Gray box analysis of Trusted execution environment(TEE) and post-quantum algorithms

Thesis submitted to the
Indian Institute of Technology, Bhilai
For award of the degree

of

Master of Technology

by

Chayan Pathak

Under the guidance of

Dr. Dhiman Saha

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY BHILAI

April 2025

To

Dedication

Sub- Caption

APPROVAL OF THE VIVA-VOCE BOARD

Date: _____

Certified that the thesis entitled **"Thesis Title"** submitted by **Your Name** to the Indian Institute of Technology, Kharagpur, for the award of the degree Doctor of Philosophy has been accepted by the external examiners and that the student has successfully defended the thesis in the viva-voce examination held today.

(Member of the DSC) (Member of the DSC) (Member of the DSC)

(Supervisor)

(External Examiner) (Chairman)

Certificate

This is to certify that the thesis entitled **"Thesis Title"** submitted by **Your Name (Roll Number)** to the Indian Institute of Technology Kharagpur, is a record of bonafide research work carried under my supervision and is worthy of consideration for the award of the Doctor of Philosophy of the Institute.

Date: dd.mm.yyyy

Place: IIT Kharagpur

Guide Name

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus

tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Your Name

IIT Kharagpur

Declaration

I Certify that

a. the work contained in the thesis is original and has been done by me under the guidance of my Supervisor;

b. the work has not been submitted to any other Institute for any degree or diploma;

c. I have followed the guidelines provided by the Institute in preparing the thesis;

d. I have conformed to ethical norms and guidelines while writing the thesis and;

e. whenever I have used materials (data, models, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis, giving their details in the references, and taking permission from the copyright owners of the sources, whenever necessary.

Date: dd.mm.yyyy

Place: IIT Kharagpur

Your Name

List of Abbreviations

SAMPLE	Sample
SAMPLE	Sample
SAMPLE	Sample

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

1.1 Motivation and Objectives

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

CHAPTER 2

Literature Survey

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla

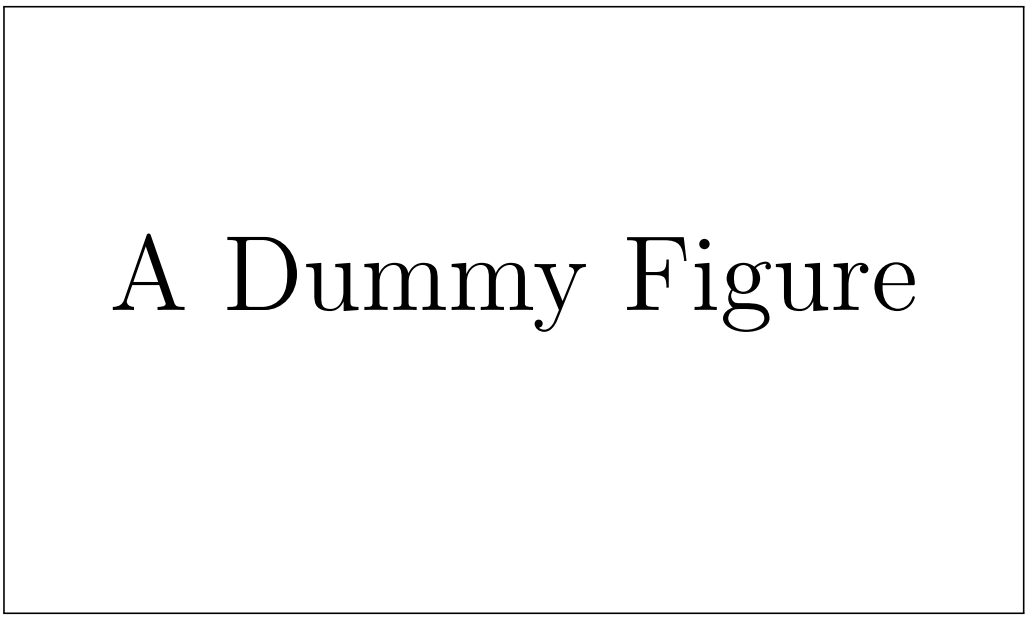

Figure 2.1: Dummy Figure

vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

To get a list of contributions as a shaded box

- First itemtext
- Second itemtext
- Last itemtext
- First itemtext
- Second itemtext

CHAPTER 3

Fault Attack Lab Setup

In recent years, fault injection attacks have gained significant attention within the hardware security community due to their effectiveness in compromising embedded systems. By intentionally introducing faults during the execution of cryptographic or security-critical operations, attackers can exploit vulnerabilities that may not be apparent under normal conditions. As research in this area advances, the need for a reliable, flexible, and well-documented laboratory environment becomes crucial. A carefully designed lab setup not only ensures the accuracy and reproducibility of experiments but also helps in the development and evaluation of robust countermeasures.

This chapter details the laboratory environment established for conducting fault injection experiments, with a focus on two key tools: the ChipWhisperer and the ChipSHOUTER. The ChipWhisperer platform is widely recognized in the research community for its capabilities in side-channel analysis and glitch-based fault injection. It provides a tightly integrated solution that combines hardware capture tools with a comprehensive software framework, allowing for fine-grained control over experiments. Through features such as precise clock glitching, voltage fault injection, and automated trace capture, ChipWhisperer serves as a foundation for a variety of fault attack methodologies.

To expand the range of possible attacks, particularly in scenarios where non-invasive methods are preferred, the ChipSHOUTER is incorporated into the setup. The ChipSHOUTER enables electromagnetic fault injection with high precision, delivering short, high-voltage pulses that can disrupt the target device without any physical contact. This capability is especially valuable for simulating real-world attack scenarios, where access to a device may be limited or where invasive methods are impractical.

The combination of ChipWhisperer and ChipSHOUTER provides a versatile environment capable of supporting a wide spectrum of fault injection techniques. This chapter will describe the specific hardware components used, the setup and configuration processes, and the challenges encountered during integration. Attention is given to aspects such as signal synchronization, power supply management, timing calibration, and safety precautions, all of which are critical for the success of fault injection experiments.

By documenting the lab setup in detail, this chapter aims to provide a clear guide for reproducing the environment, thereby supporting transparency and repeatability in experimental research. Furthermore, it serves as a reference for anyone seeking to extend the setup for more advanced fault analysis, side-channel studies, or the testing of hardware security mechanisms against active fault-based threats.

3.1 Techniques and analysis: Side-Channel Attacks Using ChipWhisperer

ChipWhisperer is an open-source platform developed by NewAE Technology Inc. for conducting side-channel analysis and fault injection attacks. It integrates hardware and software components in a tightly-coupled system, allowing for efficient experimentation and research. This section provides a detailed overview of the ChipWhisperer setup used in the laboratory, covering the hardware tools, software environment, and essential firmware components.

3.2 ChipWhisperer Hardware Platform

The ChipWhisperer ecosystem includes a suite of hardware components specifically designed for performing side-channel analysis and fault injection on embedded systems. These components are broadly categorized into scope boards and target boards, which together create a complete testbench for evaluating the security of cryptographic implementations.

3.2.1 Scope Boards (Capture Hardware)

Scope boards are at the heart of ChipWhisperer's side-channel capture capabilities. They serve as the interface between the host computer and the target device, enabling the collection of high-resolution power traces or electromagnetic emissions during cryptographic operations. These boards also provide fine-grained control over clock generation, triggering, and synchronization.

A popular example is the ChipWhisperer-Lite, an all-in-one board combining scope and target functionality, ideal for students and researchers. ChipWhisperer-Husky offer higher sampling rates, more precise timing, and expanded features such as fault injection support. Key features of scope boards:

- Adjustable clock generation and distribution
- Trigger input/output for synchronization with DUT
- High-speed ADCs for capturing power consumption
- Communication interfaces (USB, serial, etc.)

Example usage scenario: Capturing power traces during AES encryption for differential power analysis (DPA).

3.2.2 Target Boards (Device Under Test - DUT)

3.2.3 Integrated Target on ChipWhisperer-Nano

The CWNano is a small and low-cost tool 3.1 used to learn about hardware security. It has both a target microcontroller and a measurement unit on a single board, making it easy to use. The built-in STM32F030F4P6 microcontroller has 16 KB of flash and 4kB of SRAM, suitable for running simple cryptographic algorithms. The board supports power analysis using an 8-bit ADC with a sampling rate of up to 20 MS/s. It also features basic voltage glitching through a crowbar method. The Nano connects to a computer via USB and is controlled using the ChipWhisperer software suite. It does not support clock glitching and has a limited flash size, but it is ideal for beginners, students, and hobbyists exploring side-channel analysis and embedded security. More details about this can be found in the official documentation [New25b].

3.2.4 Integrated Target on ChipWhisperer-Lite

The CWLite is a compact and low-cost hardware tool designed to help users learn about hardware security. It combines a power measurement system and a microcontroller target on a single board, making it easy to use. This device [New25c] can capture power traces with high speed and allows for glitching attacks like clock or voltage glitches. It is mainly used to perform and study side-channel attacks, such as breaking encryption by analyzing power usage. The board connects to a computer through USB and works with open-source ChipWhisperer software. Because of its simplicity and low price, it is widely used in education and research.

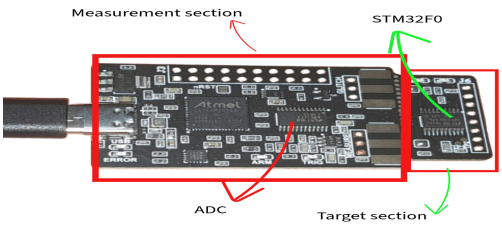

Figure 3.1: ChipWhisperer-Nano

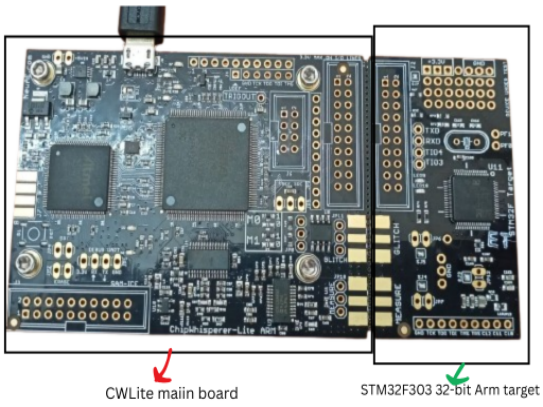

Figure 3.2: ChipWhisperer-Lite with 32-bit ARM target

It is offered in four hardware options [New25a] to suit different learning and testing needs. The first version is the CWLite-XMEGA, which includes both the capture hardware and an ATxmega128D4 microcontroller on a single board. The second option, CWLITE-ARM, features an STM32F3 ARM Cortex-M4 microcontroller instead of the XMEGA, also on a single board as shown in figure 3.2.
The CWLite 2-Part version as shown in figure 3.3 separates the capture and target sections into two distinct boards, connected via SMA and 20-pin IDC cables. This modular design allows users to easily swap or upgrade target devices,

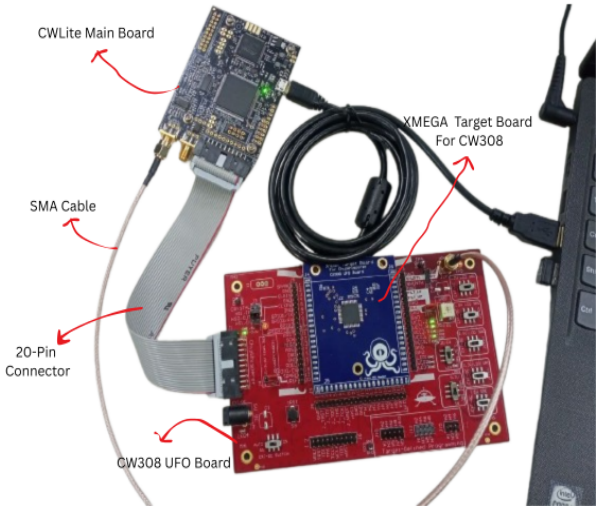

Figure 3.3: ChipWhisperer-Lite 2-Part version connections

enhancing flexibility for various testing scenarios. The capture board features a 10-bit ADC with a maximum sample rate of 105 MS/s, adjustable gain up to +55 dB, and supports both voltage and clock glitching with fine-grained control. The target board includes an ATxmega128D4 microcontroller, suitable for implementing and analyzing cryptographic algorithms. This two-part configuration is ideal for users seeking a versatile and open-source platform for embedded hardware security research.
The CWLite Standalone is a specialized capture board designed for side-channel power analysis and fault injection experiments. Unlike other variants, it does not include an integrated target microcontroller, providing flexibility to connect external targets via standard interfaces. It is ideal for researchers and educators who wish to interface with custom or third-party targets while utilizing ChipWhisperer's powerful capture and analysis capabilities.

3.2.5 ChipWhisperer-Husky

Together, the scope and target boards in the ChipWhisperer hardware ecosystem offer a comprehensive solution for conducting side-channel and fault injection attacks. Their modular, open-source nature makes them accessible to students, researchers, and engineers alike, and they are widely used in academic studies, industry evaluations, and security training environments.

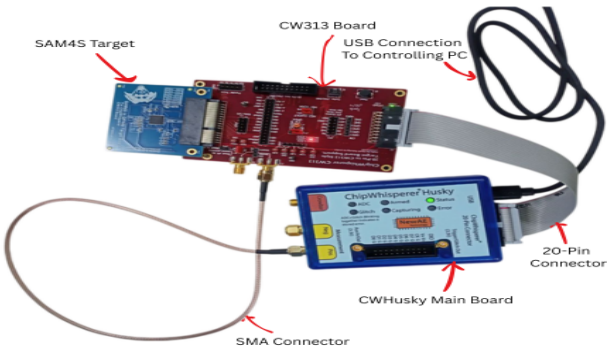

Figure 3.4: ChipWhisperer-Husky with SAM4S target

3.3 Software Environment

The ChipWhisperer platform includes a flexible and scriptable software environment built primarily around Python and Jupyter Notebooks. This environment is well-suited for side-channel analysis and fault injection research, offering full control of the hardware and reproducibility.

ChipWhisperer Software Environment

Setting up the ChipWhisperer software environment involves installing the necessary tools to interface with the hardware, control experiments, and analyze captured data. This section outlines the steps required to install the ChipWhisperer software stack using the recommended method.

For Windows :

Installing ChipWhisperer on Windows is straightforward with the provided ėxe file. Once the download is complete, double-click the .exe file to launch the installer. If a security prompt appears, allow the installation to proceed. The ChipWhisperer Setup Wizard will open—follow the on-screen instructions by clicking "Next." Choose a destination folder or use the default option, then continue by clicking "Next" again. After the installation is finished, click "Finish" to complete the setup. The simplest way to start using ChipWhisperer and access its tutorials is to open the ChipWhisperer application. You can find it in

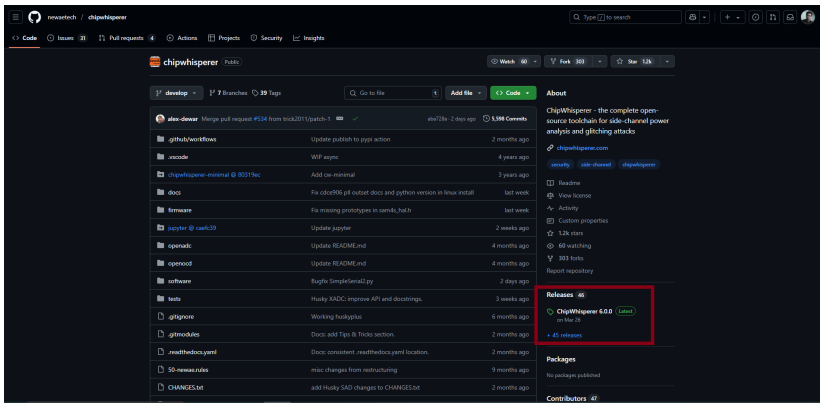

Figure 3.5: Chipwhisperer official GitHub page

the Start Menu, the installation directory, or on your desktop if you chose to create a shortcut during setup.

for Linux:

If you're using Ubuntu and want to install ChipWhisperer quickly, follow these steps. Start by updating your system:

```
sudo apt update && sudo apt upgrade
```

Next, install all required dependencies:

```
sudo apt install libusb-dev make git avr-libc gcc-avr \
    gcc-arm-none-eabi libusb-1.0-0-dev usbutils python3 python3-venv
```

Clone the ChipWhisperer repository from GitHub and navigate to the directory:

```
cd ~/
git clone https://github.com/newaetech/chipwhisperer
cd chipwhisperer
```

Set up a Python virtual environment and activate it:

```
python3 -m venv ~/.cwvenv
source ~/.cwvenv/bin/activate
```

Copy the USB rules and reload them:

```
sudo cp 50-newae.rules /etc/udev/rules.d/50-newae.rules
sudo udevadm control --reload-rules
```

Add your user to the necessary groups:

```
sudo groupadd -f chipwhisperer
sudo usermod -aG chipwhisperer $USER
sudo usermod -aG plugdev $USER
```

Initialize required submodules and install Python packages:

```
git submodule update --init jupyter
python -m pip install -e .
python -m pip install -r jupyter/requirements.txt
```

After rebooting your system, you can launch ChipWhisperer Jupyter notebooks with:

```
cd ~/chipwhisperer
jupyter notebook .
```

for MacOs

3.3.1 Directory Structure

The ChipWhisperer framework is a comprehensive toolkit for side-channel power analysis and glitching attacks. Its directory structure is designed to support both hardware interfacing and educational experimentation. Understanding this structure is key to navigating the project efficiently, especially when working with Jupyter notebooks or compiling firmware.

Here's a breakdown of the most relevant parts:

/jupyter Directory:

One of the most user-friendly components of the ChipWhisperer project is the `/jupyter` directory. This folder contains a wide range of Jupyter Notebooks designed to guide users through hands-on tutorials and experiments.

These notebooks provide interactive lessons in topics such as: Side-channel analysis, Cryptographic attacks, Glitching techniques The content within the `/jupyter` directory is well-organized into subfolders, such as:

The Jupyter directory in ChipWhisperer includes several folders designed to support learning and experimentation. The `demos` folder contains simple, hands-on notebooks meant for beginners. These notebooks introduce key concepts like power analysis and fault injection through guided examples. The `courses` folder offers more structured content, similar to academic classes, with lecture-style explanations and lab exercises for deeper learning. The `setupscripts` folder provides example scripts and configurations that demonstrate different experiments and usage scenarios for ChipWhisperer hardware.

Together, these resources help users understand and explore various aspects of side-channel analysis and hardware security.

Python and Hardware Integration

The notebooks leverage Python code to directly interface with ChipWhisperer hardware. This allows for:

- Real-time data capture
- On-the-fly analysis
- Dynamic control of experiments

These notebooks are intended to be executed within a Jupyter Lab environment. Users can easily launch the environment using the `chipwhisperer` command-line tools, providing a smooth workflow from code to hardware interaction.

3.3.2 Scope API

The `scope` object in ChipWhisperer is used to manage the capture and glitching operations of the hardware.The official documentation [New25d] provides a comprehensive overview of the scope API, detailing how to configure the hardware, capture traces, and perform glitching operations. The scope API is designed to be intuitive and flexible, allowing users to easily set up their experiments and interact with the ChipWhisperer hardware.

To create a `scope` object, the `chipwhisperer.scope()` function is the easiest approach to use. This function connects to a ChipWhisperer device and returns an instance of the appropriate scope object:

```
import chipwhisperer as cw
scope = cw.scope()
```

There are two types of `scope` : OpenADC for `CWLite/Husky` and CWNano for `CWNano`. The choice of scope depends on the specific hardware being used. The `scope` object provides various properties and methods to configure the hardware, capture traces, and perform glitching operations. Here are some of the key properties and methods:

`scope.adc.samples` :

This property sets the number of samples to capture by the ADC (Analog-to-Digital Converter). It is useful for determining the length of the captured trace. maximum number of samples for Lite is 244000 and for Husky is 131070

`scope.adc.timeout :`

Specifies the maximum duration (in seconds) the ADC will wait for a trigger signal before aborting the capture. This prevents indefinite waiting during a capture session.

$scope.clock.adc\_src$:

Determines the clock source for the ADC module. Generally it will be $clkgen\_x1$ or $clkgen\_x4$ which correspond to different clock frequencies.

`glitch.clk_src :`

This setting selects the clock source used by the glitch module's DCM (Digital Clock Manager). The DCM determines the timing of when glitches are produced. The available clock sources are:

- clkgen: Uses the output from the internal clock generator. *Note: This option is not supported on Husky.*
- pll: Uses the on-board Phase-Locked Loop (PLL) available on Husky devices. *Note: This is specific to Husky.*

Selecting the correct clock source is important for ensuring that glitches are timed correctly with the target device's operation.

$scope.glitch.output$:

This setting controls the type of signal produced by the glitch module. It defines how the regular clock and glitch pulses are combined to create the final output signal. The available output modes are:

- clock_only: Outputs only the normal clock signal, with no glitches applied.
- glitch_only: Outputs only the glitch pulses, ignoring the clock signal entirely.
- clock_or: The output is high when either the clock or the glitch pulse is high.
- clock_xor: The output is high only when the clock and glitch pulse differ.
- enable_only: The output stays high for a set number of full clock cycles. In this mode, `width` and `width_fine` have no effect, but `offset` and `offset_fine` are still used.

These modes are chosen based on the type of glitching attack:

- For clock glitching, use `clock_or` or `clock_xor`.
- For oltage glitching, use `glitch_only` or `enable_only`.

scope.glitch.trigger_src :

The glitch module supports four trigger modes:

- Continuous: Glitches are triggered constantly. Parameters like `ext_offset`, `repeat`, and `num_glitches` have no effect.
- Manual: Glitches are triggered manually via `manual_trigger()` or `scope.arm()`. Only `repeat` is relevant; others are ignored.
- ext_single: Triggers once per arming when a condition is met. Ignores further triggers until re-armed. `ext_single` provides a controlled, one-time glitch per arming cycle based on an external condition—making it useful for repeatable and precise fault injection experiments.
- ext_continuous: Triggers glitches repeatedly whenever the trigger condition is met, regardless of arm status.

scope.glitch.repeat :

`repeat` is a setting that controls how many glitch pulses are generated per trigger.

- If the output mode is `glitch_only`, `clock_or`, or `clock_xor`, each count in `repeat` represents a separate glitch pulse.
- If the output mode is `enable_only`, the glitch is a single pulse that lasts for `repeat` clock cycles.
- This helps in creating stronger glitches, especially for voltage glitching.
- On **CW-Husky**, if multiple glitches are used (`num_glitches > 1`), `repeat` should be a list with one value per glitch. Each value must be $\leq$ `ext_offset`$[i+1]+1$.
- On **CW-Lite/Nano**, only one glitch is supported, so `repeat` is a single integer.
- The value of `repeat` must be in the range [1, 8192], and it has no effect in `continuous` mode.

`scope.glitch.width :`

The `width` property defines how wide a single glitch pulse is and can be set using either a float or an integer. Its meaning depends on the type of hardware used. For CWHusky, the width is measured in phase shift steps. A value of 0 gives the smallest pulse, while the maximum is at half the total number of phase shift steps. Negative values are allowed and are interpreted as wrapping around the phase shift range, so $-x$ is treated the same as `total_steps` $- x$. The values also wrap around when going beyond the maximum.

For other devices like CWLite or CWNano, the width is given as a percentage of one clock cycle. You can set the pulse from about -49.8% to $+49.8\%$ of the cycle. A width of 0% will not work reliably. Negative widths behave like their positive counterparts but are applied to the opposite half of the clock cycle. This setting has no effect if the output mode is set to `enable_only`.

`scope.glitch.offset :`

The `offset` property sets the delay between the rising edge of the clock and the start of the glitch pulse. It can be a float or an integer, and its meaning depends on the hardware used.

For CWHusky, the offset is measured in phase shift steps. An offset of 0 means the glitch pulse starts exactly at the rising edge of the clock. At half of `scope.glitch.phase_shift_steps`, the glitch starts at the falling edge of the clock. We can use negative values, and $-x$ is treated the same as `scope.glitch.phase_shift_steps` $-x$. The value also wraps around, so $+x$ is the same as `scope.glitch.phase_shift_steps` $+x$.

For other devices like CWLite or CWPro, the offset is given as a percentage of one clock period. The glitch can start anywhere from -49.8% to $+49.8\%$ of the clock cycle. This allows us to move the glitch to any point within the cycle.

$scope.glitch.ext\_offset$:

The `ext_offset` property defines how many clock cycles the glitch module should wait after receiving a trigger before generating a glitch pulse. This delay is useful when you want to insert the glitch at a specific point in the target device's operation, such as during the execution of a particular instruction. On CW-Lite and CW-Pro, multiple glitches are not supported, so `ext_offset` is simply an integer representing the delay after the trigger to the single glitch. This setting has no effect if the trigger source is set to `manual` or `continuous`.

The `offset` property, on the other hand, controls the position of the glitch within a single clock cycle. While `ext_offset` determines *when* the glitch

should start in terms of clock cycles after the trigger, `offset` adjusts the glitch *within* a chosen clock cycle—allowing very precise control relative to the rising or falling edge of the clock signal. In CW-Husky, `offset` is measured in phase shift steps, and in CW-Lite or CW-Pro, it is expressed as a percentage of one clock period. Together, these two properties provide both coarse and fine control over glitch timing.

$scope.arm()$:

The `arm()` function prepares the scope to start capturing data or performing a glitch when a trigger is received. This step is required before any capture or glitch attempt can begin. If the scope is set to `ext_single` mode, it will remain idle until it is armed and a valid trigger event occurs. Without calling `arm()`, the scope will not respond to trigger signals.

$scope.capture()$:

The `capture(poll_done=False)` function starts the process of capturing a trace. Before using this function, the scope must be armed using `arm()`. Once called, it waits for a trigger event. When the trigger occurs or a timeout is reached, the function stops the capture, disarms the scope, and retrieves the recorded data.

$scope.get\_last\_trace()$:

The `get_last_trace(as_int=False)` function returns the most recent trace captured by the scope. By default, it provides the data as a NumPy array of floating-point values scaled between `-0.5` and `0.5`. If the parameter `as_int` is set to `True`, the function returns the trace as raw integer values, which are the direct outputs from the ADC of the ChipWhisperer device. The resolution of these values depends on the hardware: for example, the ChipWhisperer-Lite uses a 10-bit ADC, the Nano uses 8-bit, and the Husky can use either 8-bit or 12-bit ADC data.

3.3.3 Target API

The `target` object in ChipWhisperer is used to manage the device under test (DUT) and perform operations such as loading firmware, executing commands, and capturing traces. It provides a high-level interface for interacting with the target device. ChipWhisperer provides two classes for UART communication:

- Simple Serial Target (default)

- Simple Serial V2 Target

The most straightforward way to create a target object in ChipWhisperer is by using the `cw.target` function. Here is a simple example:

```
import chipwhisperer as cw
scope = cw.scope()
try:
    if SS_VER == "SS_VER_2_1":
        target_type = cw.targets.SimpleSerial2
    else:
        target_type = cw.targets.SimpleSerial
except:
    SS_VER="SS_VER_1_1"
    target_type = cw.targets.SimpleSerial

try:
    target = cw.target(scope, target_type)
```

This code initializes the ChipWhisperer scope and sets up the Simple Serial target.

some useful methods and properties of the `Simple Serial V2 Target` object include:

$target.flush()$:

The `flush()` function clears all data currently stored in the serial buffer. This is useful when you want to discard any previous or unwanted serial communication data before starting a new operation. It helps ensure that the buffer only contains fresh data relevant to the current task.

$target.simpleserial\_write(cmd, data)$:

This function sends a command and associated data to a target device using the SimpleSerial protocol. This function is typically used when communicating with firmware that implements cryptographic functions such as AES encryption.

The `cmd` parameter is a one-character string that specifies the type of command. For example, using `'p'` tells the device to encrypt the given plaintext, while `'k'` sets the encryption key. These special cases internally map to specific command and sub-command values expected by the firmware.

The `data` parameter is a `bytearray` containing the actual data to be sent with the command. If no data is provided, a single byte `[0x00]` is sent by

default. The optional end argument is reserved but not used in this implementation.

Example: To send a 16-byte plaintext to be encrypted using the AES block cipher, one might use the following:

```
target.simpleserial_write('p', bytearray([0x00]*16))
```

This line sends a block of 16 zero bytes to the device with the command `'p'`, which typically triggers AES encryption of the plaintext using a previously loaded key.

$target.simpleserial\_read\_witherrors(cmd, length)$:

The `simpleserial_read_witherrors()` function is used to read data from a device over a serial connection, especially when doing fault injection or glitch experiments. It tries to receive a full response from the device that includes a command, some data, and a special ending byte. If the response is incomplete, contains errors, or takes too long, the function will try one more time with a longer timeout to get whatever data is available. This is helpful when the target device behaves in unexpected ways due to glitches. The result is returned as a dictionary, which includes whether the response was valid, the decoded data if successful, the full raw output, and any return values if needed.

Example: Suppose we expect a 16-byte result from a previous command, such as an AES ciphertext. We can read it like this:

```
response = simpleserial_read_witherrors(cmd='r', pay_len=16)
if response['valid']:
    print("Received:", response['payload'])
else:
    print("Invalid response. Raw output:", response['full_response'])
```

In this example, the function tries to read 16 bytes from a response starting with the command `'r'`. If it fails due to glitches or timing issues, it still gives the raw data so we can analyze what went wrong.

$target.simpleserial\_wait\_ack(cmd, timeout = 1)$:

This function is used to wait for an acknowledgment (ack) or error message from the target device after sending a command. This is useful to confirm whether the target received and understood the previous instruction. You can set a timeout in milliseconds, which defines how long to wait for the ack. If

the timeout is set to 0, the function will wait indefinitely until it receives a response. If no ack is received within the given time, the function returns `None`. Otherwise, it returns a code that indicates the result of the command.

3.4 Process of Clock/Voltage Glitching: Loop Skip Fault Attack Using CWLite

Clock and voltage glitching are effective hardware fault injection techniques used to disrupt normal device behavior by introducing brief disturbances in timing or power. With tools like the ChipWhisperer, these attacks can be precisely controlled and analyzed to identify and exploit vulnerabilities in embedded systems.

The Loop Skip Fault Attack od AES128C aims to bypass certain instructions in a loop, often used in cryptographic operations like key checking or encryption rounds. By injecting a carefully timed glitch, an attacker can force the program to skip one or more loop iterations, potentially weakening the algorithm or leaking secret information. This type of fault is particularly useful when the attacker wants to reduce the number of encryption rounds or skip password verification logic, leading to unintended access or data leakage.

3.4.1 Target Configuration Setup

The first step is configuring the ChipWhisperer environment to match the target device. This involves specifying the correct hardware platform, communication protocol, and cryptographic target. These settings ensure that the glitching process can interact properly with the device under test.

Below is the basic configuration for setting up the CWLite with an ARM-based target running a TinyAES encryption implementation:

```
# Use the OpenADC for data capture
SCOPETYPE = 'OPENADC'

# Target is ChipWhisperer-Lite with ARM MCU
PLATFORM = 'CWLITEARM'

# Target firmware running TinyAES encryption
CRYPTO_TARGET = 'TINYAES128C'

# SimpleSerial version 2.1 for communication
SS_VER = 'SS_VER_2_1'
```

3.4.2 Running the Setup Script

Once the target configuration is defined, the next step is to initialize the ChipWhisperer environment by running a setup script. This script loads all necessary modules, applies configuration settings, and ensures that the scope and target are ready for communication and glitching.

The following command is used to execute the generic setup script:

```
%run "../../Setup_Scripts/Setup_Generic.ipynb"
```

This script sets up the scope, target, and communication interface automatically. It simplifies the initialization process by applying standard settings required for most experiments, allowing the user to focus on customizing parameters for the specific attack.

3.4.3 Compiling the Target Firmware

After setting up the environment, the firmware running on the target device must be compiled to match the chosen platform and cryptographic implementation. This step ensures that the device is loaded with the correct code, such as TinyAES, and is compatible with the glitching setup.

The following command compiles the firmware using the specified platform, crypto target, and SimpleSerial version:

```
%%bash -s "$PLATFORM" "$CRYPTO_TARGET" "$SS_VER"
cd ../../../firmware/mcu/simpleserial-aes
make PLATFORM=$1 CRYPTO_TARGET=$2 SS_VER=$3
```

This Bash cell changes the directory to the AES firmware source and runs the ‘make‘ command with the appropriate arguments. The output is a firmware binary that will be loaded onto the target for testing the glitching attack.

3.4.4 Loading the Target Firmware

Once the firmware is compiled, it needs to be loaded onto the target device. This step ensures that the device is running the correct code for the glitching attack. The following command loads the compiled firmware onto the target:

```
fw_path = "../../../firmware/mcu/simpleserial-aes/" +
          "simpleserial-aes-CWLITEARM.hex"

cw.program_target(scope, prog, fw_path)
```

CHAPTER 4

Real-world replication of state-of-the-art

The Advanced Encryption Standard (AES) has stood resilient for decades, becoming a cornerstone in modern cryptographic systems. However, side-channel attacks and fault analysis have emerged as powerful methods to compromise even the most robust ciphers, not by breaking the algorithm itself, but by exploiting its implementation. One such sophisticated technique is the Diagonal Fault Attack (DFA), a targeted fault analysis method designed to extract cryptographic keys from AES by injecting faults into specific portions of the cipher's internal state.

In this chapter, we reproduce the diagonal fault attack on AES as presented by Dhiman et al. in their 2009 paper titled "A Diagonal Fault Attack on the Advanced Encryption Standard" [SMC09]. Their work introduced a fault model where a single fault injected into a diagonal of the AES state matrix during the final rounds of encryption enables efficient recovery of the secret key.

To validate their proposed attack in a practical setting, we implemented the diagonal fault injection using two physical fault methods: clock glitching and voltage glitching, both facilitated by the ChipWhisperer Lite (CWLite) platform. Our goal is to observe diagonal fault patterns in the faulty ciphertexts and subsequently perform key recovery using the methodology described in the original paper.

This experimental reproduction helps bridge the gap between theoretical fault models and their real-world applicability, emphasizing the feasibility and efficiency of diagonal fault attacks under controlled glitching conditions.

4.1 Four Diagonals in the AES State Matrix

In the AES algorithm, the internal 128-bit state is arranged as a 4×4 matrix of bytes:

Diagonal 0 $[D_0]$

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Diagonal 1 $[D_1]$

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Diagonal 2 $[D_2]$

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Diagonal 3 $[D_3]$

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

In the above matrices, each element a_{ij} represents the entry in the i^{th} row and j^{th} column of a 4×4 state matrix. This state matrix is commonly used in AES (Advanced Encryption Standard) to represent the internal data at various stages of the encryption or decryption process. The highlighted elements correspond to the entries located along one of the four diagonals of the matrix, indexed as D_0, D_1, D_2, and D_3, respectively.

4.2 Diagonal Fault Attack on AES-128 with Fault Injection at Round 8

In AES-128, the encryption process consists of 10 rounds, with each round transforming a 4×4 byte matrix called the state. Each round includes operations such as SubBytes, ShiftRows, MixColumns, and AddRoundKey. Notably, the AES state is updated in a predictable pattern, and faults injected in earlier rounds propagate through subsequent transformations in a structured way. Step-by-Step Fault Propagation:

Fault Introduction (Start of Round 8): A fault is injected into one or more bytes of one or more diagonals in the AES state matrix. This occurs just after the `AddRoundKey` operation of Round 7 and before the `SubBytes` transformation of Round 8.

SubBytes (Non-linear step): Each byte in the state, including any faulty bytes, undergoes substitution through the AES S-box. This non-linear transformation alters the faulty bytes unpredictably. However, at this stage, the fault remains localized to the originally affected bytes.

ShiftRows (Byte reordering): In this step, each row of the state matrix is cyclically shifted by a specific offset. As a result, the previously localized fault bytes—initially within a single diagonal—are now redistributed into different columns. This marks the beginning of spatial fault propagation.

MixColumns (Diffusion step): This transformation applies a fixed matrix multiplication over a finite field to each column of the state. Consequently, a single faulty byte in any column results in the corruption of all four bytes within that column. At this stage, the fault diffuses significantly across the state.

AddRoundKey : The transformed (and now faulty) state is XORed with the Round 8 key. The fault remains embedded and is now more spread out across the matrix.

Rounds 9 and 10: The corrupted state continues to evolve through the remaining AES rounds. Round 9 further propagates the fault via `SubBytes`, `ShiftRows`, and `MixColumns`. However, Round 10 omits the `MixColumns` step, causing the fault pattern to stabilize. The final faulty ciphertext thus reflects this structured propagation, which can be analyzed to extract internal state information.

4.3 Fault Injection at the Start of Round 8 using CW-Lite

To analyze the behavior of AES under fault conditions, a fault can be intentionally introduced at the beginning of Round 8 using the ChipWhisperer-Lite (CW-Lite) platform. Two common techniques supported by CW-Lite for fault injection are voltage glitching and clock glitching. These methods allow precise control over when and how a fault is introduced into the target device running AES.

4.3.1 Using Clock Glitching

Firstly we have identified a probable location(5080 to 5120 samples) of the starting of 8th round from the power trace of the AES-128 running in the integrated target of CWLite-ARM. Then we have set the parametsers for voltage glitching as follows:

```
scope.glitch.clk_src = "clkgen"
```

```
scope.glitch.output = "clock_xor"
scope.glitch.trigger_src = "ext_single"
scope.glitch.repeat = 1
scope.io.hs2 = "glitch"
```

Result

A glitch was introduced at location 5086 samples, which corresponds to the start of Round 8. The parametsers

```
scope.glitch.offset= 10
scope.glitch.width= 3
```

were responsible for this fault injection. This resulted obtaining a faulty ciphertext b'56 9b 6f 66 c9 41 96 f7 9c f1 49 44 29 13 04 e4' where the correct ciphertext without any fault would be b'f5 d3 d5 85 03 b9 69 9d e7 85 89 5a 96 fd ba af'.

Fault Propagation & Analysis

We have one correct ciphertext and one faulty ciphertext after the fault injection. As the key was already known to us we have compared the two ciphertexts to analyze the fault propagation using the AES-128 Decryption method. The following tables show the state of the AES matrix at various stages of the encryption process, both with and without the fault.

After The Fault Injection At Starting of 8^{th} Round :

Without Fault

96	e9	e9	3c
71	87	61	89
6a	91	04	13
e4	c7	90	ff

With Fault

96	e9	e9	3c
71	a3	61	89
6a	91	04	13
e4	c7	90	ff

Difference

00	00	00	00
00	24	00	00
00	00	00	00
00	00	00	00

After The Completion of 8^{th} Round(SubByte,ShiftRow,MixColumn,AddRoundkey(8)) :

Without Fault

0c	b7	3b	ad
77	b8	a0	c3
31	0a	19	d8
43	b0	70	eb

With Fault

2b	b7	3b	ad
4d	b8	a0	c3
2c	0a	19	d8
5e	b0	70	eb

Difference

27	00	00	00
3a	00	00	00
1d	00	00	00
1d	00	00	00

After The Completion of 9^{th} Round(SubByte,ShiftRow,MixColumn,AddRoundkey(9)) :

Without Fault				With Fault				Difference			
c2	10	a5	54	dc	52	13	6e	1e	42	b6	3a
df	31	da	c0	d0	73	1b	ec	0f	42	c1	2c
67	79	42	5d	68	bf	35	4b	0f	c6	77	16
9b	74	40	fa	8a	f0	f6	ec	11	84	b6	16

After The Completion of 10^{th} Round(SubByte,ShiftRow,AddRoundkey(10)) :

Without Fault (ct)				With Fault (ct_f)				Difference			
f5	03	e7	96	56	c9	9c	29	a3	ca	7b	bf
d3	b9	85	fd	9b	41	f1	13	48	f8	74	ee
d5	69	89	ba	6f	96	49	04	ba	ff	c0	be
85	9d	5a	af	66	f7	44	e4	e3	6a	1e	4b

Here we notice that the fault was injected at 1 bytes of the D_0 diagonal of the state matrix of 8th round. According to the fault model proposed by Dhiman et al.,[SMC09] the fault in the D_0 diagonal of the state matrix at the start of Round 8 propagates through the subsequent rounds, affecting the final ciphertext. The differences in the ciphertexts before and after the fault injection reveal how the fault has altered specific bytes, which can be exploited to recover parts of the secret key. If we consider Byte inter-relations at the end of ninth round corresponding to D_0 diagonal, we can express the following relation:

After 8th Round				After 9th Round				After 10th Round Shift Row			
f_1				$2f_1$	f_4	f_3	$3f_2$	$2f_1$	f_4	f_3	$3f_2$
f_2				f_1	f_4	$3f_3$	$2f_2$	f_4	$3f_3$	$2f_2$	f_1
f_3				f_1	$3f_4$	$2f_3$	f_2	$2f_3$	f_2	f_1	$3f_4$
f_4				$3f_1$	$2f_4$	f_3	f_2	f_2	$3f_1$	$2f_4$	f_3

If we represent the 10^{th} round key as K_{10}, it can be expressed as:

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

Now we can frame 3 equations based on the differences in the ciphertexts before and after the fault injection to guess the key bytes K_{00},K_{13},K_{22} and K_{31} . The equations are as follows:

$$\mathsf{ISB}(\mathsf{ct}[0] \oplus K_{00}) \oplus \mathsf{ISB}(\mathsf{ct}_f[0] \oplus K_{00}) = \mathsf{mul2}\,(\mathsf{ISB}(\mathsf{ct}[13] \oplus K_{13}) \oplus \mathsf{ISB}(\mathsf{ct}_f[13] \oplus K_{13}))$$
$$\mathsf{ISB}(\mathsf{ct}[13] \oplus K_{13}) \oplus \mathsf{ISB}(\mathsf{ct}_f[13] \oplus K_{13}) = \mathsf{ISB}(\mathsf{ct}[10] \oplus K_{22}) \oplus \mathsf{ISB}(\mathsf{ct}_f[10] \oplus K_{22})$$
$$\mathsf{ISB}(\mathsf{ct}[7] \oplus K_{31}) \oplus \mathsf{ISB}(\mathsf{ct}_f[7] \oplus K_{31}) = \mathsf{mul3}\,(\mathsf{ISB}(\mathsf{ct}[13] \oplus K_{13}) \oplus \mathsf{ISB}(\mathsf{ct}_f[13] \oplus K_{13}))$$

Similarly for keybytes K_{01},K_{12},K_{23} and K_{30} we can frame the following equations:

$$\mathsf{ISB}(\mathsf{ct}[11] \oplus K_{32}) \oplus \mathsf{ISB}(\mathsf{ct}_f[11] \oplus K_{32}) = \mathsf{mul2} \cdot (\mathsf{ISB}(\mathsf{ct}[4] \oplus K_{01}) \oplus \mathsf{ISB}(\mathsf{ct}_f[4] \oplus K_{01}))$$
$$\mathsf{ISB}(\mathsf{ct}[1] \oplus K_{10}) \oplus \mathsf{ISB}(\mathsf{ct}_f[1] \oplus K_{10}) = \mathsf{ISB}(\mathsf{ct}[4] \oplus K_{01}) \oplus \mathsf{ISB}(\mathsf{ct}_f[4] \oplus K_{01})$$
$$\mathsf{ISB}(\mathsf{ct}[14] \oplus K_{23}) \oplus \mathsf{ISB}(\mathsf{ct}_f[14] \oplus K_{23}) = \mathsf{mul3} \cdot (\mathsf{ISB}(\mathsf{ct}[4] \oplus K_{1}) \oplus \mathsf{ISB}(\mathsf{ct}_f[4] \oplus K_{1}))$$

For keybytes K_{02},K_{11},K_{20} and K_{33} we can frame the following equations:

$$\mathsf{ISB}(\mathsf{ct}[2] \oplus K_{20}) \oplus \mathsf{ISB}(\mathsf{ct}_f[2] \oplus K_{20}) = \mathsf{mul2} \cdot (\mathsf{ISB}(\mathsf{ct}[8] \oplus K_{02}) \oplus \mathsf{ISB}(\mathsf{ct}_f[8] \oplus K_{02}))$$
$$\mathsf{ISB}(\mathsf{ct}[15] \oplus K_{33}) \oplus \mathsf{ISB}(\mathsf{ct}_f[15] \oplus K_{33}) = \mathsf{ISB}(\mathsf{ct}[8] \oplus K_{02}) \oplus \mathsf{ISB}(\mathsf{ct}_f[8] \oplus K_{02})$$
$$\mathsf{ISB}(\mathsf{ct}[5] \oplus K_{11}) \oplus \mathsf{ISB}(\mathsf{ct}_f[5] \oplus K_{11}) = \mathsf{mul3} \cdot (\mathsf{ISB}(\mathsf{ct}[8] \oplus K_{02}) \oplus \mathsf{ISB}(\mathsf{ct}_f[8] \oplus K_{02}))$$

Similarly for keybytes K_{03},K_{10},K_{21} and K_{30} we can frame the following equations:

$$\mathsf{ISB}(\mathsf{ct}[9] \oplus K_{12}) \oplus \mathsf{ISB}(\mathsf{ct}_f[9] \oplus K_{12}) = \mathsf{mul2} \cdot (\mathsf{ISB}(\mathsf{ct}[6] \oplus K_{21}) \oplus \mathsf{ISB}(\mathsf{ct}_f[6] \oplus K_{21}))$$
$$\mathsf{ISB}(\mathsf{ct}[3] \oplus K_{30}) \oplus \mathsf{ISB}(\mathsf{ct}_f[3] \oplus K_{30}) = \mathsf{ISB}(\mathsf{ct}[6] \oplus K_{21}) \oplus \mathsf{ISB}(\mathsf{ct}_f[6] \oplus K_{21})$$
$$\mathsf{ISB}(\mathsf{ct}[12] \oplus K_{03}) \oplus \mathsf{ISB}(\mathsf{ct}_f[12] \oplus K_{03}) = \mathsf{mul2} \cdot (\mathsf{ISB}(\mathsf{ct}[6] \oplus K_{21}) \oplus \mathsf{ISB}(\mathsf{ct}_f[6] \oplus K_{21}))$$

Here in AES, `mul2` and `mul3` denote multiplication by 2 and 3 in the finite field $\mathrm{GF}(2^8)$, used in the MixColumns step to achieve diffusion through field arithmetic. Solving these 12 equations allows us to recover the key bytes of 10th Round and thus the complete key of AES-128.

4.3.2 Using Voltage Glitching

Firstly we have identified a probable location(5090 to 5120 samples) of the starting of 8th round from the power trace of the AES-128 running in the integrated target of CWLite-ARM. Then we have set the parametsers for voltage glitching as follows:

```
scope.glitch.clk_src = "clkgen"
scope.glitch.output = "glitch_only"
scope.glitch.trigger_src = "ext_single"
scope.io.glitch_lp = True
scope.io.glitch_hp = True
```

Result

A glitch was introduced at location 5100 samples, which corresponds to the start of Round 8. The parametsers

```
scope.glitch.offset= -37.890625
scope.glitch.width= 37.109375
```

were responsible for this fault injection.

Analysis and Fault Propagation

We have one correct ciphertext and one faulty ciphertext after the fault injection. As the key was already known to us we have compared the two ciphertexts to analyze the fault propagation using the AES-128 Decryption method. The following tables show the state of the AES matrix at various stages of the encryption process, both with and without the fault.

After The Fault Injection At Starting of 8^{th} Round :

Without Fault

96	e9	e9	3c
71	87	61	89
6a	91	04	13
e4	c7	90	ff

With Fault

96	e9	e9	52
52	87	61	89
6a	91	04	13
e4	c7	90	ff

Difference

00	00	00	6e
23	00	00	00
00	00	00	00
00	00	00	00

After The Completion of 8^{th} Round(SubByte,ShiftRow,MixColumn,AddRoundkey(8)) :

Without Fault

0c	b7	3b	ad
77	b8	a0	c3
31	0a	19	d8
43	b0	70	eb

With Fault

0c	b7	3b	9e
77	b8	a0	75
31	0a	19	90
43	b0	70	6e

Difference

00	00	00	33
00	00	00	b6
00	00	00	48
00	00	00	85

After The Completion of 9^{th} Round(SubByte,ShiftRow,MixColumn,AddRoundkey(9)) :

Without Fault

c2	10	a5	54
df	31	da	c0
67	79	42	5d
9b	74	40	fa

With Fault

b4	11	6b	73
a9	32	a7	5e
fd	7b	f1	c3
77	75	f3	43

Difference

76	01	ce	27
76	03	7d	9e
9a	02	b3	9e
ec	01	b3	b9

After The Completion of 10^{th} Round(SubByte,ShiftRow,AddRoundkey(10)) :

Without Fault

f5	03	e7	96
d3	b9	85	fd
d5	69	89	ba
85	9d	5a	af

With Fault

5d	4b	9e	39
37	b2	67	b0
58	0b	58	2d
b2	7c	55	ab

Difference

a8	48	79	af
e4	0b	e2	4d
8d	62	d1	97
37	e1	0f	04

Here we notice that the fault was injected at 2 bytes of the D_3 diagonal of the state matrix of 8th round. According to the fault model proposed by Dhiman et al.,[SMC09] the fault in the D_3 diagonal of the state matrix at the start of Round 8 propagates through the subsequent rounds, affecting the final ciphertext.According to The Byte inter-relations corresponding to D_3 diagonal, we can express the following relation:

After 8th Round

			f_1
			f_2
			f_3
			f_4

After 9th Round

f_4	f_3	$3f_2$	$2f_1$
f_4	$3f_3$	$2f_2$	f_1
$3f_4$	$2f_3$	f_2	f_1
$2f_4$	f_3	f_2	$3f_1$

After 10th Round Shift Row

f_4	f_3	$3f_2$	$2f_1$
$3f_3$	$2f_2$	f_1	f_4
f_2	f_1	$3f_4$	$2f_3$
$3f_1$	$2f_4$	f_3	f_2

Again we can make the following equations based on the differences in the ciphertexts with or without the fault injection to guess the key bytes K_{00},K_{13},K_{22} and K_{31} . The equations are as follows:

$$\mathsf{ISB}(\mathsf{ct}[0] \oplus K_{00}) \oplus \mathsf{ISB}(\mathsf{ct}_f[0] \oplus K_{00}) = \mathsf{ISB}(\mathsf{ct}[13] \oplus K_{13}) \oplus \mathsf{ISB}(\mathsf{ct}_f[13] \oplus K_{13})$$
$$\mathsf{ISB}(\mathsf{ct}[10] \oplus K_{22}) \oplus \mathsf{ISB}(\mathsf{ct}_f[10] \oplus K_{22}) = \mathsf{mul3} \cdot (\mathsf{ISB}(\mathsf{ct}[13] \oplus K_{13}) \oplus \mathsf{ISB}(\mathsf{ct}_f[13] \oplus K_{13}))$$
$$\mathsf{ISB}(\mathsf{ct}[7] \oplus K_{31}) \oplus \mathsf{ISB}(\mathsf{ct}_f[7] \oplus K_{31}) = \mathsf{mul2} \cdot (\mathsf{ISB}(\mathsf{ct}[13] \oplus K_{13}) \oplus \mathsf{ISB}(\mathsf{ct}_f[13] \oplus K_{13}))$$

Similarly for keybytes K_{01},K_{10},K_{23} and K_{32} we can frame the following equations:

$$\begin{aligned}
\mathsf{ISB}(\mathsf{ct}[11] \oplus K_{32}) \oplus \mathsf{ISB}(\mathsf{ct}_f[11] \oplus K_{32}) &= \mathsf{ISB}(\mathsf{ct}[4] \oplus K_{01}) \oplus \mathsf{ISB}(\mathsf{ct}_f[4] \oplus K_{01}) \\
\mathsf{ISB}(\mathsf{ct}[1] \oplus K_{10}) \oplus \mathsf{ISB}(\mathsf{ct}_f[1] \oplus K_{10}) &= \texttt{mul3}\,(\mathsf{ISB}(\mathsf{ct}[11] \oplus K_{32}) \oplus \mathsf{ISB}(\mathsf{ct}_f[11] \oplus K_{32})) \\
\mathsf{ISB}(\mathsf{ct}[14] \oplus K_{23}) \oplus \mathsf{ISB}(\mathsf{ct}_f[14] \oplus K_{23}) &= \texttt{mul2}\,(\mathsf{ISB}(\mathsf{ct}[4] \oplus K_{01}) \oplus \mathsf{ISB}(\mathsf{ct}_f[4] \oplus K_{01}))
\end{aligned}$$

For keybytes K_{02},K_{11},K_{20} and K_{33} we can frame the following equations:

$$\begin{aligned}
\mathsf{ISB}(\mathsf{ct}[8] \oplus K_{02}) \oplus \mathsf{ISB}(\mathsf{ct}_f[8] \oplus K_{02}) &= \texttt{mul3}\,(\mathsf{ISB}(\mathsf{ct}[2] \oplus K_{20}) \oplus \mathsf{ISB}(\mathsf{ct}_f[2] \oplus K_{20})) \\
\mathsf{ISB}(\mathsf{ct}[15] \oplus K_{33}) \oplus \mathsf{ISB}(\mathsf{ct}_f[15] \oplus K_{33}) &= \mathsf{ISB}(\mathsf{ct}[8] \oplus K_{02}) \oplus \mathsf{ISB}(\mathsf{ct}_f[8] \oplus K_{02}) \\
\mathsf{ISB}(\mathsf{ct}[5] \oplus K_{11}) \oplus \mathsf{ISB}(\mathsf{ct}_f[5] \oplus K_{11}) &= \texttt{mul2}\,(\mathsf{ISB}(\mathsf{ct}[2] \oplus K_{20}) \oplus \mathsf{ISB}(\mathsf{ct}_f[2] \oplus K_{20}))
\end{aligned}$$

For keybytes K_{03},K_{10},K_{21} and K_{30} we can frame the following equations:

$$\begin{aligned}
\mathsf{ISB}(\mathsf{ct}[9] \oplus K_{12}) \oplus \mathsf{ISB}(\mathsf{ct}_f[9] \oplus K_{12}) &= \mathsf{ISB}(\mathsf{ct}[6] \oplus K_{21}) \oplus \mathsf{ISB}(\mathsf{ct}_f[6] \oplus K_{21}) \\
\mathsf{ISB}(\mathsf{ct}[3] \oplus K_{30}) \oplus \mathsf{ISB}(\mathsf{ct}_f[3] \oplus K_{30}) &= \texttt{mul3}\,(\mathsf{ISB}(\mathsf{ct}[6] \oplus K_{21}) \oplus \mathsf{ISB}(\mathsf{ct}_f[6] \oplus K_{21})) \\
\mathsf{ISB}(\mathsf{ct}[12] \oplus K_{03}) \oplus \mathsf{ISB}(\mathsf{ct}_f[12] \oplus K_{03}) &= \texttt{mul2}\,(\mathsf{ISB}(\mathsf{ct}[6] \oplus K_{21}) \oplus \mathsf{ISB}(\mathsf{ct}_f[6] \oplus K_{21}))
\end{aligned}$$

4.4 Conclusion

In this experiment, we successfully demonstrated the use of CWLite for voltage glitching and Clock glitching to inject faults into the AES-128 encryption process. By analyzing the fault propagation through the rounds of AES, we were able to derive equations that allowed us to recover key bytes from the faulty ciphertext as mentioned in [SMC09]. These attacks can also be perfomed with the help of CWHusky with slight modification in the setting by making

```
scope.glitch.clk_src = "pll"
```

and then finding a suitable glitch location, width and offset. This highlights the vulnerabilities of cryptographic systems to physical attacks and emphasizes the importance of implementing robust countermeasures against such threats.

CHAPTER 5

Flipping Bits to Break BipBip

CHAPTER 6

Preparing PQC for Fault Analysis: A Kyber Implementation

About the Author

Aenean scelerisque. Fusce pretium porttitor lorem. In hac habitasse platea dictumst. Nulla sit amet nisl at sapien egestas pretium. Nunc non tellus. Vivamus aliquet. Nam adipiscing euismod dolor. Aliquam erat volutpat. Nulla ut ipsum. Quisque tincidunt auctor augue. Nunc imperdiet ipsum eget elit. Aliquam quam leo, consectetuer non, ornare sit amet, tristique quis, felis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque interdum quam sit amet mi. Pellentesque mauris dui, dictum a, adipiscing ac, fermentum sit amet, lorem.

Bibliography

[New25a] NewAE Technology Inc. *ChipWhisperer-Lite*, 2025. 8

[New25b] NewAE Technology Inc. *ChipWhisperer-Nano Target Board*, 2025. 7

[New25c] NewAE Technology Inc. *CW1173 ChipWhisperer-Lite Capture Documentation*, 2025. 7

[New25d] NewAE Technology Inc. *Scope API — ChipWhisperer Documentation*, 2025. 13

[SMC09] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. A diagonal fault attack on the advanced encryption standard. *IACR Cryptol. ePrint Arch.*, 2009:581, 2009. 23, 27, 30, 31