CSAW 2021 Final Phase Report

Konstantinos S. Mokos
Department of Informatics
University of Piraeus
Piraeus, Greece
kostasmkuni@gmail.com

Dimitrios Lazarakis
Department of Digital Systems
University of Piraeus
Piraeus, Greece
dlazarak@gmail.com

Ilias Koudounas
Department of Informatics
University of Piraeus
Piraeus, Greece
elias.koudounas@gmail.com

Paschalis Kyranoudis
Department of Informatics
University of Piraeus
Piraeus, Greece
paschalis@kyranoudis.com

***Abstract*—This document outlines the response to the challenges published for the final phase of CSAW 2021 Embedded Security Challenge. In particular this report consists of a presentation of general conclusions drawn from the analysis of all the challenge sets, the methodology approach followed by our team and research performed for each task. The report is structured in 3 main sections, introduction, methodology and finally challenges where we provide the necessary evidence along with the steps followed towards the solution for each challenge with a brief description of them.**

***Index Terms*—Side Channel Attack, Fault Injection, Timing Attacks , Cryptanalysis**

I. Introduction

Our team, according to the published instructions, used the ChipWhisperer Nano device and its target to load the provided challenges to perform Side Channel Analysis and fault injections to complete each task. By knowing the capabilities of the device we were able to tailor the attack methodology to our device and utilize Pattern Recognition, Signal Analysis and functions provided by the chipwhisperer library [1] alongside powerful open-source libraries to attack each challenges.

As a consequence of the pandemic physical contact was limited , so we were not able to meet in person therefor we searched for efficient ways to perform virtual meetings during the full span of the competition. Our remote meeting setup included a Discord Server and Raspberry Pi with the CW-Nano attached to it as we will see in Section, II

II. Methodology

In order to maximize the efficiency of our team we crafted a methodology to approach each challenge to perform a thorough examination of each problem at hand and ensure that we have identified all the possible exploitation ways and selected the most optimal in each case. This process can be seen in the Figure 1.

Python was our primary choice for interacting with the device and performing the attacks required due to the extensive API provided by chipwhisperer. Furthermore we used a plethora of libraries and tools that provides a great integration with the ChipWhisperer line of products.

Our solutions and scripts were developed and tested using Python 3.x in a Jupyter Notebook environment. These scripts where utilized on every aspect of the event from remote programming the firmware to the target device to validate the each solution.

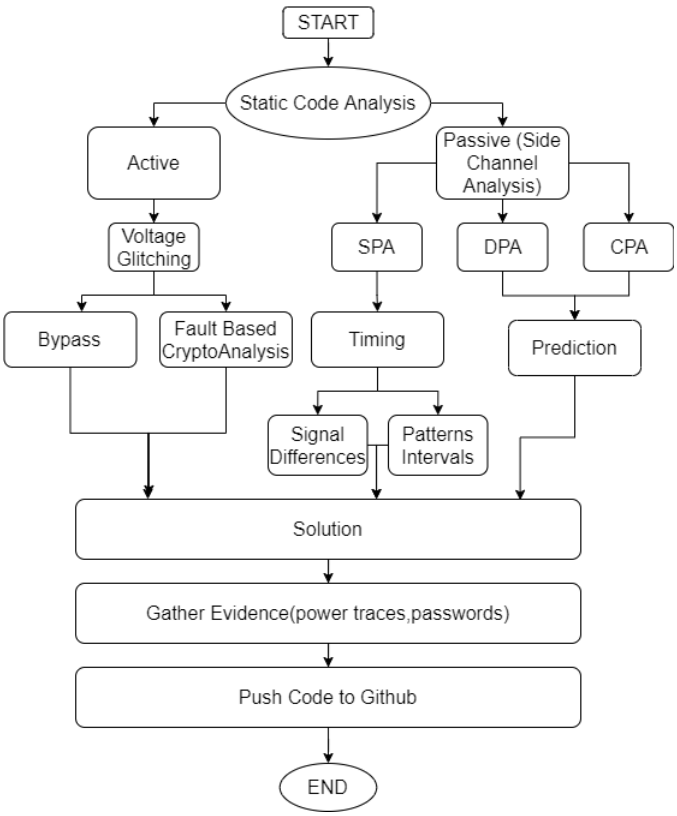

Fig. 1: Methodology Flow

Despite all the planning process, we faced some difficulties while performing the attacks under the latest version of the chipwhisperer's library (5.5.2) and we had to settled on version 5.5.0 for our test lab which was the latest stable library for the Nano board. Finally a major part of our attack process was the visualization of the captured power traces, we performed signal analysis and **visual pattern recognition** by utilizing plot.ly, an open source plotting library. **Plot.ly** was a massive aid since we could interact with the graph and analyze the points with ease giving us valuable information about each challenge to help us understand its inner workings.

It is worth mentioning that even tho we did have the capability on performing profiled attacks, since he had access to the source code and the target board, the complexity of

the challenges and the time constrain deemed these types of attacks inefficient.

A. Remote Lab

As mentioned already from the early stages of the final phase we strived to create a remote lab, where each member of our team could gain access to the provided board (CW Nano) with the same ease of physical access in order to execute code, load firmware and the attack the victim device. At first we tried a combination of **Wireguard** and **usbip** in order to create a tunnel from the USB port to the each team members machine. While the device was recognized by the integrated *scope()* method, any resource intensive operation such as firmware flashing and capturing power traces was prone to timeouts. To mitigate that, we installed Jupyter on the remote device where the CW Nano was connected and use the VPN to access the interface. The device that facilitated both the Wireguard and Jupyter is a **Raspberry pi 3B+** which is a low cost computer that enable us to have a dedicated device for for the sole purpose of accessing the Nano board. We also created an API to perform certain operations on the chipwhisper such as device reset and diagnostics since due to security concerns only a web interface was accessible via the pi board. Moreover to increase security of the device exposed to the internet we restricted access to the LAN via the VPN with the use of **iptables**. A full lab layout can be seen in the following diagram.

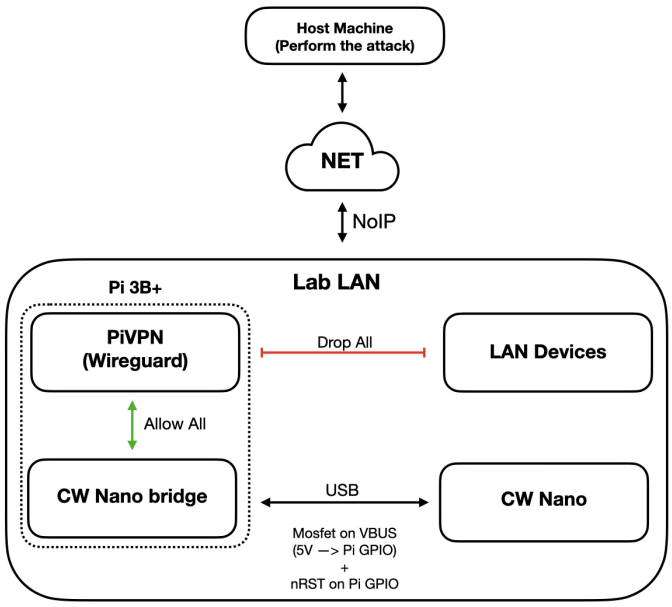

Fig. 2: Remote Lab Diagram

B. Coding Framework

While moving through the challenges of the final phase we released that we tended to reuse some specific code snippets for functions such as flashing the device, analysing the signals, plotting and saving the captured traces. This lead to the creation of a set of wrappers (callable functions) to automate specific processes . Investing time into this particular task proved beneficial since it increases the productivity and work flow during the development of our solutions.

III. Challenges

The section is comprised of three subsection, one per challenge set provided during the competition. Each of the subsections contains a summary box for each challenge accompanied by a challenge description consisting of a brief explanation of the code, the vulnerability in it and the exploitation method followed to achieve each challenges goal.

It is worth noting that during testing on the device we were obtaining inconsistent results in the captured traces which resulted in some of our solutions to not be reproducible without additional coding. Since a few of the challenges where solvable by a simple SPA attack and our primary objective was to achieve a goal in each challenge we decided not to spend the additional time required to further generalize our solvers.

A. Set 1

Fizzy - 150

```
Algorithm: Bubble sort
Attack type: SPA timing attack
Challenge solution:
TIMINGSIDECHANNELSARESOCOOL
```

`Challenge Description:`

This challenge is using the firmware to sort an array using the Bubble sort algorithm. We are tasked to find the original unsorted array which is hardcoded into the compiled code.Bubble sort repeatedly iterates over the list, compares adjacent elements and calls the swap function if they are in a wrong order. This algorithmic process is reversible if we have the information of total number of swaps and in what iteration each took place. The provided source code indicates that when swap() is called a snippet of code that performs 10 multiplications occurs. Those multiplications are clearly visible as power spikes in the power trace, that is captured while the micro-controller executes the algorithm. This enables us to convert the signal to a ”digital” format and eliminate any unnecessary information or noise from it by keeping as logical high (1) the spikes that indicate the multiplications and logical low (0) for the rest of the trace. By extracting timing information in regards of the intervals between each swap we can recover the required information to perform a reverse Bubble short and un-sort the array.

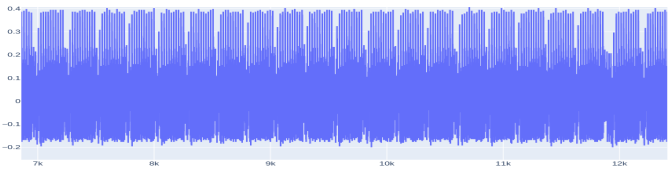

Fig. 3: Traces captured demonstrating the power difference when a swap happens.

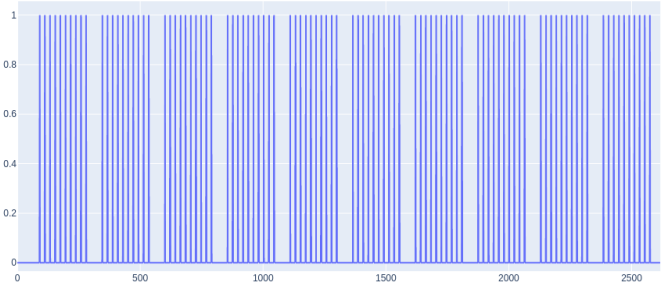

Fig. 4: Traces captured demonstrating the digitization.

recall - 50

```
Algorithm: Array Comparison
Attack type: Timing attack
Challenge solution:
p0w3R1skn0wl3dg3
```

Challenge Description:

This challenge implements the classic password check functionality where it compares our input to a predefined array. As we can observe in the provided source code the verify() is vulnerable to a timing attack since its dependant on the hardcoded value and not the user provided input.

The code iterates over the correct password until it reaches an incorrect comparison between the hard-coded value and the user input, thus enabling us to reduce the attack surface to a single character (256 bytes) at a time as opposed to the classic and time consuming brute-force attack. Upon successful verification of one byte the function moves to the next which can be clearly seen in the corresponding power-traces for each check. By setting a reference frame on each stage and comparing the difference of the absolute values of each sample point in our traces we can extract timing information that can lead to successful recovery of each byte leading up to the full 16 byte secret value, since successful verification of one byte is indicated by a longer compare time the patterns is also visible via a simple visual inspection as seen in Figure 5.

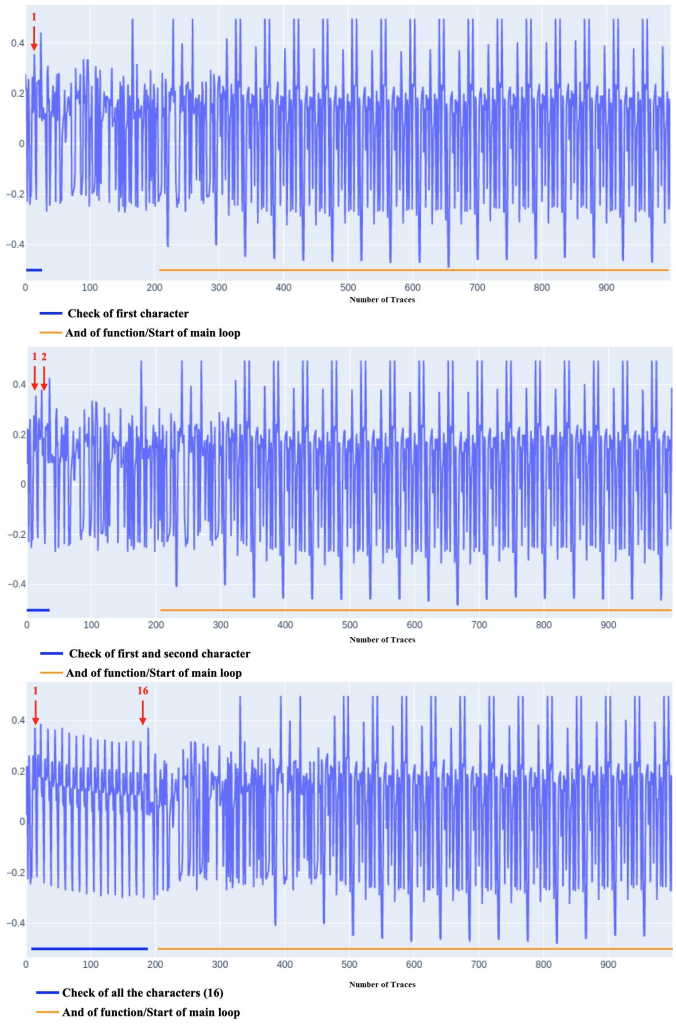

Fig. 5: Traces captured demonstrating when no characters are correct, when 1 character is correct and when 16 are correct.

err0r - 50

```
Algorithm: CRC32 calculation and
comparison
Attack type: Voltage Glitching
Challenge solution:
[Fault Injection Bypass]
```

Challenge Description: The target calculates the CRC32 hash of the provided input two times and then compares them. The goal is for the comparison to output False, meaning that the two hashes should differ in value. This can be achieved by inserting a fault while one of the two calculations takes place. The only way to perform the fault is by utilizing voltage glitching features by the CW Nano attack board. Since the complexity is low and it does not require a precise insertion of a fault we can simple iterate over a reasonable amount of offsets (time to wait after the trigger) and number of faults until a successful completion. Since randomly faults might

brick the firmware on the target board setting it into an IDLE state we might have to reset the target during our attempts.

CRT - 100

```
Algorithm: RSA with CRT
Attack type: Voltage Glitching
Challenge solution:
p: 962476599190059883, q:
1084024262488859977
```

Challenge Description:

This challenge implements an RSA Signature service with the Chinese Remainder Theorem optimization. The goal is to recover p and q primes. Several papers have researched this topic the one that we focused our attention on was the following [2] [3].

Based on the article, a public key that implements the signature may be tricked into revealing its secret key, if the following three conditions are met:

- the signed message is known;
- a certain type of faulty behaviour occurs during signature generation.
- the device outputs the faulty signature.

Consequently, we can control all these conditions because we can send a message and get the signature also we can inject a fault to the device. The exploitation is based on the modexp() that include the CRT exponents, one using p and the latter using q. If an error happens in the calculation of one of the two, the final signature produced, which contains a mathematical expression involving both exponents, can reveal information. Since one of the operations is corrupted the final outputs mathematical properties will change and it will only be a multiple of one of the two primes (the non corrupted one) which enables us to use the Greatest Common Divisor of the signature and N to recover it. Once we have one of the two primes, let it be p, we can recover the latter one via a simple division N/p = q. We can perform voltage glitching to insert the fault in a similar manner as on **recall**. To ensure that we get a faulty signature we also need a reference point thus we will first perform the operation without inserting a fault and then repeat the process while performing voltage glitching with the same input.

The RSA using the Chinese Remainder Theorem consists of these equations:

$$S_p = m^{d_p} mod p$$

$$S_q = m^{d_q} mod q$$

$$S = CRT(S_p, S_q) = S_q + q((S_p - S_q) I_q mod p)$$

In order to be able to factorize the output signature and recover the p and q primes either *Sp* or *Sq* must be corrupted.

B. Set - 2

casino - 100

```
Algorithm: Password Verification
Attack type: SPA Timing attack
Challenge solution:
[120, 90, 80, 60, 110, 10, 50, 20,
30, 40, 150, 140, 70, 100, 130]
```

Challenge Description: The goal of this particular challenge is to recover the secret array. The secret array is called in the function draw(). This function contains 2 loops, one for cycling the array and one for multiplying a counter. The latter takes place as many times as the value of each array element. Since the multiplication leaves visible power spikes as is the most power-consuming action in the set of commands being executed, we captured a trace and looked for spikes that correspond to high power consumption. By counting these spikes, analogous to the multiplications performed, we can recover the byte value of the array's elements.

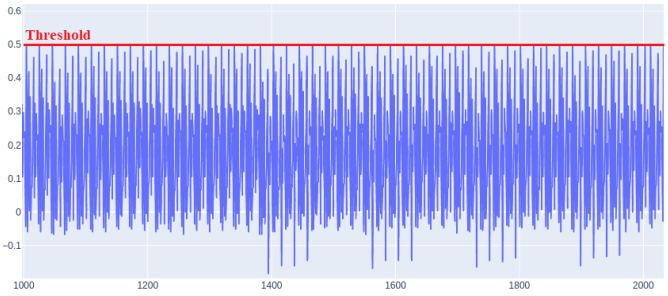

Fig. 6: Trace spikes that correspond to high power consumption.

search - 150

```
Algorithm: Binary Search
Attack type: Timming attack
Challenge solution:
[59, 91, 118, 152, 207, 232]
```

Challenge Description: The firmware is running the binary search algorithm, that takes place at least after 6 elements have been removed. The goal is to identify the missing elements. Binary search has a worst time complexity of O(logn) which is observed if the search term is not present in the list. With that in mind we can iterate over all the elements of the list and perform a search and monitor the execution time via the power consumption for each term. The ones with the largest execution time are the missing elements. To further simplify our SPA we can have a single pointer of the spiking power consumption right after the visible search patterns in the trace.

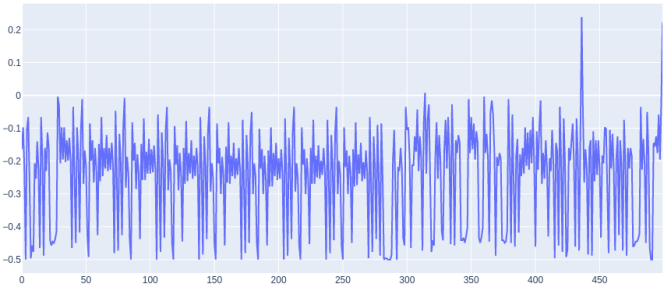

Fig. 7: Traces that demonstrate the power consumption of the binary search algorithm.

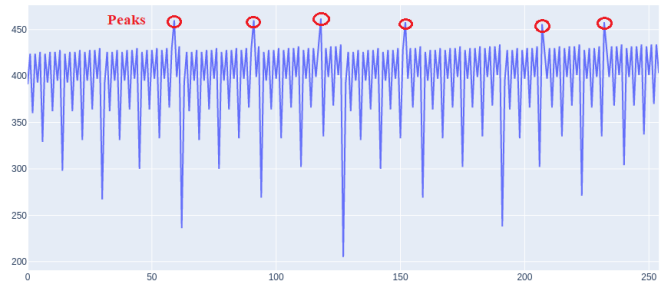

Fig. 8: Finalized power peaks that demonstrate the numbers sorted.

FIAsco - 150

```
Algorithm: AES
Attack type: CPA attack
Challenge solution:
[2b 7e 15 16 28 ae d2 a6 ab f7 15 88
09 cf 4f 3c]
```

Challenge Description:

This challenge implements the algorithm AES-128. AES is one of the most attacked cryptographic algorithms in regards to side channel attacks, so there is plethora of approaches to breaking a firmware implementation of it. For the scope of the challenges we decided to perform a CPA attack since it is one of the most efficient ways to recover the private key. The leakage in the power consumption of the device can lead to uncovering critical information about the operations that the device is performing if we use the necessary statistical analysis methods.

The main objective of this type of attack is to produce an accurate power model that describes the power consumption of the embedded device under normal operation (during the encryption process). After we obtain such a model with the use of a Hamming weight function we will need to find correlations between the predicted output and the actual power consumption which will indicate in a probabilistic manner what the cipher key might be. Moreover since the encryption process is happening per byte, we can further simplify the attack surface by attacking the key one byte at a time and selecting each byte based on the highest correlation probability between our power model and the actual power consumption of the device. In order to better understand the objective of the attack we must dive into the internals of the AES algorithm and how it is implemented. During the encryption process of AES two main operations are performed which are of interest in our case, *AddRoundKey* and *SubBytes*. The substitution phase (SubBytes) uses the Substitution Box (Sbox) to substitute each byte to its corresponding value on the lookup table (Sbox) and it is the only non linear transformation performed in AES. Since we know the sbox matrix and have control over the plaintext value used for encryption we can perform a CPA attack targeting this operation to retrieve information about the key from each byte operation, reducing dramatically the attacks time complexity (compared to brute-force and other traditional approaches).

This is possible due to the physical properties of the microcontrollers memory and the way its operating. In order to alter the state of a bit in memory the microcontroller consumes a certain amount of power for each of the two possible conditions *logic LOW* and *logic HIGH*. The latter one consumes more power since the devices needs to alter the state from Low to High and this power difference is measurable through the devices power consumption. Therefore we can calculate the Hamming Weight Power Model in order to represent the total number of alteration that happened to the memory. Hamming weight units are then compared to the actual voltage levels of power traces captured when a device was performing cryptographic operations. This act of comparison is performed by calculating the Pearson's Correlation Coefficient for each pair of modelled power values and the actual power values (samples) consumed by the device.

C. Set - 3

The following subsection outlines a general analysis of the challenges contained in set 3. Due to limited amount of time our team did not have the time to implement the attacks and exploitation methods that will be described for each challenge.

calc - 150

```
Algorithm: Math operations
Attack type: CPA
Challenge solution: -
```

Challenge Description:

The challenge as the names suggests is a form of low level calculator which is performing a set of basic mathematical operations to a secret array given a user input (byte). Since we are able to collect traces while each of the operations is performed, there is a high likelihood that a Correlation Power Analysis will be effective in recovery information in regards of the secret value in a similar manner as on the Fiasco challenge. A few adjustments on the preexisting code from

mentioned challenge did not yield any promising results and further parameterization of the code was prohibited due to time constrains.

homebrew - 150

```
Algorithm: Custom encryption
algorithm
Attack type: Timing/CPA
Challenge solution: -
```

Challenge Description:

The challenge implements a custom encryption algorithm with two main sets of operations being performed based on the bit value of the key at each iteration. Since the second set of operations (when the key bit is equal to 0) is performing two multiplications there is a high possibility that the implementation is susceptible to a timing attack as observed with other challenges. Moreover the algorithm is vulnerable to a CPA attack since correlating the set of operations performed at each cycle and selecting the set with the highest probability will reveal information about the state of each if condition which is directly connected with the corresponding key bit of the secret key, which as a result will leak the key one bit at a time.

NotSoAccessible - 200

```
Algorithm: Simon Cipher
Attack type: Deferential Fault
Analysis (DFA)
Challenge solution: -
```

Challenge Description:

The final challenge presented on the final phase implements the Simon Cipher which is developed my NSA. Based on the structure of the challenge (trigger is set high on round 25) and the provided hints and leaked information it was trivial to identify the intended solution described in the following paper [4]. Due to the complexity of the algorithm and time constrains it was not feasible to develop the algorithm for this attack. Furthermore it is worth noting that with precise calculations and setting the corresponding offsets we might be able to inject a fault at the T-2 round and reduce the complexity of the exploitation method since it is only required to recover the last rounds key byte for the successful completion of the challenge.

REFERENCES

[1] NewAE Technology Inc. Side-Channel analysis tool-chain. kernel description, 2021.

[2] Marc Joye, Arjen Lenstra, and Jean-Jacques Quisquater. Chinese remaindering based cryptosystems in the presence of faults. *Journal of Cryptology*, 12, 11 1998.

[3] Chong Hee Kim and Jean-Jacques Quisquater. Fault attacks for crt based rsa: New attacks, new results, and new countermeasures. *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems Lecture Notes in Computer Science*, page 215–228, 2007.

[4] Duc-Phong Le, Sze Ling Yeo, and Khoongming Khoo. Algebraic differential fault analysis on simon block cipher. *IEEE Transactions on Computers*, 68(11):1561–1572, 2019.