Lab 04: SCA Attacks-Statistical Analysis

Ganesh Veluru and Jwala Sri Hari Badam
University of South Florida
Tampa, FL 33620

I. Introduction

In the world of cybersecurity, Side Channel Analysis (SCA) is a sophisticated technique that plays a critical role in understanding and addressing potential security risks in computer systems. It revolves around the intriguing concept that electronic devices unintentionally emit subtle signals or leakages while performing various tasks. These leakages can manifest as fluctuations in power consumption, electromagnetic radiation, or even variations in heat emissions. These seemingly innocuous signals can be a goldmine for attackers who possess the right knowledge, as they can exploit these side channels to uncover sensitive information. This information may include details about the mathematical operations being executed by the hardware, potentially granting unauthorized access to critical data or system functionalities [1].

In this lab exploration, we embark on a journey to delve into the realm of SCA and its far-reaching implications. Our primary tool for this endeavor is the ChipWhisperer (CW) Nano platform, a potent resource for developing custom firmware for microcontrollers. As we run mathematical operations, these operations inadvertently leave behind traces in the form of power consumption patterns. These traces serve as a treasure trove of information, and by meticulously scrutinizing them using statistical techniques, particularly the Welch's t-test, we aim to conduct a Test Vector Leakage Assessment (TVLA). This assessment helps us identify potential information leaks and vulnerabilities that could compromise the security of electronic systems. Furthermore, we seek to gain a deeper understanding of how the power traces vary when the device performs different mathematical operations on the same data or the same operation on different data. Through these hands-on investigations, our objective is to fortify computer systems against the ever-looming threat of SCA attacks while unraveling the mysteries hidden within these intricate digital pathways [2].

II. Reading Check

Question 1: What are some examples of physical parameters that an attacker can leverage in an SCA attack?

Answer: In Side Channel Analysis (SCA) attacks, attackers exploit various physical parameters unintentionally leaked by electronic devices during their operation. Some of the main physical parameters that attackers can leverage shown in Fig 1.

Power Consumption: Devices draw varying amounts of electrical power as they perform operations. Analyzing power consumption patterns can reveal information about internal processes, cryptographic keys, or processed data [3]. Electromagnetic Emissions (EMI): Electronic devices emit electromagnetic radiation during operation. Capturing and analyzing EMI can provide insights into a device's behavior, potentially revealing sensitive information or cryptographic keys.

Timing Analysis: Variations in the timing of a device's responses to inputs or instructions can indicate differences in executed operations. Attackers use timing analysis to uncover hidden information.

Heat (Thermal Analysis): Devices generate heat during operation, and temperature variations can indicate specific activities. Thermal analysis allows attackers to detect and exploit temperature-related side channel information.

Acoustic Emanations (Sound): Some devices produce audible sounds as they operate, conveying information about their internal state. Attackers use acoustic analysis to gather insights into device behavior. These physical parameters serve as inadvertent sources of information leakage, offering attackers opportunities to gain insights into a device's inner workings, potentially leading to security breaches or data compromise.

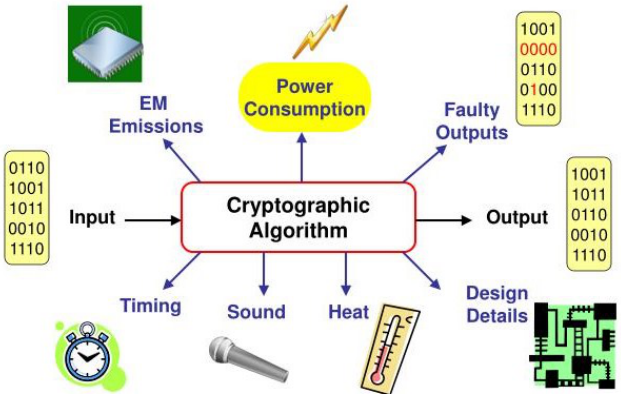

Fig. 1: Physical Parameters in Side Channel Analysis

Question 2: Why does an unprotected electronic device (one without any countermeasures in place) leak information through side channels?

Answer: Unprotected electronic devices inadvertently leak information through side channels primarily because of the fundamental physical characteristics of their operations. These characteristics include the switching of transistors, data transmission along bus lines, power consumption fluctuations, electromagnetic emissions, and even the generation

of heat and sound. Each of these activities is a natural consequence of a device's functioning. Importantly, these operations are not perfectly uniform across different tasks or data inputs. For instance, performing complex mathematical calculations typically consumes more power and generates different electromagnetic radiation patterns than simpler operations like addition or multiplication. These inherent variations in physical characteristics can inadvertently transmit valuable information about what operations the device is executing and, in some cases, the specific data it is processing [4].

Moreover, in unprotected devices, there may be inadequate isolation between different components and circuits, allowing side-channel information to propagate beyond its intended boundaries. Additionally, the lack of countermeasures, such as noise injection or data obfuscation, further exacerbates the problem. Side-channel analysis attacks take advantage of the relatively clean and discernible nature of these physical signals, making it possible for attackers to extract meaningful patterns and gain insights into the device's inner workings. To mitigate these vulnerabilities, countermeasures are introduced to reduce the predictability and exploitability of side-channel information, bolstering the security of electronic systems.

Question 3: What does the p-value represent in a statistical hypothesis test, such as the t-test?

Answer: Welch's t-test is a statistical hypothesis test used to determine if there is a significant difference between the means of two sets of data, particularly when the variances of these sets are unequal. In the context of the provided information, Welch's t-test is utilized to assess whether there is a significant difference in the information leakage (in this case, power consumption) between two sets of data obtained during side-channel analysis.

Hypothesis Testing: Hypothesis testing is a statistical technique used to determine whether there is a significant difference between two groups or populations. In the context of the project, these two groups might represent different sets of power traces obtained during side-channel analysis.

Null Hypothesis (H0): This hypothesis is the default assumption that there is no significant difference between the two groups. In the project, it could be stated as "There is no significant difference in information leakage (power consumption) between the two sets of power traces."

Alternative Hypothesis (Ha): This is the opposing hypothesis that suggests there is a significant difference between the two groups. In the project, it might be expressed as "There is a significant difference in information leakage between the two sets of power traces."

Significance Level (α): The significance level, denoted as (α), represents the probability threshold used to assess the results of the hypothesis test. It determines how strong the evidence must be to reject the null hypothesis. In most cases, a common choice is a 5% significance level (α = 0.05), indicating a 5% probability of rejecting a true null hypothesis. This corresponds to a 95% confidence level, meaning there is a 95% probability of not rejecting a true null hypothesis.

P-value: The p-value is a crucial metric in hypothesis testing. It quantifies the probability of observing the results (or more extreme results) if the null hypothesis is true. In other words, it provides a measure of how likely it is to obtain the observed data under the assumption that there is no significant difference between the groups.

Interpretation of P-value: When $P <= \alpha$ (e.g., $P <= 0.05$), it suggests that the observed results are statistically significant. In the context of the project, if the p-value is less than 0.05, it indicates that there is a statistically significant difference in information leakage between the two sets of power traces.

When $P > \alpha$ (e.g., $P > 0.05$), it implies that there is no statistical significance, and the observed results are likely due to chance. In the project, if the p-value is greater than 0.05, it suggests that there is no statistically significant difference in information leakage [5],

III. Methods

In this experiment, we are utilizing ChipWhisperer (CW) Nano board., as depicted in Fig 2. The CW Nano board operates in two main sections: "TARGET" and "CAPTURE." The "TARGET" section emulates the victim system or hardware under analysis, running its code. In contrast, the "CAPTURE" section acts as an oscilloscope, capturing power traces from the victim system's operation. It includes an ADC to convert analog power signals to digital and communicates with a PC via USB for power trace analysis. We have a PC and PHS- virtual machine at our disposal, and within this virtual environment, we are executing scripts to gather power traces.

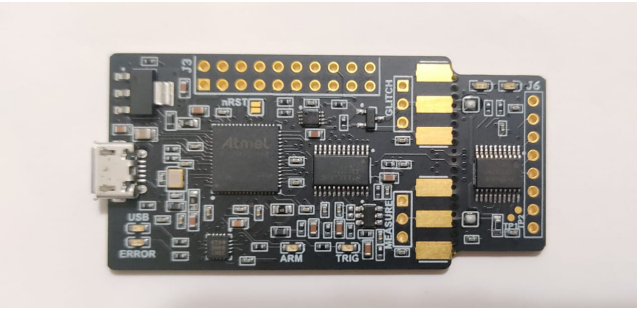

Fig. 2: ChipWhisperer Nano Board

A. Hardware and Software Setup

Our procedure for Hardware and Software setup will take the following steps:

Step 1: Connect CW Nano board-After starting the PHS-VM, connect the CW Nano board to our PC via USB. To confirm that the device is properly connected, we can navigate to "Device" and "USB" in the VM. We should see "NewAE Technology Inc. ChipWhisperer Nano

[0900]" listed if the connection is established correctly. If not, we should double-check our hardware, software installations, and connections. The complete setup of our experiment is shown in the Fig 3.

Step 2: Access Jupyter Notebook-To start our investigation, we can open a web browser on our host machine and enter "http://localhost:8888/." This will allow us to access Jupyter Notebook locally. Inside Jupyter Notebook, we'll find various directories, and we should focus on the "PHS-Lab-04" directory.

Step 3: Unzip PHS-Lab-04 Content-Within the "PHS-Lab-04" folder, we can upload the provided "PHS-Lab-04.zip" file. Run the unzip code located in the "RUN-TO-ZIP.ipynb" file to extract the contents of "PHS-Lab-04.zip." Make sure that the name of the zipped file matches the current directory for successful extraction.

Step 4: Organizing the Extracted Content - After unzipping the provided archive, we will observe the folder structure illustrated in the accompanying figure.

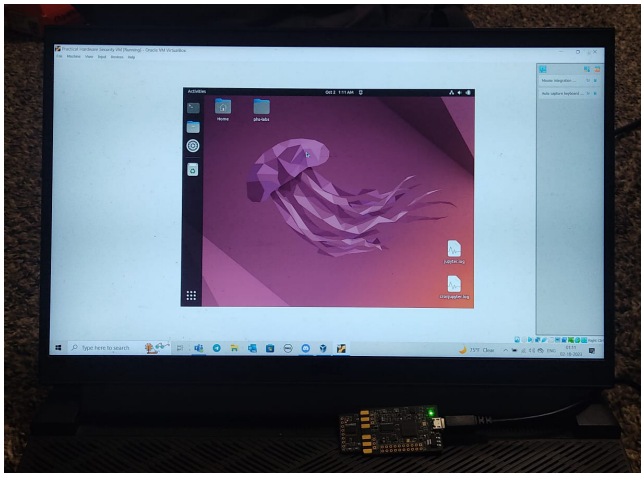

Fig. 3: Hardware and Software setup

The directory is structured as follows:

```
PHS-Lab-04
├── Firmware
│   ├── makefile
│   ├── tvla-test-CWNANO.hex
│   └── tvla-test.c
├── Python
│   └── Lab-04-SCA-Statistical-Analysis.ipynb
└── Figures
```

a) Firmware: In the "cw-base-setup/simpleserial-base" directory, duplicate the files "simpleserial-base.c" and the makefile, then transfer them to your "PHS-Lab-04/Firmware" folder. Within the "PHS-Lab-04/Firmware" directory, rename the copied "simpleserial-base.c" file to "tvla-test.c" and make the necessary adjustments to the makefile. In tvla-test.c we need to write the functions/victim code.

b) Python: Inside the Python folder we have SCA-Statistical-Analysis.ipynb where we are going to write the scripts related to collecting the power traces by interfacing with Firmare code.

c) Figures: Figures generated during the analysis phase, such as plots and graphical representations, will be stored in the "Figures" directory.

B. Experimental Procedure

Part A: Firmware

Our procedure for Part A will take following steps:

Step 1: In tvla-test.c we need to write 3 functions namely tvla_test_A, tvla_test_B and tvla_test_C containing logic to perform product, xor and modular multiplication respectively on the list passed.

Step 2: In all these functions, Initially, the system receives a 16-byte array consisting of 8-bit unsigned integers. Subsequently, it marks the commencement (trigger_high()) and culmination (trigger_low()) points for data acquisition, despite the absence of actual data collection in this scenario. Lastly, the system employs the simpleserial interface to transmit the unaltered array pt back to the user, allowing the user to access this information from the ChipWhisperer.

Step 3: In the main(), The process begins by initializing and configuring both the ChipWhisperer (CW) and simpleserial for seamless communication with the CW. Subsequently, each function is linked to a unique single-character command using the simpleserial_addcmd syntax (e.g., simpleserial_addcmd('a', 16, tvla_test_A)). Finally, the system awaits user input in the form of commands ('a' for tvla_test_A 'b' for tvla_test_B) to execute the corresponding function.

Part B: Python

Step 1: we will start by importing the required python libraries and setting up the chip whisperer board by initializing scope and target sections.

Step 2: program the target with the hex file which comes after executing the makefile.

Step 3: reset the target device connected to the ChipWhisperer Nano (CW Nano) board. Resetting the target is essential in certain scenarios, especially when conducting experiments or tests that require a clean or specific starting state for the target device.

Step 4: To verify that the written functionality in the target is working or not. Send the list to target and read back the output to verify the results.

Step 5: Next we need to create a function to get the power traces from the scope section of CW nano board.

The function begins by resetting the target device, ensuring a clean and consistent starting state for data collection. The data_in as a bytearray is sent to the CW target device along with a specified command (cmd) using target.simpleserial_write(). The function retrieves the power trace data from the CW scope using scope.get_last_trace(). The code snippet for this is shown in 4

```python
# Function: get_pwr_trace()
#   Input:  'data_in'  - input list of 16 positive integers from 0 to 255
#   Input:  'cmd'      - input character command to select the respective function from the CW
#   Output: 'trace'    - power trace collected from the CW as a Numpy Array
def get_pwr_trace(data_in: list, cmd: str) -> np.ndarray:
    # reset target and read start-up text
    reset_target()


    # arm the CW scope to be ready to capture power traces
    scope.arm()

    # convert data_in list to bytearray
    b = bytearray(data_in)

    # send the data_in as bytearray to the CW function with the respective 'cmd' parameter
    target.simpleserial_write(cmd,b)

    # start the scope to capture power traces
    ret = scope.capture()
    if ret:
        print("timeout")
    # get the power trace from the CW
    trace = scope.get_last_trace()

    # return the power trace
    return trace
```

Fig. 4: Python script to get power traces from scope

Step 6: we need to collect 32 power traces for 2 datasets for each function tvla_test_A, tvla_test_B and tvla_test_C. Convert the trace list to numpy arrays and average (sample-wise) the traces. The code snippet to collect power traces for data-1 by setting the target function tvla_test_A is shown in 5. Similarly we need to collect power traces for other functions tvla_test_B and tvla_test_C.

```python
# get list of test data
test_a1 = random_prod_list()

print(test_a1)
# initialize variables
d1ta_all = []
d1ta_avg = []

# for each of the 32 iterations, send the same first test data to tvla_test_A
# append this to a list
for i in range(32):
    d1ta_all.append(get_pwr_trace(test_a1, 'a'))


# convert trace list to numpy array
d1ta_all=np.array(d1ta_all)

# average the traces (sample-wise)
d1ta_avg =  np.mean(d1ta_all,axis=1)
print(d1ta_avg)
print(len(d1ta_all))
```

Fig. 5: Python script to collect power traces by sending data

Step 7: Now we have averaged power traces where the length of the list is 32 for 2 datasets for all A,B,C.

TABLE I: Mean and SD values for Averaged traces

Dataset	Average	Std
d1ta_A	-0.187	0.001
d2ta_A	-0.187	0.001
diff_tr_A	0.0	0.002
d1tb_B	-0.178	0.001
d2tb_B	-0.178	0.001
diff_tr_B	0.0	0.001
d1tc_C	-0.165	0.001
d2tc_C	-0.167	0.001
diff_tr_C	0.002	0.001

To compute the differential trace for A, subtract averaged data1 trace and data2 trace for A. Similarly we need to calculate differential trace for B,C. For all these averaged traces we need to compute mean and standard deviation and the result is shown in Table II.

Step 8: Now we have total of 9 Lists, three for tvla_test_A namely d1ta_avg, d2ta_avg and diff_tr_a and similarly for tvla_test_B and tvla_test_C. To plot these values we are using pyplot by taking power values on y-axis and sample number on x-axis for all 9 averaged datasets. The code snippet for plotting the data is shown in Fig 6

```python
# plot traces using pyplot
plt.figure(figsize=(5.5, 3.5), constrained_layout=True)

# plots the data
plt.plot(d1ta_avg, color="#F84B27")

# format your plots
plt.title("TVLA Test A: Data 1 Trace")  # adds title
plt.xlabel("Sample Number")  # adds x-axis label
plt.ylabel("Power")          # adds y-axis label
# ... and more ...

# saves the plot
plt.savefig("../Figures/d1ta_A.pdf")
plt.savefig("../Figures/d1ta_A.png", dpi=500)

# show the plot on your screen
plt.show()
```

Fig. 6: Python script to plot the Averaged power trace values

The plots drawn for all averaged datasets are shown in **Fig 7 to 11.**

IV. Results

We need to perform welch's test to know any potential inform leakage using the power traces. Using python scipy scipy.stats.ttest_ind() we can get t and p values. We have to pass original pair of 32 traces for data-1 and data-2 to the function and it return t and p values. Next we will calculate

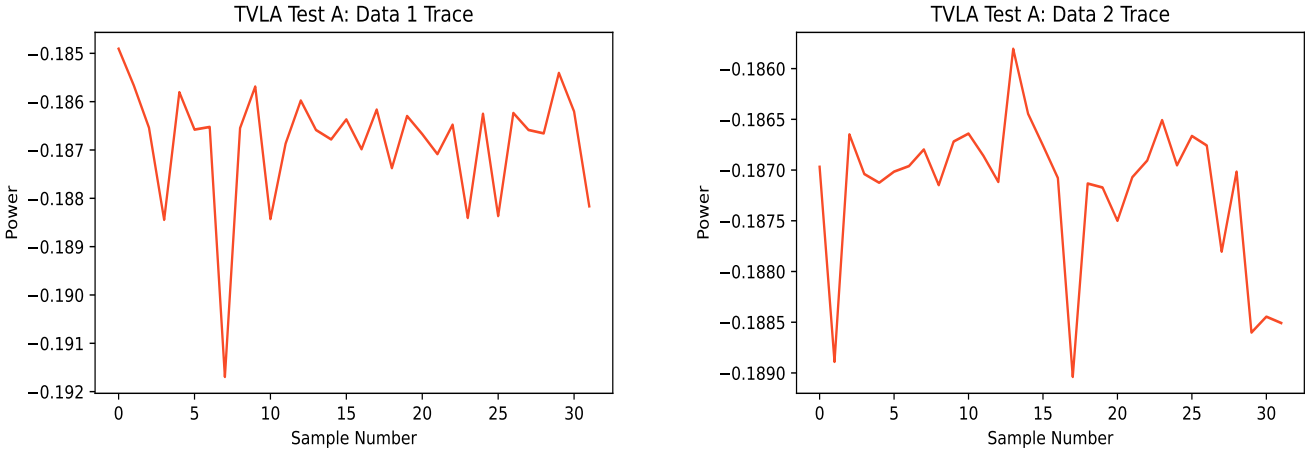

Fig. 7: Line Graph for TVLA Test-A Averaged Power Trace

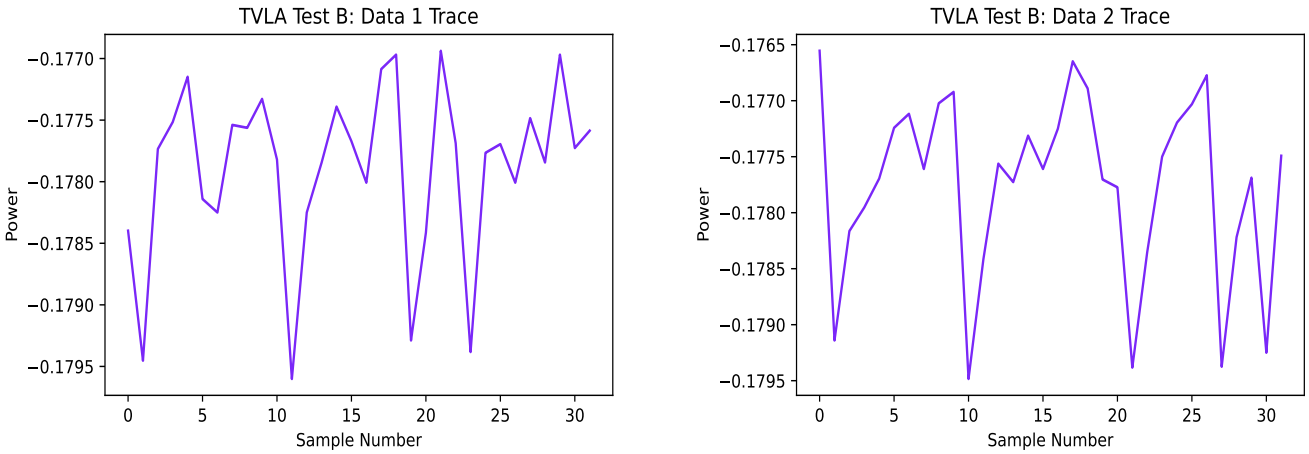

Fig. 8: Line Graph for TVLA Test-B Averaged Power Trace

TABLE II: Statistical Results of Welch's Test

Dataset	t-Average	t-std	p-statistical significance(%)
Trace A	0.2206	1.7743	18.2
Trace B	8.695	84.15	96.4
Trace C	1.50	46.71	80.0

the mean and standard deviation for the t-values followed by plotting of t-values.

The code snippet for the above is shown in Fig 12

To know the difference in set of 2 power traces we will compute the percentage of p-values that are less than or equal to significance level. The code snippet to calculate the percentage is shown in Fig 13 This process needs to be repeated for A,B and C data.

The mean and standard deviation of t-values along with percentage of statistical significance(p-values) is shown in Table II

The plots for t-values for all traces are shown in **Fig 14 to 16**.

V. Discussion

We conducted three distinct tests, labeled as tvla_Test_A (product operation), tvla_Test_B (XOR operation), and tvla_Test_C (modular multiplication). These tests provided valuable insights into how these operations might reveal sensitive information through side-channel leakage.

In tvla_Test_A, where we evaluated the product operation, we obtained a statistical significance percentage of 18%. This indicates that approximately 18% of the collected data exhibited side-channel variations significant enough to raise security concerns. While this percentage may seem relatively low, it is important to note that even seemingly simple operations like multiplication can introduce subtle variations in power consumption, potentially revealing confidential information.

In tvla_Test_B, which focused on XOR operations, demonstrated a notably higher statistical significance of 96%. This result indicates a significantly greater susceptibility to side-channel attacks. XOR operations involve bitwise manipulation of data, and this nature makes them prone to inducing distinct power fluctuations. The high statistical significance aligns with

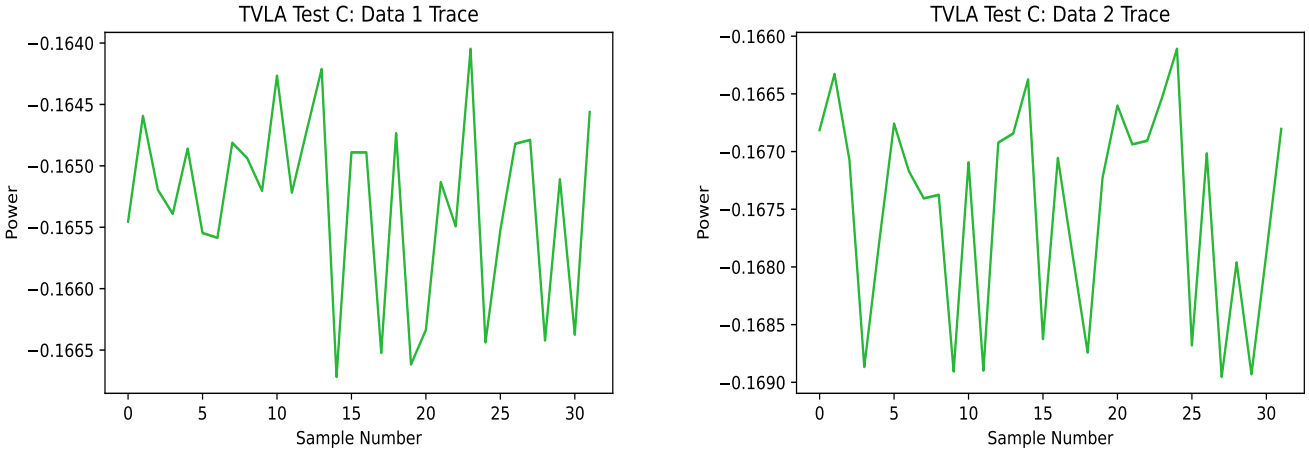

Fig. 9: Line Graph for TVLA Test-C Averaged Power Trace

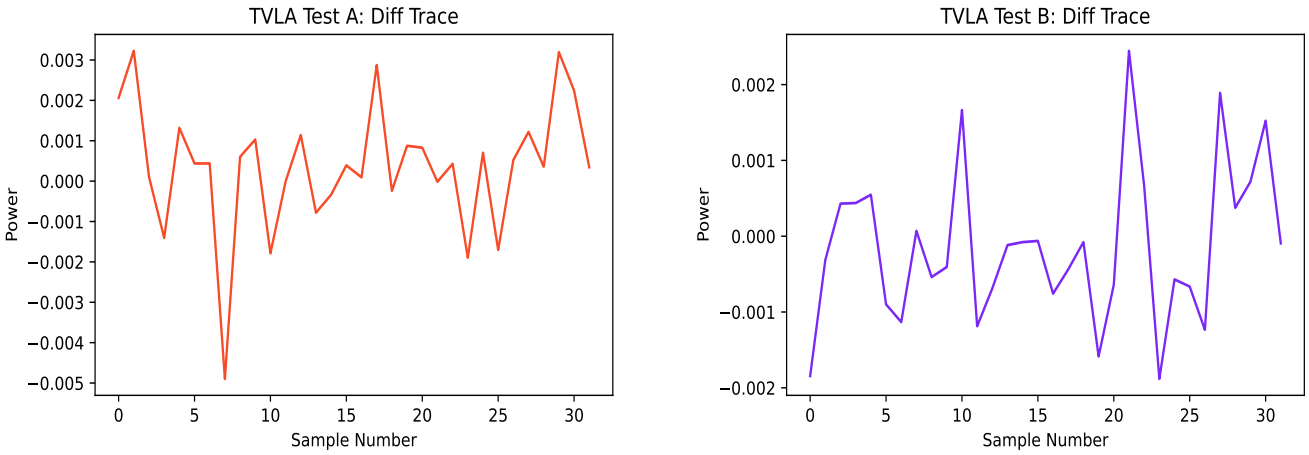

Fig. 10: Line Graph for Differential Trace of TVLA Test-A and TVLA Test-B

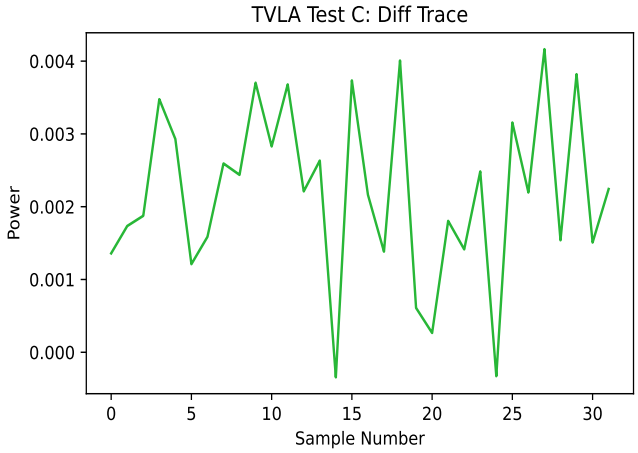

Fig. 11: Line Graph for Differential Trace of TVLA Test-C

```
ttest_a = scipy.stats.ttest_ind(d1ta_all, d2ta_all, equal_var=False)
tval_a = ttest_a[0]
pval_a = ttest_a[1]


print(tval_a)
print(pval_a)
```

```
# get mean and standard deviation for t values
tval_a_mean = np.nanmean(tval_a)
tval_a_std = np.nanstd(tval_a)

print("Trace A t-value | Mean:",tval_a_mean)
print("Trace A t-value | Std:",tval_a_std)
```

Fig. 12: Python script to get t and p values for traces

expectations, highlighting the importance of securing systems employing XOR operations.

The third test, tvla_Test_C, concentrated on modular multiplication, revealing a statistical significance of 80%. This suggests that the operation has a relatively high potential for side-channel attacks. Modular multiplication involves complex mathematical calculations, and as such, it generates noticeable

```python
# plot traces using pyplot
plt.figure(figsize=(5.5, 3.5), constrained_layout=True)

# plots the data
plt.plot(tval_a[0:250], color="#F84B27")

# format your plots
plt.title("Trace A: Welch's t-test (t-values)")  # adds title
plt.xlabel("Sample Number")  # adds x-axis label
plt.ylabel("t-value")          # adds y-axis label
# ... and more ...

# saves the plot
plt.savefig("../Figures/ttesta.pdf")
plt.savefig("../Figures/ttesta.png", dpi=500)

# show the plot on your screen
plt.show()
```

Fig. 13: Python script to draw line graph for t-values

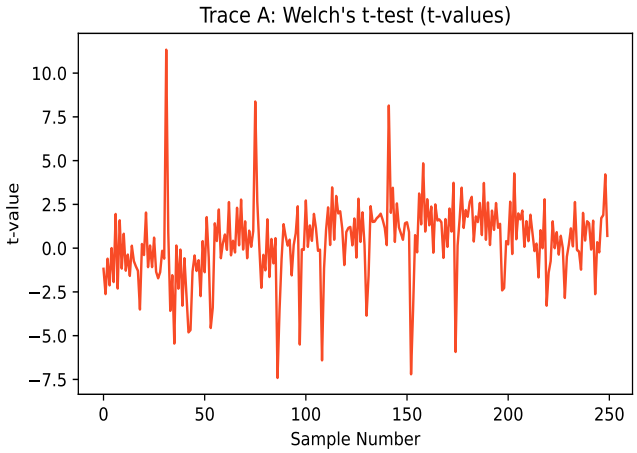

Fig. 14: Line Graph showing t-values for Trace-A

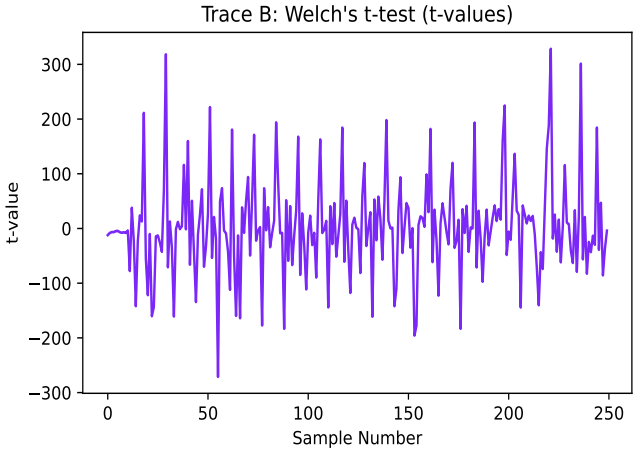

Fig. 15: Line Graph showing t-values for Trace-B

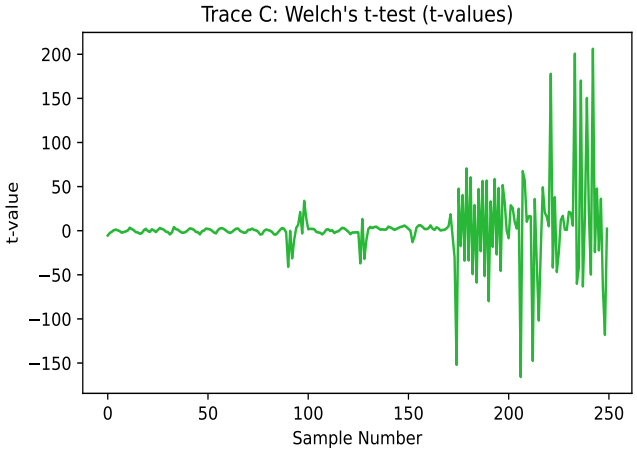

Fig. 16: Line Graph showing t-values for Trace-C

variations in power consumption. These results affirm the notion that more intricate operations can be attractive targets for attackers seeking to exploit side-channel vulnerabilities.

VI. Conclusion

In conclusion, this lab provided valuable insights into the realm of side channel attacks and their significance in cybersecurity. We learned that attackers can exploit various physical parameters such as power consumption, electromagnetic emissions, timing, heat, and sound to glean sensitive information from electronic devices. Moreover, we gained a comprehensive understanding of the functionality of the CW Nano board, including its target and capture sections, which play a crucial role in capturing power traces for analysis.

One of the key takeaways from this lab was the practical application of statistical hypothesis testing, particularly Welch's t-test, to assess the significance of differences between power traces. We discovered how to rigorously analyze these traces and determine whether there is a substantial distinction between them, which is vital for identifying vulnerabilities and enhancing security. Looking ahead, the exploration of machine learning techniques in the context of side channel attacks could be an intriguing avenue for further research. Machine learning has the potential to provide more advanced and precise analyses, allowing us to uncover even subtler information leakage from electronic devices. Overall, this lab has equipped us with valuable skills and knowledge in the realm of side channel analysis, opening doors to deeper investigations into the evolving landscape of cybersecurity.

References

[1] R. Karam, S. Katkoori, and M. Mozaffari-Kermani, "Lecture Note 7:Side Channel Analysis/Signals and SCA," in *Practical Hardware Security Course's Lecture Notes*. University of South Florida, Sep 2023.

[2] ——, "Lecture Note 7: Side Channel Analysis/Side-channel Leakage," in *Practical Hardware Security Course's Lecture Notes*. University of South Florida, Sep 2023.

[3] ——, "Lecture Note 7:Side Channel Analysis/Power attacks," in *Practical Hardware Security Course's Lecture Notes*. University of South Florida, Sep 2023.

[4] ——, "Experiment 4: Side Channel Analysis Attacks/Introduction," in *Practical Hardware Security Course Manual*. University of South Florida, Sep 2023.

[5] ——, "Experiment 4: Side Channel Analysis Attacks/Background," in *Practical Hardware Security Course Manual*. University of South Florida, Sep 2023.