$$

Figure 3.10: Formula for Covariance.

$$\sigma_X = \sqrt{\sum_{n=1}^{N}(X_n - \bar{X})^2}$$

Figure 3.11: Formula for Standard Deviation.

The two datasets used for this calculation are:

Hypothetical Data (X): A vector containing the predicted Hamming weight of the S-Box output for each trace, based on a single key guess.

Measured Data (Y): The actual power measurements at a specific point in time across all captured traces.

- We will implement this in our code using numpy. Now, here are the traces we captured:

$$\begin{bmatrix} -0.1057 & -0.3000 & -0.1879 & \cdots & 0.0791 & 0.1030 & 0.1316 \\ -0.1008 & -0.2981 & -0.1887 & \cdots & 0.0784 & 0.1006 & 0.1330 \\ -0.0994 & -0.2761 & -0.1726 & \cdots & 0.0767 & 0.1045 & 0.1316 \\ -0.1091 & -0.2964 & -0.1892 & \cdots & 0.0762 & 0.1016 & 0.1306 \\ -0.1045 & -0.2859 & -0.1794 & \cdots & 0.0781 & 0.1018 & 0.1323 \\ -0.1006 & -0.2766 & -0.1746 & \cdots & 0.0803 & 0.1006 & 0.1279 \end{bmatrix}$$

- where the rows of the array are the different traces we captured and the columns of the array are the different points in those traces. The columns here will be one of the two data sets for our correlation equation. The other data set will be the Hamming weight of the SBox output.

3.3.4 Key Recovery

We are performing a Correlation Power Analysis (CPA) to recover one byte of the AES key. First, we collect power traces during encryption and compute the Hamming weight of a key-dependent intermediate (SBox output) for each trace and key guess (0–255). For each guess, we calculate the correlation between the Hamming weight vector and all points in the power traces. The maximum absolute correlation indicates both the most likely key byte and the time index of leakage. NumPy makes this efficient by allowing vectorized operations like mean() and cov() without manual loops. The key guess with the highest correlation becomes our best estimate. Over 0-255, we will now get the key byte along with the correlation, that looks something like this:

```
Key guess:  0x2b
Correlation:  0.8270952071731353
```

Figure 3.12: Key byte.

3.4 Validation of the Masking Countermeasure

To mitigate the vulnerability demonstrated in the initial attack, a first-order **Boolean masking** countermeasure was implemented in hardware. This section details the principle behind this technique, its specific Verilog implementation, and the results of a repeated CPA attack to validate its effectiveness.

3.4.1 Principle of Boolean Masking

Masking is a side-channel attack countermeasure technique that involves randomizing intermediate data during cryptographic operations. It works by combining sensitive values (like plaintext or key material) with random masks, so that power or EM traces during execution do not directly reflect the true data being processed [20]. In a Boolean masking

scheme, a sensitive value X is combined with a random value M using XOR:

$$\text{Masked value} = X \oplus M$$

Side-channel attacks such as Correlation Power Analysis (CPA) exploit the fact that power consumption or electromagnetic emissions often leak information correlated with the values being processed. In particular, early AES operations (e.g., `plaintext` $\oplus$ `key` during the AddRoundKey step) provide strong leakage.
If an attacker knows the plaintext and guesses key bytes, they can correlate the measured power traces with predicted intermediate values to recover key bits.
Masking mitigates this vulnerability by randomizing the internal computation. Even if the attacker knows the plaintext, the AES core only processes a *masked plaintext*, which is unpredictable. As a result:

- Intermediate states (e.g., $P \oplus K$) are decorrelated from the real inputs.
- The attacker's statistical models no longer match actual intermediate values.
- Correlation or template attacks fail, and the recovered key will be incorrect.

This demonstrates that masking effectively breaks the exploitable leakage that CPA relies upon.

3.4.2 Hardware Implementation

The countermeasure was implemented by creating a new Verilog wrapper module named `masked_aes.v`. This module encapsulates the original AES core and performs the masking and unmasking operations.

The Verilog implementation follows this logic:

(i) **Mask Generation:** When a new encryption is initiated (`load_i` is asserted), a new 128-bit random `mask` is generated using the `$random` system task.

(ii) **Masking:** The incoming plaintext (`data_i`) is XORed with the `mask` to produce `masked_data_i`.

(iii) **Encryption:** This `masked_data_i` is passed to the original `aes_core` instance for encryption.

(iv) **Unmasking:** The encrypted output from the core (`masked_data_o`) is XORed again with the same `mask` to produce the final, correct ciphertext (`data_o`).

aes_masked.v

```
`timescale 1ns / 1ps

module masked_aes (
    input wire clk,
    input wire load_i,
    input wire [255:0] key_i,
    input wire [127:0] data_i,
    input wire [1:0] size_i,
    input wire dec_i,
    output wire [127:0] data_o,
    output wire busy_o
);

    reg [127:0] mask;
    wire [127:0] masked_data_i;
    wire [127:0] masked_data_o;

    always @(posedge clk) begin
        if (load_i) begin
            mask <= $random ^ ($random << 32) ^ ($random << 64) ^ ($random << 96);
        end
    end

   assign masked_data_i = data_i ^ mask;

    aes_core aes_core_inst (
        .clk(clk),
        .load_i(load_i),
        .key_i(key_i),
        .data_i(masked_data_i),
        .size_i(size_i),
        .dec_i(dec_i),
        .data_o(masked_data_o),
        .busy_o(busy_o)
    );

    // Unmask the ciphertext output
    assign data_o = masked_data_o ^ mask;

endmodule
```

Figure 3.13: AES Masked.

This new `aes_masked.v` file was added to the Vivado project, and the top-level design was updated to instantiate this module. We can verify this by running the traces and performing the same attack as we did before.

This time, just to be sure, we will go with 5000 traces only. Before execution of the script, we have to make sure we have created a separate file for `aes_masked.v` and added it in the Design Sources for Vivado. Note that this file must come in higher precedence as

compared to the `aes_core.v` module.

In the top module, we will replace the `aes_core.v` with the `aes_masked.v` function. After these steps, we will finally regenerate the bitstream and dump it into the FPGA and we will see in the results section.

4

Results

This section presents the empirical outcomes of the Correlation Power Analysis (CPA) attacks conducted on both the unprotected baseline AES core and the core protected with the masking countermeasure. The results confirm the initial vulnerability and subsequently validate the effectiveness of the implemented defense.

4.1 Experimental Details:

Equipment: ChipWhisperer Husky connected to CW305 FPGA board (100T variant) via 20-pin connector. SMA cables link FPGA measure point to Husky's positive terminal for power monitoring. USB-C connects Husky to laptop, separate USB programs the FPGA.

Software: Vivado 2025.1 synthesizes Verilog HDL into bitstreams. Python with ChipWhisperer libraries captures and analyzes power traces. NumPy handles statistical computations.

Configuration: PLL provides synchronized clocking for both FPGA operation and ADC sampling on the Husky platform [21].

4.2 Key Recovery from the Unprotected AES Core

We are performing a Correlation Power Analysis (CPA) to recover one byte of the AES key. First, we collect power traces during encryption and compute the Hamming weight of a key-dependent intermediate (SBox output) for each trace and key guess (0–255). For each guess, we calculate the correlation between the Hamming weight vector and all points in the power traces. The maximum absolute correlation indicates both the most likely key byte and the time index of leakage. NumPy makes this efficient by allowing vectorized operations like mean() and cov() without manual loops. The key guess with the highest correlation becomes our best estimate.

4.2.1 Visual Analysis of Power Traces:

Like we already discussed, it's difficult to gain meaningful traces from FPGAs since it's implemented directly into Logic Bits, so to tackle this issue, we will have to capture a lot more traces. In this demonstration, we have chosen to capture in the range of 5000 traces. We might require more or less traces depending on the hardware design [3].

Before beginning the capture process, we must first obtain a key text pair object using ChipWhisperer and ensure that a valid key is properly loaded on the target device. Once

these prerequisites are met, the basic capture loop follows a systematic five-step sequence that repeats for each trace collection. First, our arm the ChipWhisperer to prepare it for data acquisition. Next, we write the plaintext data to the target device that will be encrypted. The third step involves triggering the encryption operation on the target, which initiates the cryptographic process we want to analyze.

During this encryption, the fourth step occurs simultaneously as we capture the power consumption trace that reveals information about the internal operations. After the encryption completes, we read the resulting ciphertext back from the target device in the fifth step. Finally, we organize and store both the captured trace data and the corresponding plaintext-ciphertext pairs for subsequent analysis. This loop continues iteratively to collect multiple traces, building a dataset suitable for side-channel analysis attacks.

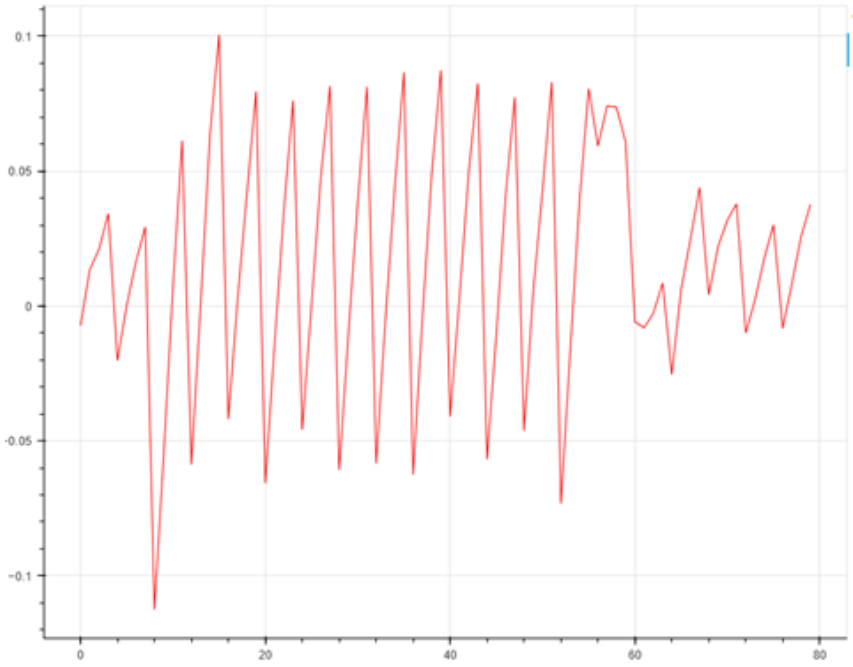


Figure 4.1: Graph of traces.

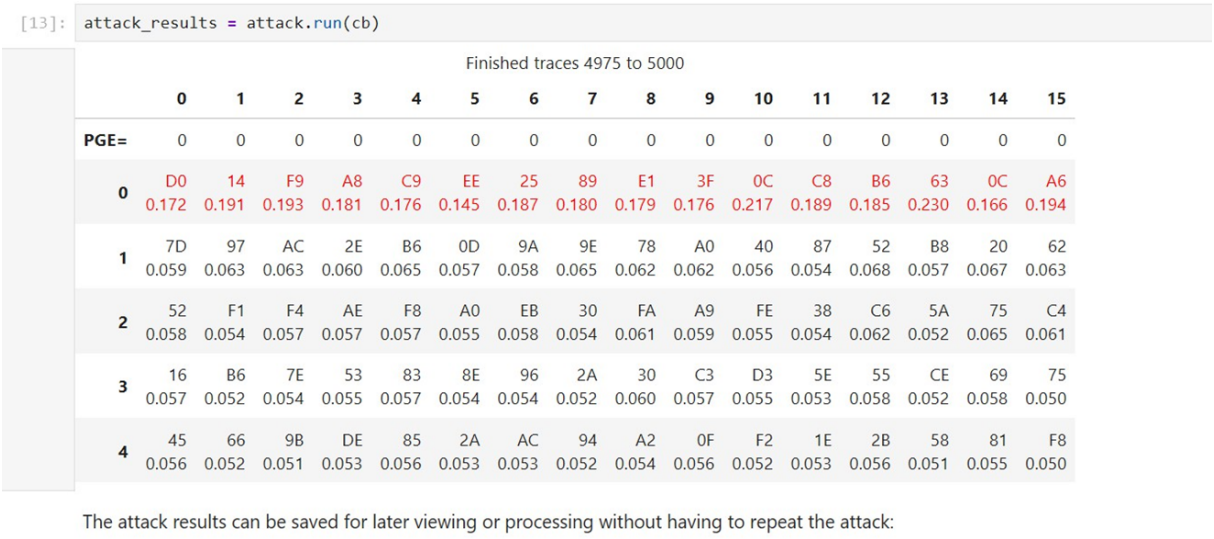
4.2.2 Quantitative CPA Results:

The CPA attack was performed using the chipwhisperer analyzer library, specifically targeting the final round of the AES algorithm with the last round state diff leakage model.

The analysis produced a ranked table of key byte guesses for each of the 16 bytes. For every byte, the correct guess consistently showed a significantly higher correlation value than any incorrect guess. For example, for the first byte, the top guess had a correlation of approximately 0.188, while the next-best guess was only 0.066, demonstrating a clear statistical distinction that made identifying the correct subkey trivial

```
import chipwhisperer as cw
import chipwhisperer.analyzer as cwa
project_file = "projects/Tutorial_HW_CW305"
project = cw.open_project(project_file)
attack = cwa.cpa(project, cwa.leakage_models.last_round_state_diff)
cb = cwa.get_jupyter_callback(attack)
```

Figure 4.2: chipwhisperer analyzer library.



[13]:
```
attack_results = attack.run(cb)
```

Finished traces 4975 to 5000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE=	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	D0 0.172	14 0.191	F9 0.193	A8 0.181	C9 0.176	EE 0.145	25 0.187	89 0.180	E1 0.179	3F 0.176	0C 0.217	C8 0.189	B6 0.185	63 0.230	0C 0.166	A6 0.194
1	7D 0.059	97 0.063	AC 0.063	2E 0.060	B6 0.065	0D 0.057	9A 0.058	9E 0.065	78 0.062	A0 0.062	40 0.056	87 0.054	52 0.068	B8 0.057	20 0.067	62 0.063
2	52 0.058	F1 0.054	F4 0.057	AE 0.057	F8 0.057	A0 0.055	EB 0.058	30 0.054	FA 0.061	A9 0.059	FE 0.055	38 0.054	C6 0.062	5A 0.052	75 0.065	C4 0.061
3	16 0.057	B6 0.052	7E 0.054	53 0.055	83 0.057	8E 0.054	96 0.054	2A 0.052	30 0.060	C3 0.057	D3 0.055	5E 0.053	55 0.058	CE 0.052	69 0.058	75 0.050
4	45 0.056	66 0.052	9B 0.051	DE 0.053	85 0.056	2A 0.053	AC 0.053	94 0.052	A2 0.054	0F 0.056	F2 0.052	1E 0.053	2B 0.056	58 0.051	81 0.055	F8 0.050

The attack results can be saved for later viewing or processing without having to repeat the attack:

Figure 4.3: Jupyter CPA Table.

4.2.3 Final Key Recovery:

We are performing a Correlation Power Analysis (CPA) to recover one byte of the AES key. First, we collect power traces during encryption and compute the Hamming weight of a key-dependent intermediate (SBox output) for each trace and key guess (0–255). For each guess, we calculate the correlation between the Hamming weight vector and all

points in the power traces. The maximum absolute correlation indicates both the most likely key byte and the time index of leakage. NumPy makes this efficient by allowing vectorized operations like mean() and cov() without manual loops. The key guess with the highest correlation becomes our best estimate. Over 0-255, we will now get the key byte along with the correlation, that looks something like this:

```
Key guess:  0x2b
Correlation:  0.8270952071731353
```

Figure 4.4: Key byte.

Now we will simply repeat these steps for the rest of the key bytes and we will have the entire key.

```
Best Key Guess: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
 [0.8270952071731353, 0.8270327634692841, 0.8720468392820594,
0.9072878953165543, 0.838143352834745, 0.7929630633702721,
0.8092169020292708, 0.9049490689960625, 0.8190023842984117,
0.8493258268356841, 0.8552424694370616, 0.8574448424694251,
0.8505092915049918, 0.8633106161682341, 0.9282234083165303,
0.8340240668178257]
```

Figure 4.5: Key and their Correlation.

Test

Check that the key obtained by the attack is the key that was used. This attack targets the last round key, so we have to roll it back to compare against the key we provided.

```
[16]: key = list(key)
      assert (key == recv_key), "Failed to recover encryption key\nGot:     {}\nExpected: {}".format(recv_key, key)
      print(key)

      [43, 126, 21, 22, 40, 174, 210, 166, 171, 247, 21, 136, 9, 207, 79, 60]
```

Figure 4.6: Test key in Decimal.

4.2.4 Attack Efficiency:

We can verify that this indeed is the correct key. Upon running multiple iterations, we have noticed that the correct key can be guessed within 800 traces only. But still to be on the safe side, we are going to run up to 1500 traces.

This high efficiency underscores the practical risk posed by the side-channel leakage [16]. So far, we have run Side Channel Analysis on unsecure implementation. There were no safety measures in place to counter SCA. Now we will try implementing some countermeasures to make the extracted key meaningless.

4.3 Result and Validation of the Masking Countermeasure:

The same CPA attack was repeated using 5,000 power traces captured from the protected core. The results demonstrated a comprehensive failure of the attack.

4.3.1 Visual Analysis:

The captured power traces appeared visually distinct from the unprotected traces, looking more random and less structured, providing an initial indication that the masking was effective.

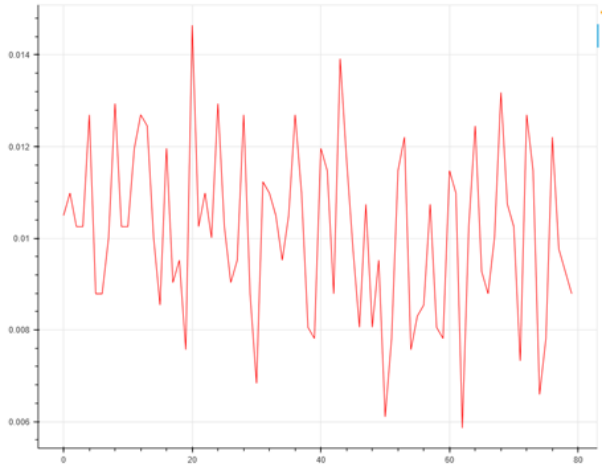


Figure 4.7: Graph Traces.

4.3.2 Quantitative Analysis:

The CPA attack **failed** to recover the key. The analysis was only able to correctly identify **1 out of 16 key bytes**, which is consistent with random chance.

Finished traces 4975 to 5000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE=	0	229	80	144	158	90	218	119	29	205	148	38	49	199	221	113
0	D0 0.055	24 0.055	08 0.057	E5 0.062	69 0.058	A3 0.065	A2 0.056	2E 0.059	FA 0.059	26 0.064	1C 0.059	EF 0.066	C2 0.058	41 0.057	F5 0.058	2B 0.055
1	07 0.053	2F 0.053	BC 0.057	DA 0.060	55 0.053	E5 0.054	ED 0.055	86 0.055	8F 0.057	45 0.056	85 0.059	60 0.055	26 0.056	30 0.055	D5 0.057	8D 0.053
2	67 0.053	43 0.053	E9 0.055	C4 0.059	BA 0.053	05 0.054	AD 0.051	04 0.053	67 0.057	E3 0.054	E3 0.059	3A 0.053	F1 0.055	86 0.052	41 0.054	E8 0.051
3	C5 0.053	FB 0.053	FC 0.055	EE 0.059	3F 0.053	98 0.054	35 0.050	56 0.051	BB 0.054	3D 0.053	7B 0.056	13 0.052	CA 0.055	D1 0.052	9B 0.054	31 0.050
4	99 0.051	AA 0.052	D2 0.054	CC 0.055	5F 0.051	45 0.053	08 0.050	49 0.050	94 0.054	90 0.052	AA 0.054	45 0.052	EF 0.054	C9 0.052	0B 0.054	B1 0.049

Figure 4.8: Masked Traces.

4.3.3 Verification of Failure:

The failure was definitively confirmed by an `AssertionError`, which showed that the recovered key was completely different from the expected key.

```
key = list(key)
assert (key == recv_key), "Failed to recover encryption key\nGot:      {}\nExpected: {}".format(recv_key, key)
print(key)
print(recv_key)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[16], line 2
      1 key = list(key)
----> 2 assert (key == recv_key), "Failed to recover encryption key\nGot:      {}\nExpected: {}".format(recv_key, key)
      3 print(key)
      4 print(recv_key)

AssertionError: Failed to recover encryption key
Got:      [200, 30, 247, 223, 185, 123, 200, 227, 90, 152, 54, 250, 121, 219, 176, 162]
Expected: [43, 126, 21, 22, 40, 174, 210, 166, 171, 247, 21, 136, 9, 207, 79, 60]
```

Figure 4.9: Only ONE correct key Recover.

This outcome demonstrates that the implemented first-order masking was successful in defeating the first-order CPA attack. It is noted, however, that the design remains vulnerable to **higher-order attacks** (e.g., second-order CPA) which could potentially bypass this countermeasure.

Bibliography

[1] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," *Advances in Cryptology – CRYPTO '99*, pp. 388–397, 1999, one of the foundational works introducing DPA.

[2] S. Mangard, E. Oswald, and T. Popp, "Power analysis attacks: Revealing the secrets of smart cards," *IEEE Transactions on Device and Materials Reliability*, vol. 7, no. 3, pp. 441–450, 2007, covers CPA and masking techniques in hardware.

[3] P. Socha, "Survey: 20 years of power side-channel attack analysis," *Cryptography Journal*, 2020, comprehensive overview of SPA, DPA, CPA, TVLA, and related techniques.

[4] E. Peeters, F.-X. Standaert, and J.-J. Quisquater, "Improved higher-order side-channel attacks with fpga experiments," *Proceedings of CHES*, pp. 309–323, 2005, demonstrates higher-order attacks on masked FPGA implementations.

[5] A. T. Mozipo and J. M. Acken, "Analysis of countermeasures against remote and local power side channel attacks using correlation power analysis," *IEEE Transactions on Dependable and Secure Computing*, 2024, studies efficacy of CPA against hardware countermeasures.

[6] D. Das, S. Maity, S. B. Nasir, S. Ghosh, A. Raychowdhury, and S. Sen, "High efficiency power side-channel attack immunity using noise injection," *IEEE Transactions on VLSI Systems*, vol. 26, no. 3, pp. 472–486, 2018, presents noise-based countermeasure for AES.

[7] H. Wang *et al.*, "Tandem deep learning side-channel attack on fpga aes," in *SN Computer Science*, 2021, uses deep learning ensemble attacks to reduce trace count in AES- FPGA.

[8] P. Socha *et al.*, "Boolean masking scheme for high-level synthesis in s–p networks," *Journal of Systems Architecture*, 2021, proposes FPGA-friendly masking at high-level RTL synthesis.

[9] O. Westman and M. Hell, "Electromagnetic side-channel attack on aes using low-end equipment," *Journal of Cryptographic Engineering*, 2019, explores AES key retrieval via EM from FPGA with consumer-grade tools.

[10] M. A. Shelton, N. Samwel, L. Batina *et al.*, "Rosita: Automatic elimination of power-analysis leakage in ciphers," *IACR ePrint Archive*, 2019, automates leak-free masked AES implementation synthesis.

[11] K. Ramezanpour, P. Ampadu, and W. Diehl, "Scaul: Side-channel analysis with unsupervised learning," *IACR ePrint Archive*, 2020, uses LSTM for model-free CPA on AES hardware.

[12] S. R. Bommana *et al.*, "Mitigating side-channel attacks on fpga through deep learning and dynamic partial reconfiguration," *Scientific Reports*, 2025, dynamic reconfiguration and ML-based defense against power SCA on FPGA.

[13] L. Zhao, G. Bontempi, and O. Markowitch, "Power analysis attack based on machine learning," in *ICApplied Crypto*, 2014, applies ML to improve key recovery in power analysis attacks.

[14] R. Lumbiarres-López *et al.*, "Hardware architecture on fpga for protecting keys against sca," *IEEE TDSC*, 2018, fPGA-based AES implementation with SCA countermeasures.

[15] D. Fujimoto and Y. Hayashi, "Current consumption modeling of logic cells for side-channel simulation," in *ICASSP*, 2024, lessons for SCA-aware FPGA design.

[16] E. Prouff and M. Rivain, "Advances in crypto – eurocrypt 2013," *LNCS*, 2013, includes updated masking countermeasure methods.

[17] W. Diehl *et al.*, "Comparing cost of protecting lightweight ciphers against dpa in fpgas," *Computers*, 2018, cost-security tradeoffs in SCA-protected hardware.

[18] N. Ismail *et al.*, "Deep learning can break microcontroller security via power analysis," in *Trusted Systems*, 2025, explores DL-driven SCA on lightweight crypto.

[19] C. Giraud, "Differential fault analysis on aes," in *CHES*, 2005, landmark DFA on AES.

[20] S. Chari *et al.*, "Note regarding evaluation of aes on smart cards," *NIST*, 1999, early caution on implementation vs. algorithmic security.

[21] N. Pramstaller *et al.*, "Masked aes asic implementation," *IACR ePrint Archive*, 2004, early masked AES hardware design.

[22] K. Ramezanpour, P. Ampadu, and W. Diehl, "Rs-mask: Random space masking against power and fault analysis," in *IEEE HOST*, 2019, a masking variant providing resistance to both SCA and fault analysis.

[23] B. Timon, "Non-profiling deep learning side-channel attacks with sensitivity analysis," in *IACR Transactions on CHES*, 2019, deep learning model does SCA on AES without profiling.

[24] B. Zhao *et al.*, "Revisiting attention mechanism in time-series classification," *arXiv preprint*, 2022, highlights advanced feature extraction (relevant to ML in SCA).

[25] M. Zhao and G. E. Suh, "Fpga-based remote power side-channel attacks," in *2018 IEEE SP*, 2018, remote power SCA on shared FPGA platforms.

[26] L. Goubin and J. Patarin, "Des and differential power analysis," *CHES*, 1999, early demonstration of DPA on cryptographic hardware.

[27] P. Kocher, "Introduction to differential power analysis and related attacks," *Technical Report*, 1998, the original introduction of DPA concepts.

[28] D. Molnar *et al.*, "Program counter security model: Removing control-flow side-channel," *USENIX WIP*, 2005, software-level countermeasure concept; included for contrast.

[29] ——, "Pc-secure programs," *USENIX Secure Systems*, 2005, mitigates control-flow side channels (timing-related).

[30] A. Wikipedia, "Power analysis (wikipedia overview)," *Wikipedia*, 2025, general overview entry for educational framing.