UNIVERSITY OF CALIFORNIA,
IRVINE

Hardware Trojan Detection in FPGA through Side-Channel Power Analysis and Machine Learning

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Electrical and Computer Engineering

by

Salam R. Zantout

Thesis Committee:
Associate Professor Mohammad Al Faruque, Chair
Professor Fadi Kurdahi
Professor Rainer Doemer

2018

DEDICATION

To

my parents and siblings

in recognition of their worth, help and guidance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express the deepest appreciation to my committee chair, Professor Mohammad Al-Faruque, who introduced me to the hardware security field and sparked my interest in it. I would also like to sincerely thank my committee member, Professor Fadi Kurdahi for his constant support and guidance throughout the previous years. In addition, a huge thank you to my committee member, Professor Rainer Doemer for his advice and constructive comments. Without their persistent help and encouragement, this thesis would not have been possible. Finally, I would like to express my gratitude to my colleagues Jiang Wan, Sujit Chhetri, and Hsin-Wei Hung for their continuous mentoring throughout my thesis.

ABSTRACT OF THE THESIS

Hardware Trojan Detection in FPGA through Side-Channel Power Analysis and Machine Learning

By

Salam R. Zantout

Master of Science in Electrical and Computer Engineering

University of California, Irvine, 2018

Professor Mohammad Al Faruque, Chair

The security of a cyber-physical system depends on the safety of the handled data, software, and underlying hardware. Securing the hardware is not a simple task because of the globalization of integrated circuits' manufacturing flow. One hardware attack to be considered is the modification of the design to insert a "backdoor" which maliciously alters the behavior of the original system. Such a malicious and intentional insertion is called a Hardware Trojan Horse (HTH). In this thesis, an HTH detection technique was proposed and implemented. The detection technique made use of side-channel power analysis along with machine learning to detect the presence of an HTH. Power traces from a golden implementation (HTH-free) of the AES encryption algorithm on an FPGA were used to train a logistic regression model. The obtained model was then tested on new power traces collected from the golden implementation and was able to make correct predictions with 95% accuracy. Next, an HTH, of a few gates, was implemented in the AES circuit to carry out a denial-of-service attack along with a breach of plaintext secrecy. The power data from the HTH-infected circuit were collected and tested on the trained logistic regression model. An amount of 81% of the HTH-infected data was detected as flawed by the logistic

regression model allowing the detection of the HTH even when it was not triggered. In fact, even when an HTH was dormant, the HTH would constantly be checking its triggering condition and hence consumed power.

INTRODUCTION

A cyber-physical system integrates both physical and computation processes where embedded computers and networks regulate the physical mechanisms [1]. Cyber physical systems are used in most fields such as medicine, transportation, financial applications, and military. It is such an important task to make sure the systems are secure because they primarily affect human lives. The security of a cyber-physical system depends on the safety of the internal and external data handled, the system's software, and the underlying hardware. A threat to the system's hardware cannot be resolved by a patch or a remote system update. Instead, the compromised hardware would need to be replaced or accepted as a source of fault in the system. Securing the hardware is not a simple task because of the globalization of integrated circuits' (IC) manufacturing flow. In fact, the semiconductor companies are not able to maintain the manufacturing flow locally because of the increasing complexity and cost of the design, fabrication and deployment of IC [2]. Consequently, any participant in the manufacturing flow can be considered as an untrusted entity which increases the probability of attacks. One attack to be considered is the modification of the design to insert a "backdoor" which maliciously alters the behavior of the original system. Such a malicious and intentional insertion is called a Hardware Trojan Horse (HTH). An HTH insertion in the IC is done secretly with the aim of never being noticed. An HTH can have multiple effects such as denial-of-service, leak of classified information, or degradation of performance. In this thesis. an HTH detection technique was proposed and implemented. This technique made use of side-channel power analysis along with machine learning to detect an HTH of a few gates. More precisely, a machine learning model was trained based on the power data collected from a circuit assumed to be HTH-

free (golden circuit). The trained model was then fed new power data from a circuit under test and the model's predictions helped determine whether the circuit at hand was HTH-infected or not.

The remainder of this thesis is organized as follows: Chapter 1 provides theoretical background on HTH and HTH detection methods, then introduces the currently implemented HTH detection techniques. Chapter 2 introduces the experimental work carried out to implement the proposed method of HTH detection using side-channel power analysis. Chapter 3 describes in detail the procedure followed to collect power data and then presents some interpretation of the collected data. Chapter 4 discusses the use of machine learning on the side-channel data and the different iterations performed to optimize the performance of the proposed model. Finally, chapter 5 concludes the work by providing a summary and future work.

CHAPTER 1: Theoretical Background and Related Work

Hardware Trojan Horse

Security of a cyber-physical system depends on the integrity of three main components: the software, the information processed, and the underlying hardware [3]. One of the attacks targeting hardware security is the insertion of a Hardware Trojan Horse (HTH). An HTH is a malicious modification of electronic hardware and can take place at different stages of the hardware's life cycle [3]. Unlike software viruses, malicious modifications to the hardware are not removable after discovery by using patches or updates to the system. A defective hardware part in a system may require a system disassembly and part replacement. Moreover, securing hardware is exceptionally difficult [4] due to the large number of gates or complexity of the design, and the manufacturing variability which results in different physical characteristics of the IC, even among the ICs coming from the same design.

The manufacturing flow of Integrated Circuits (IC) is long and globally distributed which increases the probability of malicious modifications and loss of integrity [5]. The manufacturing flow of an IC is composed of ([6] and [7]):

1- System Specifications: High-level technical requirements indicating the expected capabilities of the design.

2- Design: In the design stage, the intellectual property (IP) providers are selected, and then the functional blocks are acquired from the chosen IP. The acquired functional blocks are then integrated into the register-transfer level (RTL) design to make the final design. RTL is a design abstraction which describes the behavioral functionality of a digital circuit through functional blocks. The description of the circuit is based on the flow of its signals and the logical operations performed on those signals. Next

a gate-level netlist is generated through logic synthesis, where the RTL design is translated into netlists of logic gates. The final step in the design stage is producing the physical layout. The physical layout is composed of geometric representations of the circuit which will guide the manufacturing process.

3- Manufacturing: At this stage the design is converted into wafers of pure semiconducting material. Afterwards, the manufacturing of ICs takes place at one or multiple foundries not necessarily in the same geographic area. Foundries receive already designed ICs, build the ICs, and finally send the manufactured ICs back to the foundry's customers.

4- Assembly and Market: The manufacturing flow ends with the packaging and then distribution of ICs to the several markets including the transportation sector, military organizations, technology companies, financial organizations, etc.

In general, any participant in the manufacturing flow described above, can be an adversary who has the accessibility to insert an HTH, as shown in Figure 1.1. At the design stage, an HTH can be inserted in the IP by the designer or the IPs coming from 3rd-party providers. Hundreds of IP vendors around the world design IP blocks which are then distributed and used in the manufacturing flow [8]. As a result, IP blocks cannot be assumed secure and free of HTHs. At the design stage also, an HTH can be inserted into the design by a computer-aided design (CAD) designer. CAD tools perform synthesis through software and hence an HTH can be injected by this software in ICs. The HTH injection could happen by maliciously altered logic in the CAD tool or in the scripts running the tool [9]. An HTH can also be inserted at the manufacturing stage by reverse engineering since, at this stage, the designers send their layout to an off-shore foundry in Graphic Database System version II

(GDSII) format [10]. The third-party foundry responsible for fabrication, may introduce an HTH in the GDSII.

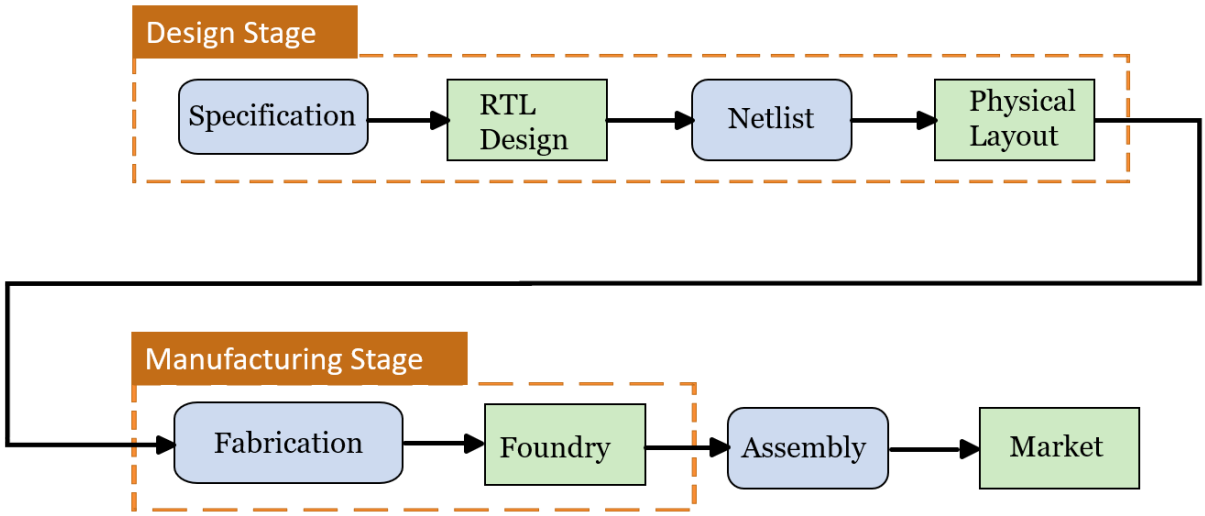

Figure 1.1: Vulnerabilities in Manufacturing Flow

A stealthy HTH is defined as being inserted in the IC secretly with the aim of never being noticed [3]. This property of an HTH makes it more difficult to detect any alteration to the hardware using conventional tests. Once inserted, an HTH remains asleep (or inactive) and is only usually activated after long hours of field operation. An active HTH can be used for denial-of-service, reduction of reliability, privacy breach, compromise or leak of sensitive information, espionage, or alteration of the IC's expected functionality for any other malicious goal [5]. For example, the first real world detection of an HTH in a military grade field-programmable gate array (FPGA), the Actel/Microsemi ProASIC3 chips, was in 2012 [11]. The inserted HTH allowed an adversary to extract configuration data from the chip or reprogram it, access encryption keys and unencrypted configuration bitstream, modify low-level silicon features, or even permanently damage the device. Once an HTH is discovered, the only two options are to either accept that the system can be easily compromised or perform a physical replacement after a redesign of the silicon itself.

Another example of real-life HTH insertion is the National Security Agency (NSA) intercepting shipments of computer network devices before their arrival to the specified destination. NSA inserted "beacon implants" directly into the network devices [12] which allowed NSA to exploit the devices and survey the network. One of the infected network devices was used in the Syrian Telecommunication Establishment (STE) cellular network. As a result, NSA was able to access the call detail records containing the billing information from STE which revealed the identity of the callers along with their geographic locations.

The structure of an HTH consists of two main elements: a trigger and a payload. The trigger represents the activation mechanism which wakes the HTH, while the payload indicates the effect of the HTH on the IC. HTHs can be defined using one or multiple of the following six categories [13]:

1- Insertion phase: The manufacturing flow is composed of different phases, mainly specification stage, design stage, fabrication stage, testing stage, and assembly and packaging stage. One characteristic defining an HTH is the manufacturing phase at which it was inserted.

2- Abstraction Level: Different HTHs can be characterized by the abstraction level at which they were inserted. The different abstraction levels are system level, development environment level, register transfer level, gate level, layout level, and physical level. The level of abstraction at which an HTH was inserted indicate the amount of control and flexibility the attacker had while performing the insertion.

3- Trigger: HTHs can be differentiated based on the type of trigger which activates the HTH. There are three different types of triggers. The first type is "always on", in this case the HTH is not waiting for a certain condition to occur; the HTH is awake from

the very beginning of the field operation. The second type is “internal”, where the HTH is awakened by an internal physical condition or signal of the design. The third type is “external”, the HTH is activated based on an external user input or component output.

4- Payload: HTHs differ in the type of alteration performed on the circuit because of the HTH’s implementation. The different effects an HTH can have on a design are change of the circuit’s functionality, degradation of the design’s performance, leak of sensitive information handled in the circuit, or denial-of-service by halting the original functionality of the system.

5- Insertion Location: Different HTHs can be characterized by the location at which they were inserted. The possible locations are the processor, the memory, the inputs or outputs, the power supply, or the clock grid. Moreover, an HTH can be distributed over several locations mentioned above or focused in only one location.

6- Physical Characteristic: HTHs can be differentiated based on their physical features such as their distribution (focused or dispersed) or HTH size (measured in number of gates or percentage of area occupied by HTH compared to the entire circuit size).

Hardware Trojan Detection

HTH detection can be very challenging because of the various options of types, sizes, and location of insertion an adversary can choose from when injecting an HTH [14]. The countermeasures against HTH can be classified into three categories [3]:

1- Run-time Monitoring: This area focuses on detecting HTH after the deployment of IC and hence can be used for the entire lifetime of the IC. HTH which were not detected using the other detection methods can be noticed using run-time monitoring when the HTH is activated [14].

2- Design for Security: There are two pre-design approaches that constitute this area; both are usually implemented throughout the manufacturing flow before the design process starts. Designers have two choices; the first choice is to prevent HTH insertion from happening by utilizing a specific design process for security. The second choice is to somehow facilitate detection of any HTH which may have been inserted while manufacturing the ICs.

3- HTH Detection approaches: The techniques in this field happen at test-time, after the fabrication stage, and can follow the destructive or the non-destructive approaches. Destructive approaches use reverse-engineering to examine each layer of the IC and validate it. Non-destructive approaches can be in one of three categories. The first category detects an HTH at the IP level pre-silicon (IP trust verification). The second category relies on directed test patterns which are applied to the IC to activate the hidden HTH and propagate its effect to be detected (Logic testing). The third category measures a side-channel parameter such as path delay or supply current to expose an inserted HTH (Side-channel Analysis) [3].

This work focuses on the side-channel analysis technique to detect an HTH. Side-channel analysis happens during the post-silicon testing stage. Side-channel parameters are physical parameters of the design which are considered a signature to the design from which they are generated. It was considered that an HTH-free design, called a golden circuit, was available and therefore golden side-channel parameters can be measured. New side-channel parameters coming from a design with unknown status (HTH-infected or free) are then compared to the parameters of the golden implementation. Using this technique allows the identification of whether the circuit at hand is secure or compromised by an HTH. Side-channel parameters characterizing the system can be the power consumption or the delay of certain paths of the circuit [3]. In fact, a modification to the original circuit in the design or manufacturing stages is likely to present itself in the power consumed by the system since the HTH, when active, will consume more power than if it does not exist. Another indication of an HTH is the change in certain path delays of the circuit. This is true since the HTH is added hardware and therefore some of the signals will pass through this added hardware and thus will get delayed more than if the hardware did not exist.

One major parameter affecting the performance of an HTH detection technique using side-channel analysis is the signal-to-noise ratio (SNR) [3]. SNR compares the levels of the signal being examined to the surrounding noise. The signal being examined in this case is the HTH effect on a side-channel parameter, while the surrounding noise is the combination of chip-to-chip, within-die process variations and measurement noise. The combination of the noise components affects the signal being examined and masks any HTH indication. As a result, detecting an HTH needs more than just a simple comparison between the side-

channel parameters of a golden circuit and an unknown circuit. In fact, one of the solutions to this issue is to use techniques which statistically isolate the HTH influence from the different types of noise.

Literature Survey

In this section, different HTH detection techniques will be reviewed and classified based on the type of data used to identify the presence or absence of an HTH.

In [15] a detection technique was used based on packets transfer from one core to another in a many-core system. An HTH was assumed to be inserted at the design-time only within the communication transfers. The implemented HTH was internally activated based on the number of core-to-core transfers and carried out a denial-of-service attack when active. There was no mention of the size of the HTH implemented. Machine learning was used with the following features: source core, destination core, packet transfer path, and distance traversed. More precisely, real-time online learning was used. The run-time anomaly detection algorithm is called Modified Balanced Winnow (MBW). The machine learning model was trained using data from an HTH-free many-core system and then passed the new unknown data to be classified as HTH-infected or not. The model got updated at each data transfer based on feedback from the many-core system. The feedback is composed of core information and core behavior to incoming data packets. The model was able to get up to 94% detection accuracy. Also, two supervised machine learning algorithms were tested with the data: Support Vector Machine and K-Nearest Neighbor. Next, the results of the 3 machine learning algorithms were compared. The conclusion was that the online learning algorithm performs better with an 8% increase in detection accuracy.

In [16] an experimental analysis of power and delay SNR requirements for HTH detection was conducted. The goal was to carry out several statistical outlier techniques to determine the sensitivity of current and delay analysis for detecting HTH. Ring oscillators were used to represent the core logic of a chip under test, which as mentioned does not emulate actual

HTH scenario. The ring oscillators provided a high degree of control over the switching activity in the FPGAs used. Moreover, subtle delay and transient power supply anomalies were introduced through simple modifications to the ring oscillator logic structure. Two HTH scenarios were investigated. The first HTH increased the delay and power consumption by adding capacitive load. The second HTH increased delay but had a small impact on power by adding series inserted gates. There was no mention of the size of the different HTH implemented. A 1-D statistical analysis using the currents from the ring oscillators was implemented first and achieved an HTH detection sensitivity of 83%. Second, a similar 1-D statistical analysis using the frequencies from the ring oscillators was performed and achieved an HTH detection sensitivity of 78%. Next, regression analysis was conducted where the frequency was plotted against the current. The regression analysis reached an HTH detection sensitivity of 100%. A calibration technique was also proposed and implemented hence improving the HTH detection sensitivity by 20%. The calibration technique used current and frequency measurements from a calibration ring oscillator to cancel the process variation effects from the data used in the 1-D and regression analysis. The calibration ring oscillator was chosen randomly from the multiple ring oscillators. It was concluded that by using calibration and regression analysis of the measured current and delay parameters, HTH detection sensitivity can be increased by an order of magnitude.

In [17] a side-channel technique was applied using segmentation and gate level characterization. Gate level characterization resulted in defining each gate of the IC in terms of its physical properties such as the leakage and switching power of the gate. A set of input vectors was applied to the system and a system of linear equations was formulated

by summing up the leakage power of each gate and considering the measurement errors. The linear equations had a single HTH variable which would indicate the presence of an HTH. Solving the system of linear equations for the HTH variable would indicate the presence or absence of an HTH. The presence of an HTH, whether activated or not, caused a systematic bias in leakage Power and hence detection of HTHs was possible by tracing the gate level leakage power. The technique was characterized as scalable since it required large circuits to be divided into small sub-circuits, hence following the divide and conquer method. HTH's with size as small as 6 gates were detected.

In [18] and [19] an HTH detection technique was implemented using multiple side-channel parameters simultaneously. The relation of the dynamic current of the circuit along with the maximum operating frequency was used to isolate HTH effect from process noise. More precisely, the presence of an HTH caused a variation in the dynamic current when compared to a golden chip, while it did not have a similar effect on the maximum frequency which violated the correlation between the current and the frequency. Both side-channel analysis and logic testing methods were integrated to detect different types and sizes of HTH. The work was based on the fact that the switching of an HTH contributed to the dynamic power of the overall system. Also, multi-power port measurements were used instead of limiting the power collection to only one port. A golden design was assumed to exist in order to extract golden data and use it as a reference. The first advantage of the method was that there were no modifications needed to the design being tested. The second advantage was that the HTH did not have to be activated in order for it to be detected. The method was able to detect, with no error, a 4-bit sequential HTH which constituted 0.03% of the circuit under test. The number of gates used in the HTH was not

specified. It was mentioned that the existing side-channel approaches suffered from three shortcomings. The first shortcoming was that existing side-channel approaches were not able to scale well with increasing process variations. The second shortcoming was that existing side-channel approaches considered only die-to-die process variations while not supporting local within-die variations. The third shortcoming was that existing side-channel approaches required design modifications which incurred considerable design overhead and could be compromised by an adversary.

In [20], a security-aware design scheme was implemented for better HTH detection sensitivity. The proposed method used reverse engineering to obtain the design's layout where each standard cell was then characterized based on its sensitivity to malicious modifications. Next, a subset of highly sensitive standard cells on which to synthesize the design was selected such that the HTHs are more easily detected. Finally, the design was re-synthesized by using the highly sensitive standard cells as replacement cells for the original cells. The re-synthesized design was called the security-aware design while the original design was called the baseline design. The one class Support Vector Machine (SVM) algorithm was trained to differentiate between noise and HTH effect for both security-aware and baseline designs. It was assumed that the designer had a golden layout from which to extract the features for the machine learning model. The features used were the difference between the golden and real layout areas as well as the difference between the two layouts' centroids. The HTH, composed of four malicious modifications (two additions and two removals of structures), was inserted in 60 security-aware chips and 60 baseline chips. The SVM algorithm was able to detect 16.87% more HTHs in the security-aware designs with very small power and area overhead compared to the detection in the

baseline designs. The main challenge faced was the realization that to create the security-aware design, some cells had very few replacement alternatives since replacement could cause timing violations.

In [21], stealthy malicious logic is identified using Boolean functional analysis. The proposed tool, Functional Analysis for Nearly-unused Circuit Identification (FANCI), flagged sets of wires that appeared suspicious to be code reviewed. Suspicious wires were rarely used and hence had a high chance of being a backdoor incorporation. More precisely, the degree of control which is the influence that an input had on the operation and outputs of a digital circuit was measured. Based on the degree of control measured, heuristics are used to determine whether a wire is suspicious enough to be flagged for inspection. This permits the discovery of backdoors inserted in the designs prior to fabrication hence the design can be fixed or rejected before getting to the market. The method was developed to be used as a first line of defense for enhancing hardware security along with standard test practices and runtime detection techniques. FANCI implements HTH detection at the level of wires and gates (RTL). The results obtained had no false negatives and low false positives rates, more precisely less than 10 wires per design on average were incorrectly flagged for inspection.

In [22] a guided test generation for isolation and detection of HTH was implemented. The proposed method aimed to highlight the differences between side-channel signal waveforms coming from HTH-infected compared to waveforms coming from HTH-free circuits. The method consisted of two phases. First, the design was subdivided into smaller subsets and randomly generated input vectors were ran to minimize the switching in all subsets except for the one subset under consideration. This was done so that if the subset

under consideration contained an HTH, the power consumed by the HTH would then be significantly higher compared to the subset not containing the HTH. Therefore, this stage resulted in a list of subset candidates which may or may not have contained an HTH. In the second phase, the subsets identified in the first phase were considered and targeted with specific vectors to conclude whether an HTH was indeed present or not. The HTH used were less than 1% of the gate count of the original circuit; the exact size of the HTH was not specified. The proposed method was able to provide 4 to 20 times magnification in the circuit activity for the HTH-infected circuit compared to the HTH-free circuit.

In [23], a temporal self-referencing approach for HTH detection was proposed. The side-channel method was used to compare the transient current signature of a chip at multiple time instances. The self-comparison provided this method with the advantage of not needing a golden chip since the chip's transient current is compared to itself, but at different time windows. More precisely, the method made use of the fact that an HTH-free circuit going through the same set of state transitions multiple times should have a constant transient current over different time instances. On the other hand, an HTH-infected circuit would have its current varying over multiple times instances for the same set of state transitions. The current variation in the HTH-infected circuit would be explained by the uncorrelated state transitions in the HTH. The HTH used was an 8-bit binary counter which was triggered when the counter reached the maximum value. The method failed to detect an HTH which had less than two flip-flops. However, it was specified that such a small HTH would activate its malicious payload in 4 cycles and hence can be detected using another HTH detection technique such as logic testing.

In [24], a technique was implemented to detect HTH inserted in the chip's layout. The method used the side-channel analysis technique to measure the steady state leakage current (I_{DDQ}) from multiple distributed power ports. A chip averaging method was proposed to eliminate within-die variations and perform calibration. In fact, averaging the I_{DDQ} data measured across multiple chips eliminated the random within-die variations in leakage current. Moreover, a statistical ellipse-based detection method was utilized to detect I_{DDQ} anomalies caused by an HTH. The detection method was based on principal component analysis and was able to distinguish between random defects and leakage current anomalies by HTH. A golden design was used to collect HTH-free data. The method was able to detect an HTH which caused a minimum of 10 μA current variation in a 2 mm x 2 mm chip. The HTH injected current in the original circuit, however, there was no mention of the HTH size.

In [25], an HTH detection technique was developed based on Path Delay Measurements. The proposed method was resilient to process variation since the golden circuit used to extract HTH-free data was constructed using 3 uninfected dies to properly smooth prospective measurement errors. Most ICs used a common clock signal for synchronization purposes. On the rising edge of the common clock, data left a register bank and moved to the combinational logic. Path delay measurements were obtained by reducing the internal clock period of the circuit which ended up with an early sampling of the signal values exchanged by register banks. The method was tested on an AES encryption circuit using three different FPGAs. The method was able to detect small sized sequential and combinational HTHs. The combinational HTH interrupted normal AES operation and was triggered when internal signals had simultaneous occurrences of the value '1'. The

combinational HTH size was 11 slices, 0.19% of slices used in the FPGA. The sequential HTH performed a denial-of-service attack and was triggered when its internal counter reached a certain value. The sequential HTH size was 21 slices, 0.36% of slices used in the FPGA. Both HTHs used were inserted after the place and route step in the original layout of the circuit to guarantee smallest visibility.

In [26], an analysis method of power signal was implemented for IC HTH detection. The side-channel analysis was used along with power simulation data from a 128-bit AES encryption circuit. The distance technique called Mahalanobis was used to compare test data with a known sample set, obtained from a golden IC, and to take into account the correlation of the data. The use of the Mahalanobis distance method resulted in successful detection although there was noise masking of the HTH in the time domain. The technique was able to detect an HTH which consisted of 2.72% of the total circuit area. The HTH's size was equivalent to 1129 2-input NAND gates and the HTH was activated when the internal clock of the circuit reached a particular count.

In [27], IC fingerprinting was used to detect HTH. Side-channel information and power traces were extracted from circuit simulation to construct a set of fingerprints for an IC family. A new set of ICs were verified using statistical analysis tests against the previously extracted fingerprints. The signal processing technique called Karhunen-Loeve expansion was used to build statistical models for the noise masking of the power data. The statistical models classified individual noisy power signals. The method was able to detect an HTH of 0.01% the size of the main circuit. The HTH used was a 3-bit comparator with an equivalent area of 3 2-input NAND gates. One of the challenges faced is that it was critical that the overall behavior of the IC in both the data and control paths was captured during

tests. It was assumed that the IC fingerprint generation and validation step was carried out by a trust-worthy IC testing facility.

In [28], an at-speed delay characterization technique was proposed utilizing the side-channel method. The side-channel parameter used was the register-to-register path delays. Extracting the delays was done on the functional paths of the core circuit without affecting timing and functionality. Hence it was difficult for an attacker to tamper with the signature extraction unit added to the circuit since it had no apparent effect on the original circuit. The technique created a fingerprint of the circuit based on the delays measured. Normally, a circuit stores data, after some manipulation, in a destination register. In this method, negatively skewed shadow registers were added to the original circuit at the end of multiple combinational paths. The shadow registers also stored the same data as the destination register. The only difference between the shadow register and the destination register was that the destination register was driven by clock 1, while the shadow register was driven by clock 2, a negative phase shifted (negatively skewed) version of clock 1. The negative phase shifting in clock 2 was equivalent to triggering the shadow register before the destination register by a precisely controlled amount of time. The data latched by the shadow register was compared to that by the destination register every clock period. If the data was the same, the technique added a “0” to the fingerprint. If the data was different, a “1” was added to the fingerprint. One of the challenges faced by this method was that if a well-hidden HTH acted fast when triggered at runtime, it would cause a brief alteration of the circuit path delays and hence the effect would not appear and as a result the HTH will go undetected.

In [29], game theory and its mathematical framework were used to determine the effectiveness of HTH detection. The method considered the adversary and the designer as opposing players in a 2-person strategic game. The first player, the adversary, used a certain HTH while the second player, the designer, followed a specific HTH detection method. To play the strategic game, utility functions were derived for both players defining the steps taken by the players in the game. In case of the adversary, the utility function was the initial gain achieved if the HTH inserted was successful in its attack, times the probability that the HTH would both go undetected and work, minus the cost of the insertion based on the labor, hardware, and software used, minus the loss in case the HTH was detected, times the probability of that detection. The initial gain, probabilities, costs, and loss values were assumed to be accurately estimated by a third party and provided to be used in the utility function. On the other hand, the designers' utility function was the initial loss suffered if the HTH was successful, minus the cost of implementing the specific HTH detection technique based on labor, software, and hardware used, minus the cost of a false labeling as HTH-infected. Also, costs and initial loss values were assumed to be accurately estimated by a third party and provided to be used in the utility function. The game was then played where the designer was trying to minimize its utility function (loss) while the adversary maximized its utility function (gain). The iterated elimination of dominated strategies (IEDS) solution concept was used to solve the game and determine the winning player. The method hence determined the effectiveness of a specific HTH detection technique against a certain HTH implementation. In fact, if the adversary won the game, the HTH detection used by the designer in the game would be considered not effective against the type of HTH used by the adversary in the game.

In [30], the vulnerability of circuits to HTH insertion at the behavioral level was analyzed. HTH insertion during the development stage was only considered. Moreover, the analysis focused only on insertions which either changed a statement rarely executed or carried out an attack through a signal which was rarely observed. The method measured a circuit's susceptibility to HTH insertion based on two criteria. First, the statement's hardness which provided a quantitative measure of the difficulty to execute a statement. More precisely, a statement with high level of hardness is rarely executed and hence its correctness cannot be fully proven using a limited number of tests. Second, the observability of circuit signals which evaluated the degree of contribution input signals had on the final value of the output signals. A new metric for HTH detectability was proposed based on the two criteria previously mentioned. In fact, to detect an HTH, statements should be frequently executed, and inputs should have a noticeable effect on the outputs of a circuit. As a result, any malicious modification to the circuit would be detected. After determining the vulnerability of a circuit, a few solutions were described to increase detectability using the proposed method. For instance, designers can pass low observable signals to primary outputs which facilitates the monitoring of previously low observable signals. Moreover, extra control statements accessible through primary inputs could be added which would increase the frequency of statement execution of once rarely executed statements.

The previously mentioned publications were classified based on the technique and the information used in the detection of HTH. Some of the publications fit under two of the categories and hence have dotted line coming from both categories. Figure 1.2 shows the classification tree. Also, [4] and [17] have the same authors, and [18], [19], and [23] have the same authors, which is indicated by the color of the boxes the publications are in.

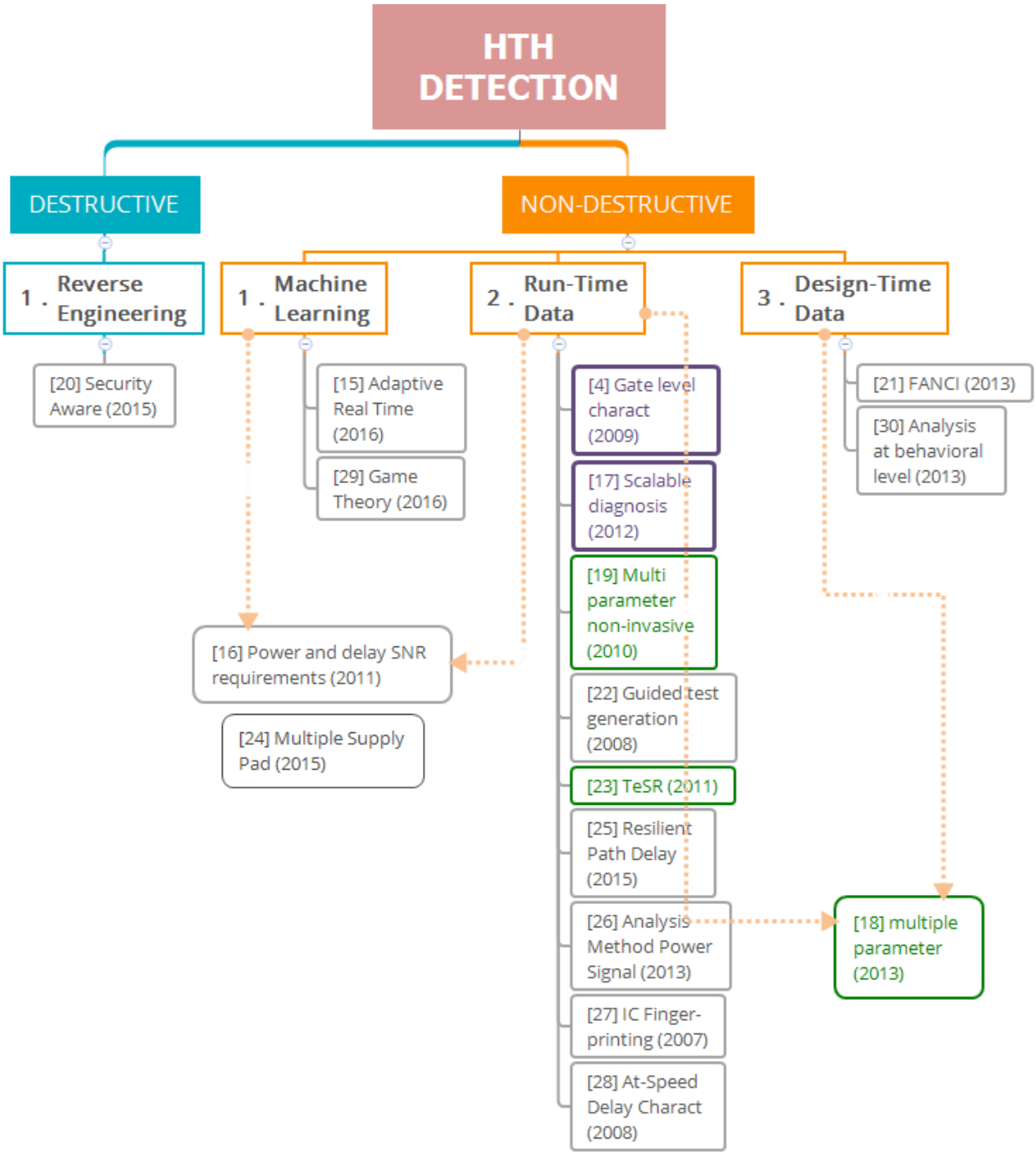

Figure 1.2: HTH Detection Methods Classifications

Chapter 2: Experimental Work

The aim of this work was to develop a method to detect an HTH inserted on an IC using machine learning, as presented in Figure 2.1. A machine learning algorithm was used to facilitate the prediction of whether a circuit was HTH-infected or HTH-free. More precisely, based on the power data collected from an HTH-free circuit (golden implementation), the machine learning algorithm was trained. The trained model was then fed new power data from a circuit under test and the predictions indicated whether the circuit at hand was HTH-infected or not.

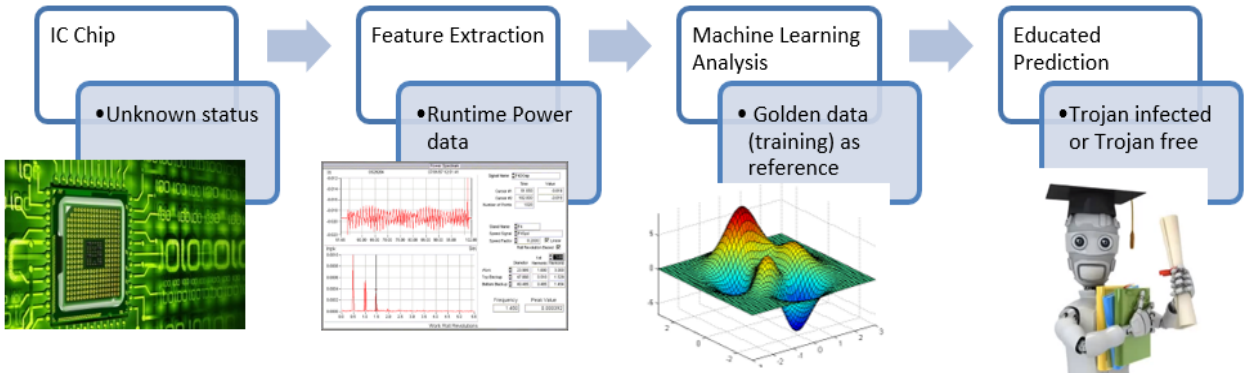

Figure 2.1: Project Overview

After literature review examination, it was concluded that power was one of the important modalities to be used in Hardware Trojan Detection. In fact, as [3] highlighted, a malicious modification or addition to a circuit during design or fabrication would cause an effect on the circuit's power consumption. An FPGA was used to implement the HTH-infected and HTH-free circuits and collect the circuits' power traces. Indeed, the authors in [31] specified that designers were more frequently choosing FPGAs as the IC of choice for commercial and military systems. The golden circuit was implemented on the FPGA and the power traces were collected. Next, features were extracted from the power traces and then labeled. The labeled features were then fed to a logistic regression machine learning

algorithm. In fact, based on the literature review done, Logistic Regression (LR) is the machine learning algorithm used mostly as a starting point when dealing with a Hardware Trojan Detection problem. The details of the proposed method are described in the following sections.

Apparatus and Setup

In this work, the SAKURA-G FPGA board was used to implement HTH-infected and HTH-free circuits and collect power traces. This board was chosen because it is a cryptographic FPGA which was developed for evaluating the security of cryptographic circuits against physical attacks [32]. The board's specifications are specified in [33]. As shown in Figure 2.2, there are two Spartan-6 FPGAs on the board, the LX9 serves as the controller while the LX75 is the main security circuit. The SAKURA-G FPGA board is an ultra-low-noise board and is equipped with an on-board amplifier making power analysis easier. The amplifier has a bandwidth of 360MHz and a gain of +20dB. Two measurement points, SMA connectors J1 and J2 are available to monitor power waveforms on the core voltage V_{CCINT} of the main FPGA. J3, the SMA connector next to the amplifier providing the amplified signal of J2, was used to collect power data. J3 was chosen instead of J2 because the on-board amplifier removed the need for an external amplification process. Usually, an amplification process is needed because otherwise the measured data would be relatively small and would not be very useful. The SAKURA-G can be configured and operated using Verilog-HDL code and the Xilinx ISE WebPACK design software.

The Xilinx ISE Design Suite [34] enabled simulating multiple designs for validating circuit functionality. Also, the tool provided the estimated simulation power of the design along with information on the number and type of gates used in the circuits. Finally, the SAKURA-G FPGA board was configured using the built-in tool from Xilinx.

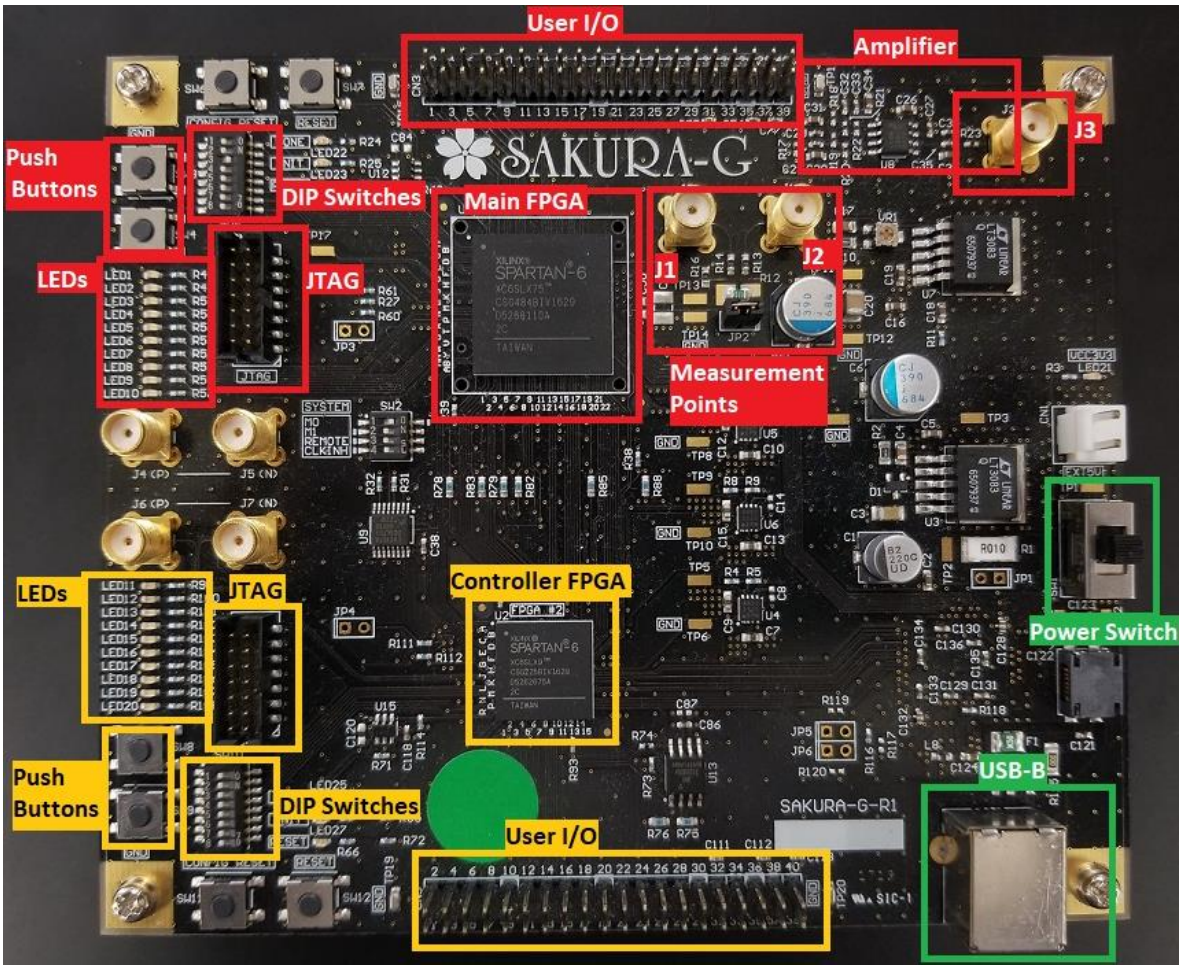

Figure 2.2: SAKURA-G Board Functions, Top View

The design flow adapted by the Xilinx tool is specified in [35] and shown in Figure 2.3. First, a design is created by building a new ISE Design Suite project and then specifying the source files. The source files can be HDL, netlist, schematic, intellectual property (IP), embedded processor, or Digital Signal Processing (DSP) modules. For this project, the source files will be hardware description language (HDL) files, VHDL and Verilog. Second, the design is synthesized by transforming the HDL source files into an architecture-specific design netlist. The built-in synthesis tool in Xilinx ISE Design Suite is the Xilinx Synthesis Technology. Third, the design constraints are specified. Design constraints include I/O pin and layout design requirements. Fourth, the design is implemented by transforming the

logical design to a physical file which is downloaded onto the target device. Fifth, the design implementation is analyzed using any of the various built-in tools such as the XPower Analyzer tool. The XPower Analyzer provides detailed power information namely dynamic and quiescent power values. Sixth, a programming file is generated to configure the target device. From a host computer, the configuration file is uploaded to the Xilinx device using the iMPACT tool. Finally, the designer may choose to simulate the implemented design at various points of the flow for testing and validation purposes. The ISim simulation tool is delivered with the ISE Design Suite and provides power information and signal variations which help verify the functionality of the design by simulation.

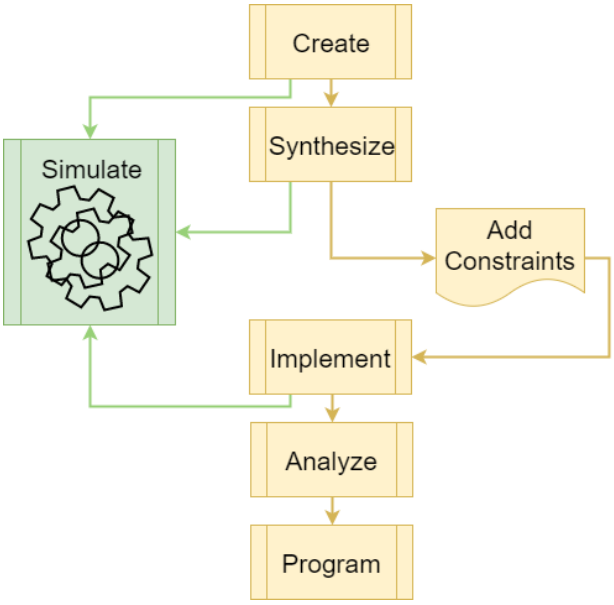

Figure 2.3: Design Flow Steps

To view and save the power traces of the SAKURA-G, a Tektronix TDS2022C oscilloscope [36] was used. More information on the power collection process using the Tektronix TDS2022C oscilloscope can be found in Appendix C.

First, the "BasicRSA-T200" benchmark found on "Trust-Hub" [13] [37] [6] was used. The benchmark, which implemented the RSA encryption algorithm [38] with an HTH inserted, was simulated. RSA is a public-key cryptosystem where the public encryption key is different than the private/secret decryption key [39]. The HTH infecting the "BasicRSA-T200" circuit is externally triggered by the plain text input to the RSA encryption algorithm. The HTH's payload is to deny service. In fact, when the input plain text to the RSA encryption algorithm is equal to the hexadecimal value of "01fa0301", the HTH disabled encoding by setting the encryption key to the value of "1". The VHDL code of the HTH is presented in Figure 2.4. Based on the documentation in [13], the HTH was inserted at the design stage of the circuit, in the processor.

```
-- Trojan disables cypher as inExp (private key exponent (e)) is set to 1,  at the transmitter end
Trojan: process (indata) is
begin
      if indata = x"01fa0301" then  -- if input is specific value, disable encoding
         inputExponent <= x"00000001";
      else
         inputExponent <= inExp;
      end if;
end process Trojan;
```

Figure 2.4: HTH Code-Trigger: External Input, Payload: Disable Encoding

The signal values obtained by simulating the infected "BasicRSA-T200" circuit (left) are compared to the HTH-free circuit (right) in Figure 2.5. The plain text input to the RSA encryption algorithm is the signal "indata", and the cypher text output of the RSA algorithm is the "cypher" signal. The value of the signals can be seen in the column "Value" in Figure 2.5. Indeed, in the HTH-infected column, when "indata" was equal to the hexadecimal value of "01fa0301", the output "cypher" was equal to that exact value indicating no encryption taking place. Compared to the normal operation of the HTH-free circuit, the value of

"cypher" is the hexadecimal value of "031f3988" which is the encryption using RSA when "indata" is equal to "01fa0301".

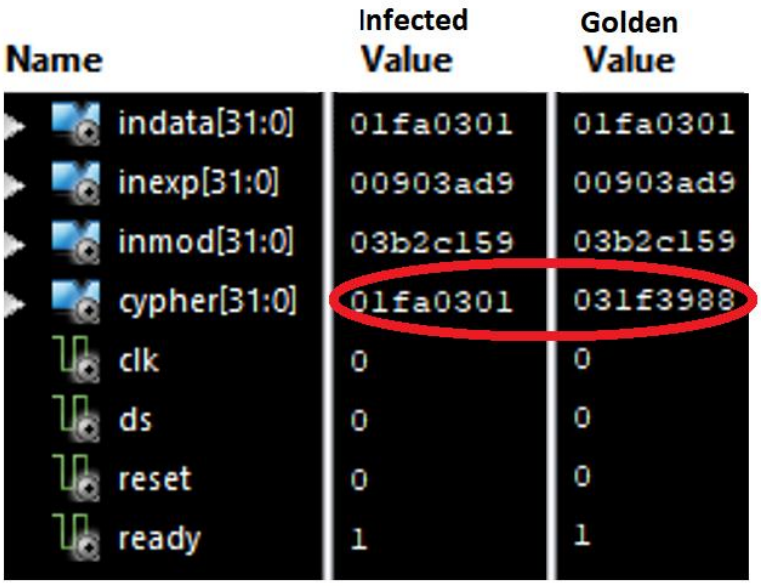

Figure 2.5: Comparison of HTH-infected and HTH-free Simulation Signal Values

In fact, the detailed report provided by the XPower Analyzer tool showed some differences in power values when the circuit under test was HTH-free compared to HTH-infected. Table 2.1 shows those differences. The on-chip power of the clocks was 0.52 mW in the HTH-free circuit, and then decreased to 0.51 mW in the HTH-infected circuit. This decrease caused the total on-chip power to behave accordingly by going from 64.51 mW in the HTH-free circuit to 64.50 mW in the HTH-infected circuit. Similarly, the power supply's dynamic current was 0.43 mA in the HTH-free circuit and then decreased to 0.42 mA in the HTH-infected circuit. As a result, the supply source's total current went from 30.58 mA for HTH-free to 30.57 mA for HTH-infected. This behavior in the power values was expected since the HTH caused denial-of-service. When the HTH stopped the circuit from performing normally, less power was consumed hence explaining the decrease seen after the comparison. Although the simulation power values in Table 2.1 showed 1% or less in

differences, the small differences were enough motivation to investigate the changes that would occur when measuring run-time power data

Table 2.1: Comparing Power Data to Showcase HTH Effect

	HTH-free Circuit	HTH-infected Circuit
On-Chip Clock Power (mW)	0.52	0.51
Total On-Chip Power	64.51	64.50
Power Supply Dynamic Current (mA)	0.43	0.42
Supply Source Total Current (mA)	30.58	30.57

The benchmarks provided on trust-hub were suitable for simulation but required extensive work to be transformed to synthesizable circuits to be implemented on the SAKURA-G. For instance, the various inputs to the benchmark were specified by the test-bench used in simulation. Moving the same benchmark from simulation to actual implementation on the FPGA would necessitate multiple changes in the design to implement some external generation and supply of inputs.

As a result, Verilog source files provided by the makers of the SAKURA-G FPGA board [40] were used instead. These source files implemented the AES encryption algorithm with no HTH, this was the golden circuit in the project. The AES encryption used was a block cipher algorithm with block length of 128 bits [41]. Block cipher is an encryption method which works on the entire block of text instead of handling one bit at a time. AES uses a symmetric key and implements 10 rounds of processing when using 128-bit keys.

The source files, provided by the makers of the SAKURA-G FPGA board and used in this project, are composed of a set of files for the main FPGA, another set of files for the control FPGA, and an AES encryption application to provide the needed inputs and display the

outputs. The first set of files are designed for the main circuit, they are Verilog-HDL source files implementing the AES encryption algorithm along with the interfaces needed for input/output communication with the controller FPGA. Figure 2.6 shows the RTL schematic view Xilinx provides of the circuit implemented on the main FPGA. This big module called "sakura_g_aes128" contains several sub-modules as shown in Figure 2.7. The most important module is the actual block performing the encryption which is called "aes_unit" and is presented in Figure 2.8.

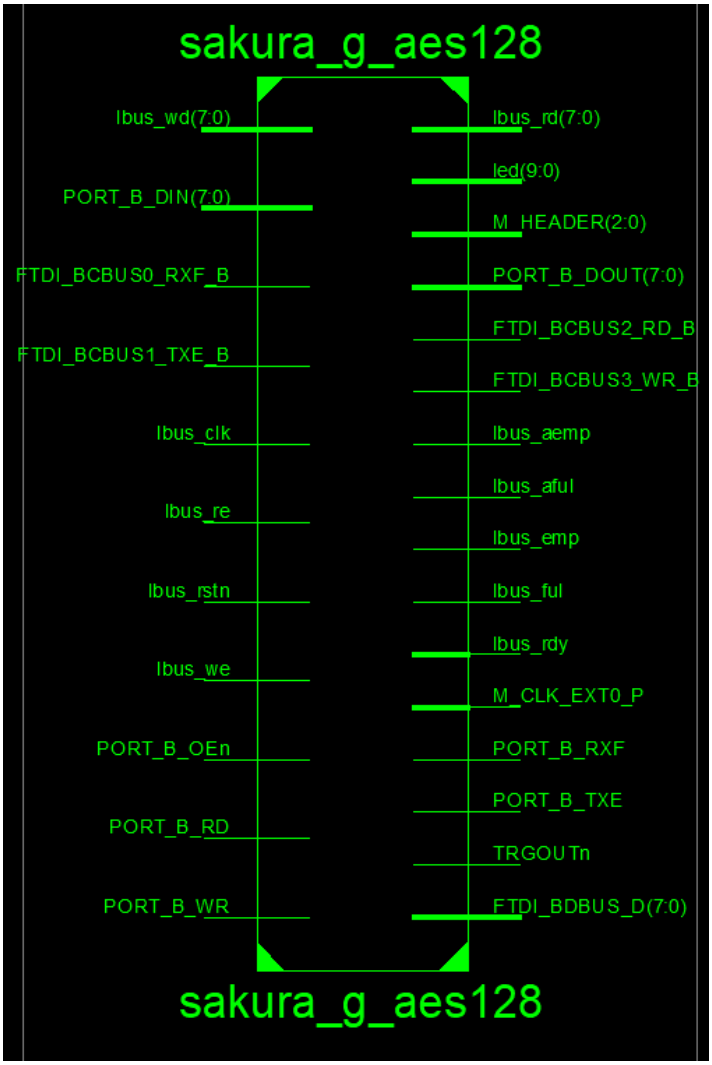

Figure 2.6: Schematic of AES Circuit on Main FPGA

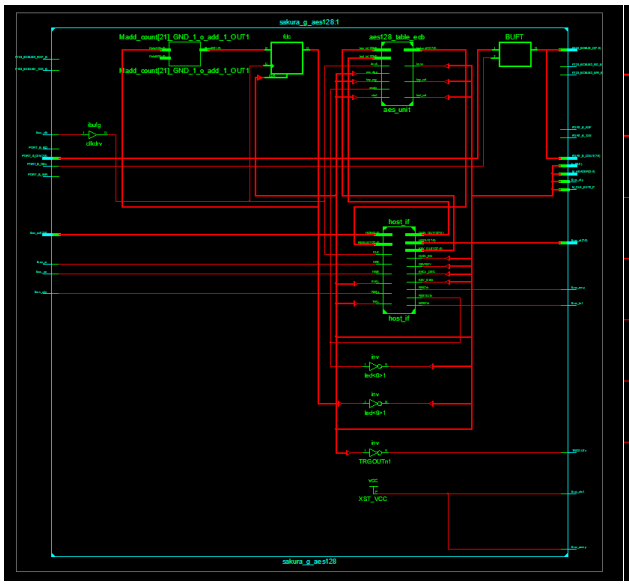

Figure 2.7: Internal View of AES Circuit on Main FPGA

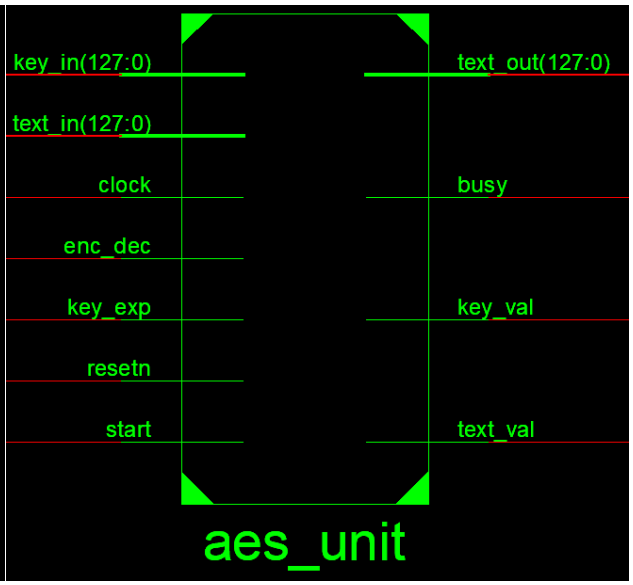

Figure 2.8: View of Module Performing AES Encryption

The second set of files are designed for the control circuit, they are Verilog-HDL source files implementing the interfaces needed for input/output communication with the main FPGA along with the exchange between the control FPGA and the AES encryption application. Finally, the AES encryption application [42] is shown in Figure 2.9. This is the program responsible for providing the inputs to the AES encryption circuit and displaying the outputs from the SAKURA-G. The "Plaintext" field in the application is a 128-bit input to be encrypted by the encryption algorithm of the SAKURA-G. The "Answer" field is a 128-bit encryption of the plaintext by the application itself and not the FPGA. The "Cipher Text" field is a 128-bit encryption of the plaintext by the FPGA. The "Traces" fields indicate how many encryptions the user wants to be performed on the SAKURA-G, along with the progress number of encryption operations done at a certain point in time. The "Key" field is a 128-bit secret key used for the AES encryption. The value can be changed by the user by pressing the "Change Key" button. Finally, there is a "Start" button to launch the encryption operation and a "Stop" button to halt the encryption.

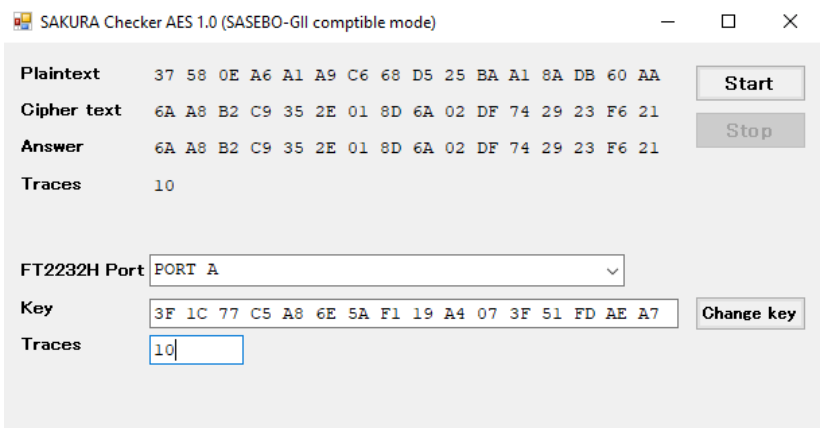

Figure 2.9: SAKURA Checker AES Application

CHAPTER 3: Power Side-Channel Analysis

The experiments started with synthesizing the AES encryption circuit, described in the earlier section, onto the FPGA to finally generate the programming file. More information on the synthesis of circuits using the SAKURA-G can be found in Appendix A. Xilinx provides its own software command language, Tool Command Language (Tcl), which is designed to complement and extend the ISE GUI. Hence, Tcl was the choice of scripting language to automate the synthesis of the design and upload of the bit stream file to the SAKURA-G. More information on the automation using Tcl can be found in Appendix B. Next, the SAKURA-G was connected, as shown in Figure 3.1, to the oscilloscope. A passive probe (1) was attached from channel 1 (2) of the oscilloscope to the trigger signal on pin 1 (3) of the main FPGA's user I/O pins. The ground wire of the probe was connected to the ground pin on the upper left side of the board (4). An SMA Male to BNC Male Coaxial Cable (5) is connected from port J3 (6) on the FPGA to channel 2 (7) on the oscilloscope. Finally, a USB flash drive was inserted into the oscilloscope's USB slot (8). This setup was used to collect the power traces of the AES encryption algorithm implemented on the SAKURA-G. The signal coming from channel 1 of the oscilloscope was the trigger signal generated from pin 1 (3) of the main FPGA's header seen in Figure 3.1. The power collection step was divided into two steps. The first step was performed on the golden implementation, the HTH-free AES circuit. The other step was performed after implementing the HTH to the AES algorithm; the details of HTH insertion will be discussed later.

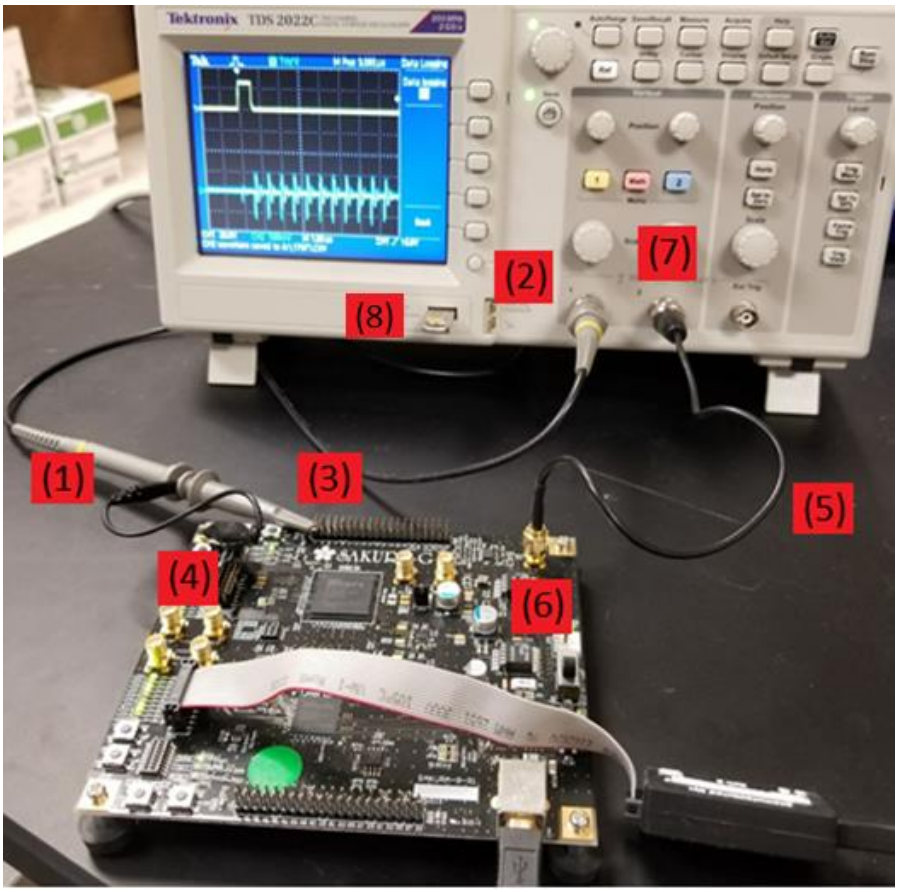

Figure 3.1: Experimental Setup for Power Collection

To collect data from the golden circuit, the AES encryption application was used. Power data corresponding to multiple secret keys and various plaintexts were collected for each secret key. For every secret key, around 1700 csv files were collected and recorded by the oscilloscope to the USB. Every csv file was composed of 2,500 samples representing the power trace from one encryption operation. Figure 3.2 shows the trace appearing on the oscilloscope for an encryption operation. It is important to point out the trigger signal from channel 1 of the oscilloscope, which starts as a baseline, then goes high at the start of the encryption. This observation goes along with the step of setting the oscilloscope's trigger to start data collection as the signal coming from channel 1. The explanation of the waveform in Figure 3.2 is given in the next section titled "Data Interpretation".

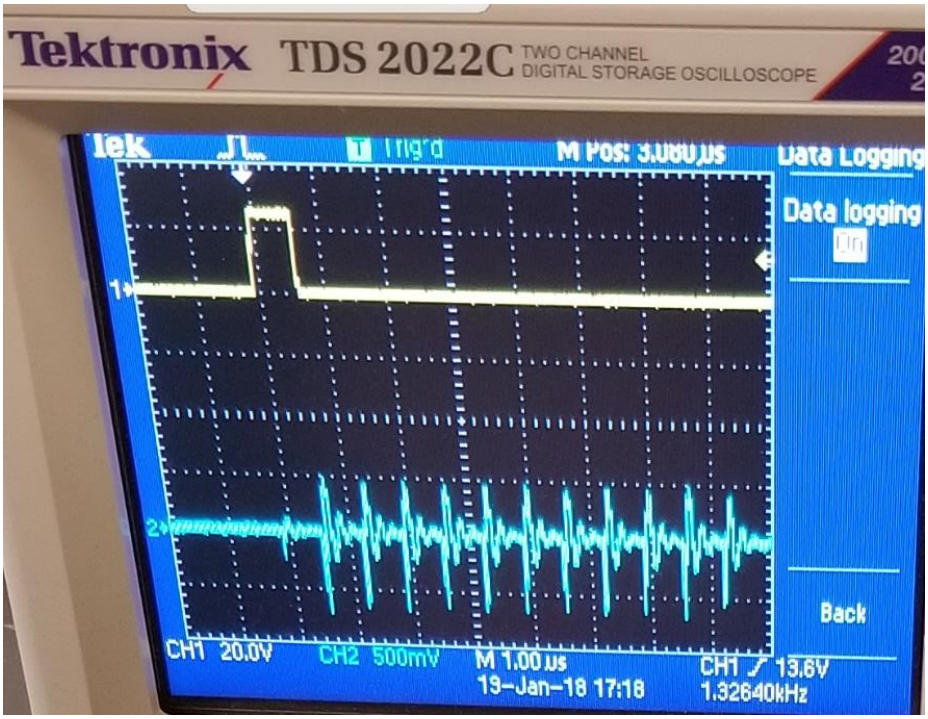

Figure 3.2: Oscilloscope Display of Power from Encryption operation

Next, an HTH was implemented in the AES encryption circuit as shown in Figure 3.3. The HTH was injected in the Verilog source file of the AES unit implemented on the main FPGA, lines 171 to 177. The HTH is triggered when the secret key used in the encryption was equal to the hexadecimal value of "3E 5A 86 2B 58 B1 65 27 F3 15 F9 7C A3 0B 1E 36" (1), also named K4. When the HTH is active, the AES circuit performs encryption with a pre-defined key called "new_key" (2) in the code, different than the key received as input to the AES circuit, called "key_in" (3) in the code. The pre-defined key value is "3F 1C 77 C5 A8 6E 5A F1 19 A4 07 3F 51 FD AE A7", also named K1.

```
159   // Value of key when HTH active
160   reg       [127:0] new_key= 'h3F1C77C5A86E5AF119A4073F51FDAEA7; // K1
161  // ------------------------------------------------------------------
162  // 128-bit Cipher Key Expansion
163  // ------------------------------------------------------------------
164   always @(posedge clock or negedge resetn) begin
165
166     if ( resetn == 1'b0 ) begin
167       { w[0], w[1], w[2], w[3] } <= {128{1'b0}};   // Cipher Round Key reg
168       round10_key <= {128{1'b0}};
169     end
170     else begin
171       // ------------------------------------------------------------
172       // HTH section - Effect: Denial of Service, plaintext compromise
173       // ------------------------------------------------------------
174     (1)if ( key_in == 'h3E5A862B58B16527F315F97CA30B1E36 ) begin // if K4
175           // Cipher Round Key
176           if ( next_state == `IDLE )                    (2)
177              { w[0], w[1], w[2], w[3] } <= new_key; // use K1 instead
178           else if (( next_state == `KEY_EXP ) ||
179                    ( next_state == `ROUND_LOOP ))
180                    { w[0], w[1], w[2], w[3] } <= next_key;
181           else { w[0], w[1], w[2], w[3] } <= { w[0], w[1], w[2], w[3] };
182
183           // Round10 Key Latch
184           if (( now_state == `KEY_EXP ) && ( round_n == 4'd10 ))
185              round10_key <= { w[0], w[1], w[2], w[3] };
186           else round10_key <= round10_key;
187       end
188       // ------------------------------------------------------------
189       //                        End of HTH Section
190       // ------------------------------------------------------------
191       else begin
192          // Cipher Round Key
193          if ( next_state == `IDLE )                 (3)
194             { w[0], w[1], w[2], w[3] } <= key_in;
195          else if (( next_state == `KEY_EXP ) ||
196                   ( next_state == `ROUND_LOOP ))
197                   { w[0], w[1], w[2], w[3] } <= next_key;
198          else { w[0], w[1], w[2], w[3] } <= { w[0], w[1], w[2], w[3] };
199
200          // Round10 Key Latch
201          if (( now_state == `KEY_EXP ) && ( round_n == 4'd10 ))
202             round10_key <= { w[0], w[1], w[2], w[3] };
203          else round10_key <= round10_key;
204       end
205     end
206   end
```

Figure 3.3: HTH Inserted in Verilog of AES Circuit

The effect caused by the HTH can be seen in more details in Figure 3.4. In the AES encryption application, K4 was entered as secret key to be used. As a result, the HTH was triggered and caused the AES encryption to happen using K1 instead of K4. In fact, an encryption of the plaintext "16 5D 8D 39 45 F6 49 D9 5F 13 8B 8C 77 DD C8 76", also called P, with K4 using an online AES tool [43] gave the output of "23 A4 AD 25 E7 75 D2 86 03 A7 A7 C8 A3 B7 BD 17" (PxK4). While the operation PxK1 resulted in the value "89 4B E3 4C C0 76 B9 2B 77 D0 15 3A F0 23 EA AE". Looking at Figure 3.4, the "Cipher text" field, which is the result of the encryption by the infected circuit, is displaying the PxK1 value instead of the PxK4 value. Moreover, the "Answer" field, which is the result of the encryption by the external SAKURA Checker tool, is displaying the PxK4 value. Consequently, the encryption result generated by the circuit as output is not correct. These observations comply with the Verilog code inserted in the AES unit as the HTH.

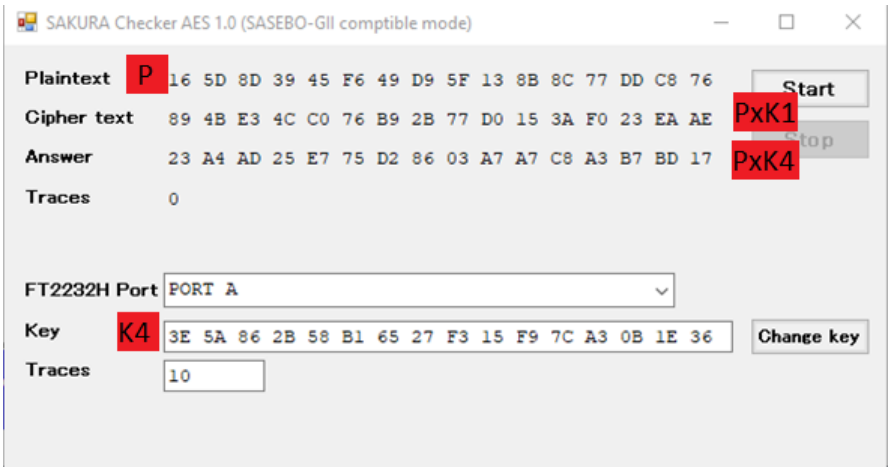

Figure 3.4: HTH Effect

The HTH hence caused denial-of-service, along with compromising the plaintext to be encrypted by using a pre-defined key known by the attacker. Finally, the power traces of the faulty circuit were collected using the same experimental setup described earlier. This

step concluded the power collection stage for the project as seen in Figure 3.5, and provided the golden power data along with infected power data.

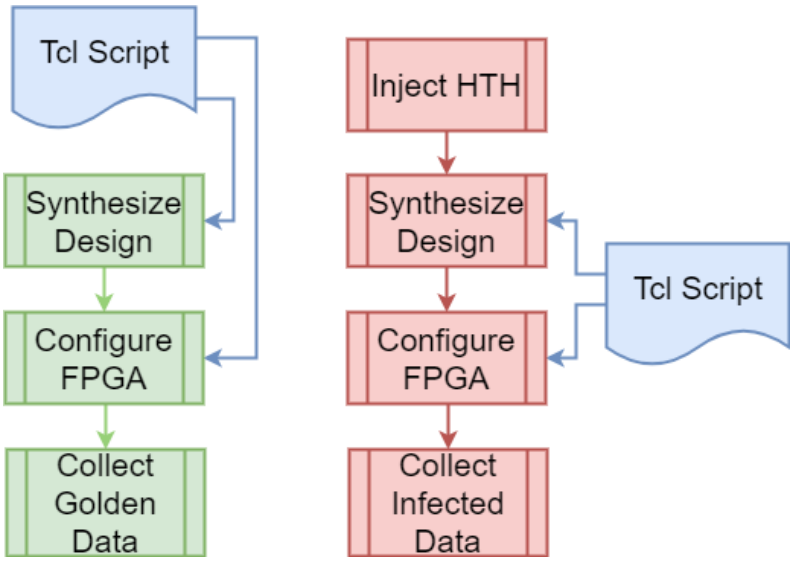

Figure 3.5: Process Flow for Data Collection Stage

Data Interpretation

The data recorded by the oscilloscope was closely observed, then the points in one of the csv files were plotted, to obtain Figure 3.6. The plot was found to have the exact shape when compared to the oscilloscope display in Figure 3.2.

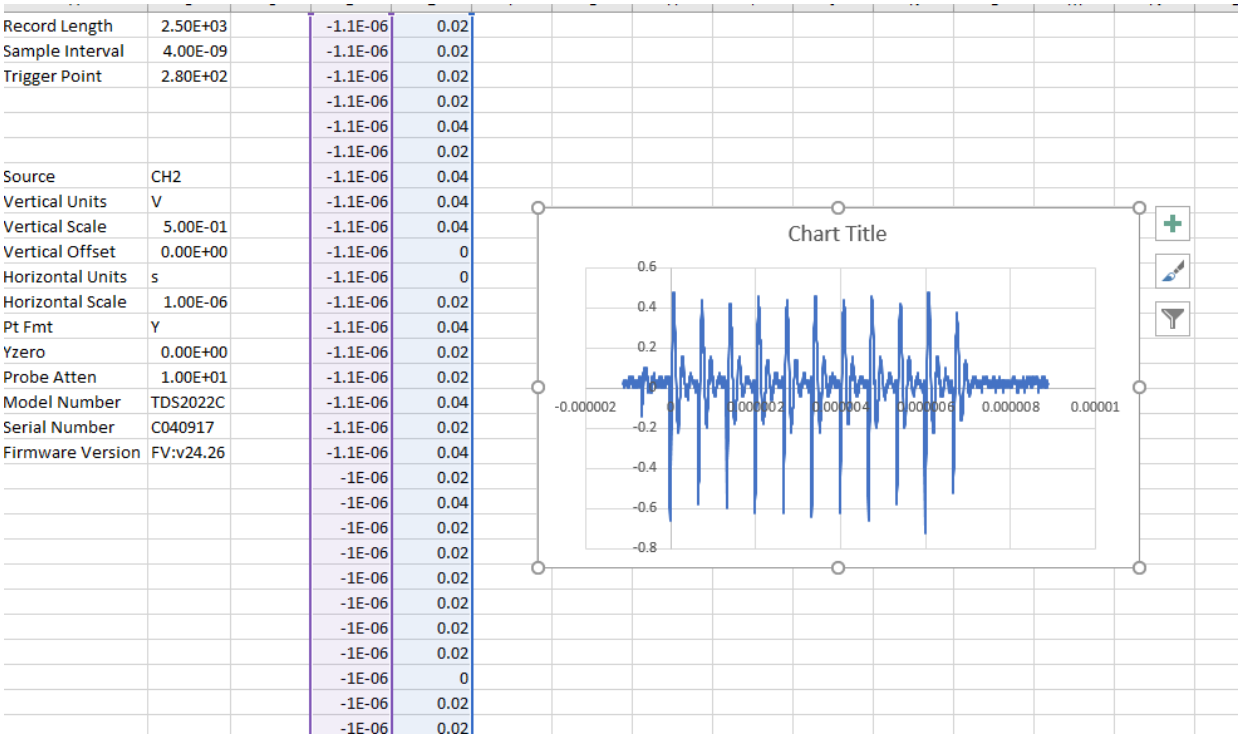

Figure 3.6: CSV file Recorded from Oscilloscope to USB

A repetitive pattern in the encryption power trace can be observed. More precisely, the shape seen in Figure 3.7 is repeated 11 times for every encryption operation.

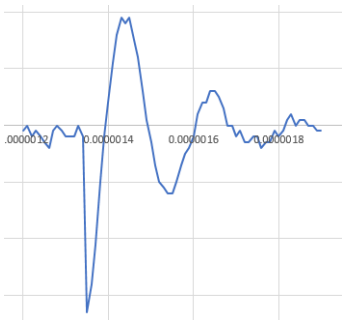

Figure 3.7: Repetitive Pattern in Power Trace

Therefore, some additional research was done on the theory behind power traces of an AES encryption circuit. It was found that the power traces obtained from an AES encryption circuit can identify the different stages in the AES encryption [44]. In fact, the authors provide Figure 3.8, which shows a power trace of a smart card chip performing AES encryption. The AES power trace was then divided into 10 parts corresponding to the 10 rounds performed in the AES algorithm.

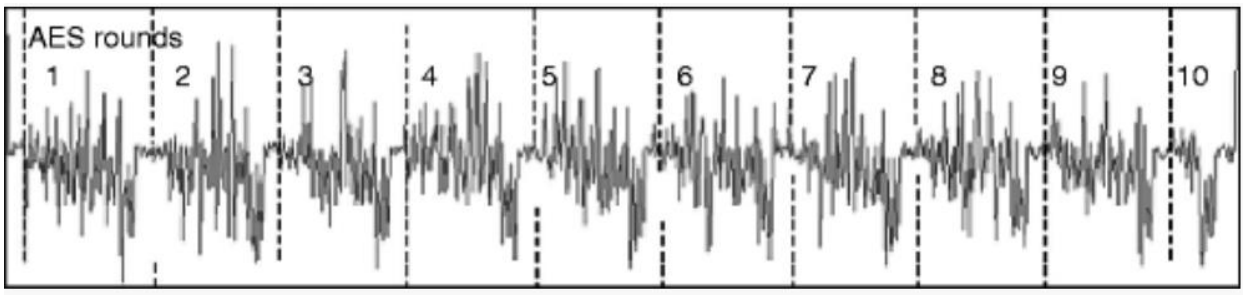

Figure 3.8: AES Power Trace [44]

Similarly, the AES power trace obtained could be divided into 11 parts as seen in Figure 3.9. This division will be helpful when extracting features to build the machine learning model in the next sections.

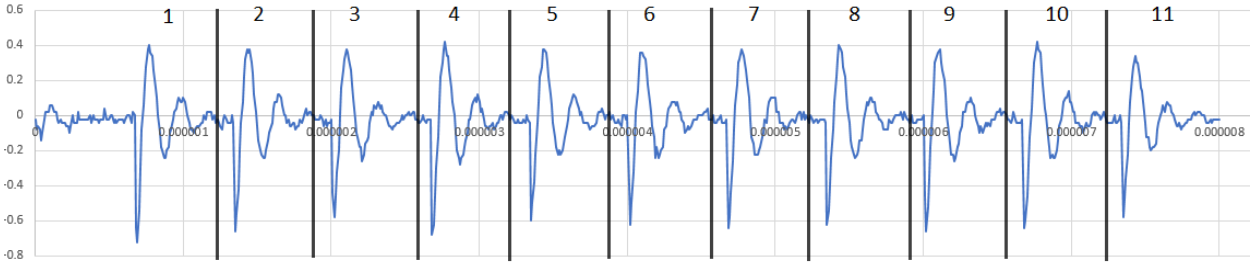

Figure 3.9: Divided AES Power Trace

CHAPTER 4: Machine Learning in HTH Detection

Machine learning can be defined as the field where computers are provided with the needed data to make an educated decision without explicitly being programmed to do so [45]. The task of providing the needed information is also called training. A machine learning algorithm is trained on a set of data called training data, then the trained model is evaluated by testing a new set of data against the model. Machine learning algorithms can be divided into two main categories, supervised learning and unsupervised learning. In supervised learning, the data used to train the algorithm includes the desired output value for the specific input. The desired output value is also known as a label. In other words, training data is labeled with the desired output while testing data is not and the machine learning algorithm is expected to predict the output for that testing data set. A supervised machine learning algorithm can either perform regression by predicting a continuous valued output, or classification by predicting a discrete valued output. Unsupervised learning on the other hand does not use labels but tries to find some unknown structure in a given data set. Figure 4.1 shows the difference between supervised and unsupervised learning where the points plotted are the features of the machine learning algorithm. In supervised learning, the training features are labeled either as blue circles or red crosses and hence a pattern can be extracted. The algorithm's task after training would be to predict if the new testing data belongs to the blue circle or the red cross class. In contrast, unsupervised learning examines unlabeled features as green by the green stars. The machine learning algorithm after training would predict whether the new testing data belonged to the upper or lower class.

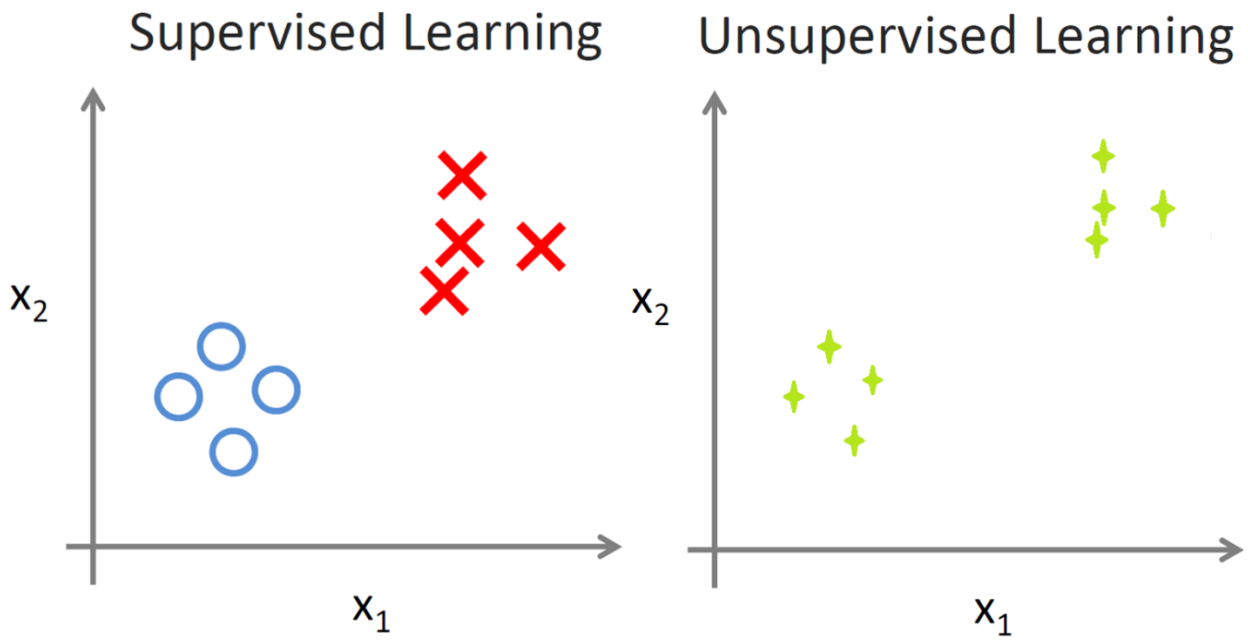

Figure 4.1: Supervised vs. Unsupervised Learning [45]

In this work, the educated decision taken by the machine learning algorithm helped with HTH detection. The input data to the machine learning algorithm was a set of features representing the characteristics of the power traces discussed in earlier sections. Power features in the training data set were labeled based on the key used in the specific AES encryption operation performed when the data was collected. Consequently, the algorithm used supervised learning. The model's output was a prediction of the key used in the encryption operation based on the testing power data. As a result, the machine learning algorithm performed classification by predicting a discrete valued output, the value of the encryption key used. If the model's prediction of the encryption key was in alignment with the actual key, it was concluded that the circuit was HTH-free. On the other hand, if the prediction by the machine learning algorithm was not the key used while encrypting, it was decided that the circuit was HTH-infected.

Model Creation and Evaluation Iterations

The final machine learning model was created by performing multiple optimization iterations. The initial setup utilized the logistic regression algorithm along with 4 different types of features, 2 labels, and 20,000 samples, also called number of examples. As specified in [45], an intermediate number of examples to start with was around 10,000 samples. Consequently, the examples were divided into 50% training data and 50% testing data to have 10,000 samples in each data set. Moreover, it was decided to follow the same approach as in [46] by starting with only 4 features: the minimum, maximum, mean and standard deviation values of the power traces' data points. Since the results from the initial setup were not acceptable, the first step in "debugging a learning algorithm" approach [45] was followed and the training data size was increased to compensate 67% of the total data set. The second step in the approach was to increase the number of features from 4 to 44. After that, the data was tested on different types of classifiers. Finally, more data was collected then a second classifier comparison was performed.

Iteration 1: Start with Preliminary Setup

Logistic Regression was the machine learning algorithm to start with as an initial stage because of its simplicity and ease to use. Logistic regression takes as input a training set composed of m tuples $(x^{(i)}, y^{(i)})$ where m is the number of examples used, $x^{(i)}$ is the i^{th} vector of features, and $y^{(i)}$ is the label corresponding to the i^{th} vector of features. Every vector of features $x^{(i)}$ consists of n components where n is the number of features used. Finally, every label y belongs to a set of discrete classes $\{y_1, y_2, \dots, y_l\}$ where l is the number of labels used. The labels represent the different keys used in the encryption operations.

68 encryption operations were done using Key 1 and an additional 68 encryptions using Key 2. Hence, in this iteration the ML algorithm will be trained using 2 labels ($l = 2$). For every trace obtained from an encryption operation, there were 2,500 power samples. The 4 features were extracted for every 17 samples in every power trace obtained from one encryption operation. The number of samples (17) was chosen randomly. The features obtained were then labelled with the key used in the encryption operation. Equation (1) describes the details mentioned above which leads to 20,000 number of examples ($m = 20{,}000$) used in the first iteration.

$$m = \left(\frac{2{,}500}{17}\right)\frac{examples}{trace} \times 68\frac{traces}{label} \times 2\ labels = 20{,}000\ examples \qquad (1)$$

The 20,000 number of examples were divided arbitrarily into 50% training data and 50% testing data. The machine learning algorithm was trained using 5,000 examples for each label and 4 features. In other words, the input file to train the machine learning algorithm consisted of 10,000 rows by 5 columns of data. Initially, the RapidMiner tool [47] was used to train the algorithm. RapidMiner is useful for visualizing data as well as implementing classification tasks since logistic regression is part of the software package. Building a process in RapidMiner requires selecting and connecting operators. Figure 4.2 shows a 3-D plot of the features in the input file. The x-axis represents the labels, the y-axis represents the "minimum" feature and the z-axis plots the rest of the features which are the "maximum, mean and standard deviation" differentiated by the colors of the points plotted. As seen, the data points for the first label ($y = 0$) are distributed very similarly to the way the points for the second label ($y = 1$) are spread. This observation hinted to the expectation that the data would not be separable, resulting in a low prediction accuracy for the machine learning algorithm.

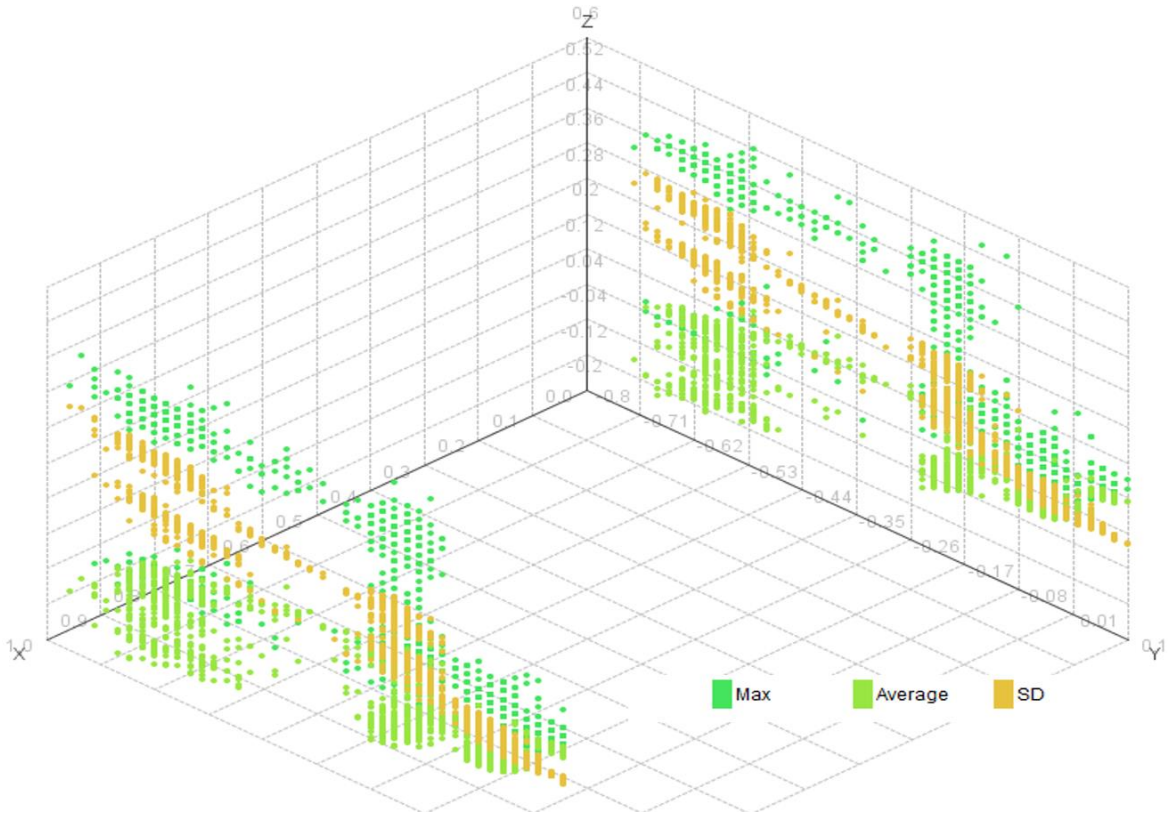

Figure 4.2: Plot of Features Iteration 1

A snapshot from the tool's process can be seen in Figure 4.3. The process started with a training phase and ended with a testing phase. More information on the functionality of the used blocks can be found in Appendix D.

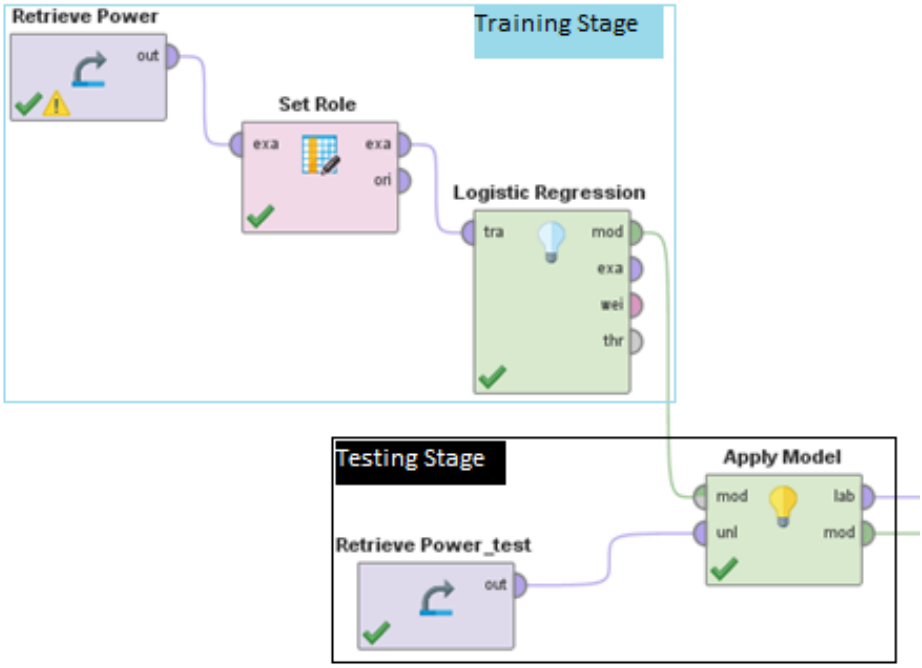

Figure 4.3: Different Stages of Creating Model

The results obtained in the testing stage are summarized in Table 4.1. As expected from the plot in Figure 4.2, the machine learning algorithm was only able to correctly predict 53% of the unseen data.

Table 4.1: Prediction Results Iteration 1

	Number of Predictions	**Percentage (%)**
Correct	5033	53.4
Incorrect	4386	46.6
Total	9419	100

Based on the unsatisfactory results obtained, the decision was made to increase the data set hoping the results would improve. This decision was based on the first step in "debugging a learning algorithm" approach [45].

Iteration 2: Use More Training Data

Since the RapidMiner tool only allowed training data sets of 10,000 training examples maximum, the work was moved to the scikit-learn tool [48] [49]. Scikit-learn was used with 20,000 examples in total, where 67% was used for training and 33% for testing. Principal component analysis (PCA) was used in scikit-learn to perform linear dimensionality reduction to project the data to a lower dimensional space, from 4-dimentionality of 4 features to 3- dimensionality to be plotted. PCA is a statistical procedure that reduces from n–dimension to k-dimension by finding k vectors called "Eigen Vectors" onto which to project the data, while minimizing the projection error [45]. The first three Eigen vectors are plotted in Figure 4.4. The data points for the two labels were intertwined and would not be separated; therefore, the machine learning algorithm would probably not be able to have accurate predictions.

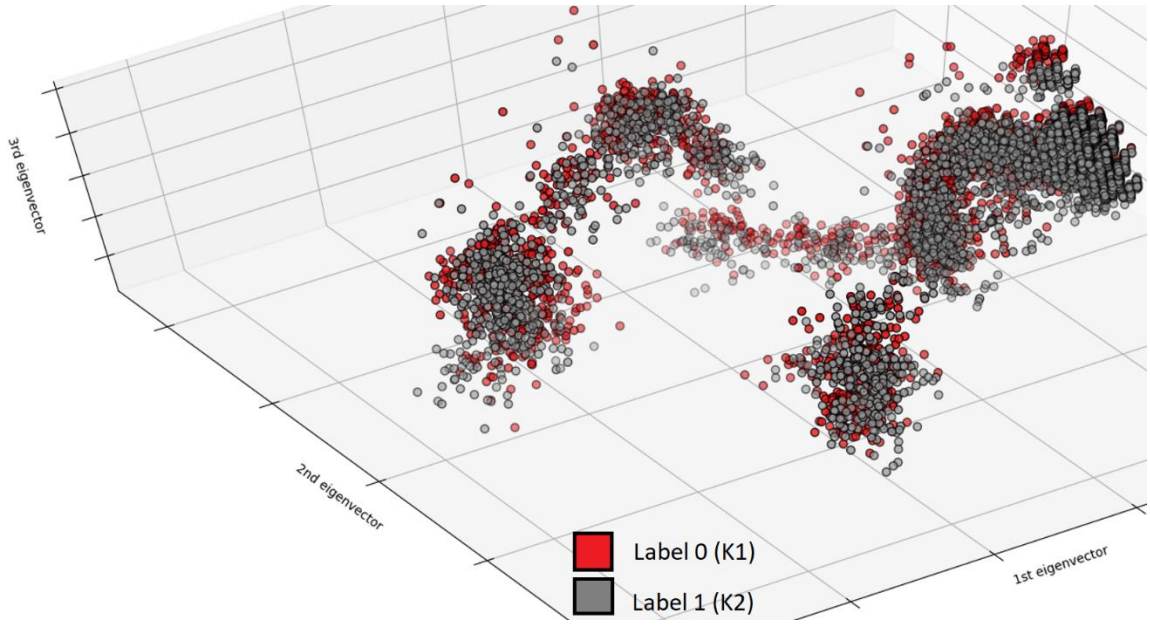

Figure 4.4: Plot of First Three PCA Directions Iteration 2

The logistic regression algorithm was trained and then tested, the results obtained were summarized in Table 4.2 and Table 4.3. More precisely, Table 4.2 presents the confusion

matrix which evaluates the accuracy of the classification. It is important to introduce the notations used in the confusion matrix: true positive(t_p), false positive(f_p), true negative(t_n), false negative(f_n) specific for every label x. Firstly, (t_p) indicates data coming from label x correctly classified as coming from label x. Secondly, (f_p) indicates data coming from label x incorrectly classified as coming from another label. Thirdly, (t_n) indicates data coming from a label different that label x correctly classified as not coming from label x. Fourthly, (f_p) indicates data coming from a label different that label x incorrectly classified as coming from label x.

Table 4.2: Confusion Matrix Iteration 2

Test Points = 5826	**Predicted: Label 0**	**Predicted: Label 1**	**Total Actual**
Actual: Label 0	($\boldsymbol{t_n}$) = 2263	($\boldsymbol{f_p}$) = 662	2925
Actual: Label 1	($\boldsymbol{f_n}$) = 1919	($\boldsymbol{t_p}$) = 982	2901
Total Predicted	4182	1644	5826

To understand the values presented in Table 4.2, the main classification metrics found in Table 4.3, precision, recall, f1-score, and support values, were computed. According to [48], the precision score is the ratio $\frac{t_p}{t_p + f_p}$ representing the ability of the classifier not to label a sample that is negative as positive. Keeping in mind that the maximum value for the precision score is 1, low values of 0.54 for label 0 and 0.6 for label 1 were obtained. Next, the recall is the ratio $\frac{t_p}{t_p + f_n}$ which measures the proportion of positives that are correctly identified as such. Keeping in mind that the maximum value for the recall score is also 1, low values of 0.77 for label 0 and 0.34 for label 1 were obtained. The f1-score is the weighted harmonic mean measuring the classification's overall performance while considering both the precision and the recall, the maximum value is 1. The low score of 0.64 for label 0 and 0.43 for label 1 were obtained.

Table 4.3: Classification Report Iteration 2

Metrics	***Precision***	***Recall***	***F1-score***
Label 0	0.54	0.77	0.64
Label 1	0.6	0.34	0.43
Average	0.57	0.56	0.53

These results indicate that, as expected by the shape of the PCA plot, adding more data to the model did not solve the problem. In fact, it was suspected that the low rates were obtained because of the lack of tailoring to the kind of data handled when extracting the features. As a result, the process of feature extraction used previously was changed.

Iteration 3: Extract More Features

In this iteration, new features were extracted based on the shape of the power trace obtained after an encryption operation. In fact, as previously mentioned, a pattern can be seen repeating 11 times throughout the power trace of one encryption operation. Consequently, every power trace was divided into 11 different parts, then the minimum, maximum, mean and standard deviation values were extracted for every part. This resulted in a number of features $n = 44$ along with using the same number of examples $m = 3{,}597$ with 2 label K1 (label 0) and K2 (label 1). 67% of the number of examples was used for training and 33% was used for testing. To visualize the data, PCA was used; the first three PCA directions or Eigen vectors are plotted in Figure 4.5.

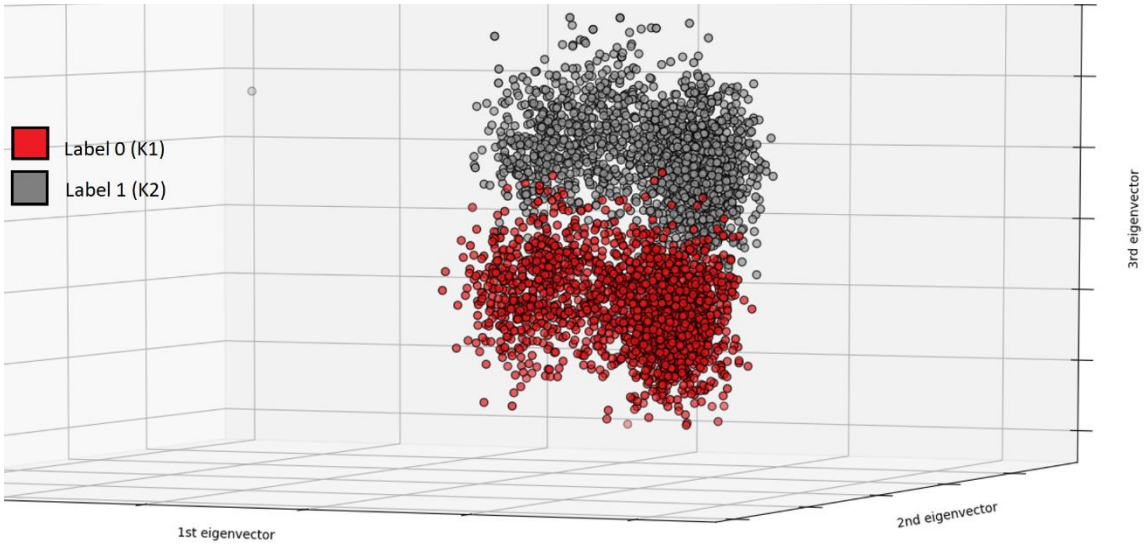

Figure 4.5: First Three PCA Directions Iteration 3

The plot shows a clear separation between the data from label 0 and the data from label 1. The algorithm was then trained, and the model tested on unseen testing data. The results obtained are summarized in Table 4.4 and Table 4.5. The confusion matrix is presented in Table 4.4, 583 out of 588 were correct predictions for label 0 and 594 out of 600 were correct predictions for label 1. These results present a huge improvement compared to the scores of the previous iteration.

Table 4.4: Confusion Matrix Iteration 3

Test Points = 1188	**Predicted: Label 0**	**Predicted: Label 1**	**Total Actual**
Actual: Label 0	(t_n) = 583	(f_p) = 5	588
Actual: Label 1	(f_n) = 6	(t_p) = 594	600
Total Predicted	589	599	1188

The precision, recall, f1-score, and support values were also computed in Table 4.5. For both labels, all the scores are 0.99 out of 1 which is very satisfactory and as expected after looking at the PCA plot.

Table 4.5: Classification Report Iteration 3

Metrics	***Precision***	***Recall***	***F1-score***	***Support***
Label 0	0.99	0.99	0.99	588
Label 1	0.99	0.99	0.99	600
Average	0.99	0.99	0.99	Total = 1188

Finally, a 10-fold cross-validation was performed for further validation of the scores obtained [50]. The entire sample of examples was randomly partitioned into 10 equal size subsamples. Of the 10 subsamples, 9 were used as training data and 1 was used for testing. The cross-validation process was then repeated 10 times, equal to the number of folds specified, with each of the 10 subsamples used exactly once as the validation data. The 10 results from the folds were averaged. The average came out to be equal to 0.99 as shown in Table 4.6. The advantage of this method is that all the data points are used for both training and validation insuring that the scores obtained don't depend on the set used in training and testing but instead, reflect the model's functionality. This score proves that the model's performance is good.

Table 4.6: 10-Fold Cross-Validation Results

Fold	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Average**
Score	0.99	0.99	0.98	0.98	0.98	0.99	0.99	0.99	0.99	0.98	0.99

The next step was to test different ML algorithms other than logistic regression on the same data.

Iteration 4: Test Different Classifiers

Scikit-learn was used to compare the performance of logistic regression against nine different classifiers namely "K-Nearest Neighbors (KNN)", "Linear Support Vector Machine (SVM)", "Radial Basis Function (RBF) SVM", "Decision Tree", "Random Forest", "Neural Network", "AdaBoost", "Gaussian Naive Bayes (GaussianNB)", and "Quadratic Discriminant Analysis (QDA)". The results obtained for every algorithm are summarized in Table 4.7.

1- K-Nearest Neighbors (KNN): This algorithm predicts the label of a new data sample based on a predefined number (k) of training samples closest in distance to that

new sample. A 10-fold cross-validation average score of 0.985 and an f1-score of 0.98 were obtained for the K-Nearest Neighbors algorithm.

2- Support Vector Machine (SVM): This algorithm outputs, after training, an optimal divider which separates the different training data points based on their labels. This divider, also called decision boundary, is used when predicting the label of new examples. The distance between the decision boundary and the closest training example is called the margin within SVM's theory [45]. Based on that, SVM's goal is to produce the optimal separating decision boundary while maximizing the margin.

    a. The linear SVM produces a linear decision boundary. The results achieved were a 10-fold cross-validation average score of 0.995 and an f1-score of 1.00, which is equal to the maximum score using the Linear SVM algorithm.

    b. The RBF SVM produces a non-linear decision boundary. RBF SVM had the lowest scores compared to the other nine classifiers: 0.510 for 10-fold cross-validation average and 0.33 f1-score average.

3- Decision Tree: This algorithm creates a model by learning simple decision rules inferred from the training data features [48]. A 10-fold cross-validation average score of 0.957 and an f1-score average of 0.97 were obtained.

4- Random Forest: This algorithm fits multiple decision tree classifiers on sub-sets of the training data. Then, averaging is used to improve the predictive accuracy of the final model [48]. A cross-validation score of 0.943 and an f1-score of 0.94 were obtained.

5- Neural Network: The multilayer perceptron (MLP) algorithm was used. It is an artificial neural network model that maps sets of input data onto a set of

appropriate outputs [51]. MLP is a feed-forward algorithm meaning the information flows only in the forward direction, from input neurons to output neurons. A cross-validation value of 0.994 and an f1-score of 0.99 were obtained for the MLP Neural Network algorithm.

6- AdaBoost: This classifier has two stages. First the algorithm fits a classifier on the original training dataset. The second phase is to fit additional copies of the classifier on the same training dataset after adjusting the weights of incorrectly classified instances. As a result, subsequent classifiers focus more on difficult instances because of the weight adjustments previously performed [48]. A cross-validation value of 0.993 and an f1-score of 1.00, which is equal to the maximum score, were obtained.

7- Gaussian Naive Bayes (GaussianNB): This algorithm assumes that the likelihood of the features' occurrence follows the Gaussian distribution. GaussionNB then applies the Bayes' theorem with the "naive" assumption that every pair of features are independent [51]. A cross-validation value of 0.987 and an f1-score of 0.99 were obtained for the GaussianNB classifier.

8- Quadratic Discriminant Analysis (QDA): This algorithm is a type of Bayesian classifier which tries to minimize the probability of misclassification [52]. It has a quadratic decision boundary which is generated by fitting a Gaussian density to each class of data [48]. A cross-validation value of 0.996, which is the highest compared to all other nine classifiers, and an f1-score of 0.99 were obtained.

Table 4.7: Classifier Comparison Results Iteration 4

	10-Fold Cross-Validation Average Score	F1-score Average
Logistic Regression	0.993	0.99
KNN	0.985	0.98
Linear SVM	0.995	1.00
RBF SVM	0.510	0.33
Decision Tree	0.957	0.97
Random Forest	0.943	0.94
Neural Network	0.994	0.99
AdaBoost	0.993	1.00
GaussianNB	0.987	0.99
QDA	0.996	0.99

The classifier comparison inferred that, on one hand using "QDA" gives the best 10-fold cross-validation average score of 0.996. On the other hand, the "Linear SVM" or "AdaBoost" classifiers result in the highest average of f1-score equal to the maximum possible score of 1.00. The decision was to continue work with the logistic regression algorithm because the scores obtained (0.993 cross-validation and 0.99 f1-score) are very close to the best values acquired from the other classifiers. Moreover, the logistic regression algorithm had a significantly low computation time compared to most of the other classifiers. The next step was to consider more than two labels, in other words, consider power data from encryption operations using four keys instead of just two keys.

Iteration 5: Consider More Labels

Power data from encryption operations using 4 different keys resulting in 4 different labels were considered for the machine learning algorithm. Consequently, the parameters were number of examples ($m = 7{,}292$), number of features ($n = 44$), and number of different labels ($l = 4$). Figure 4.6 shows the plot of the first PCA directions for the new dataset with four labels. Good prediction results were expected from this iteration since data separation is still observable in Figure 4.6 even after doubling the number of labels.

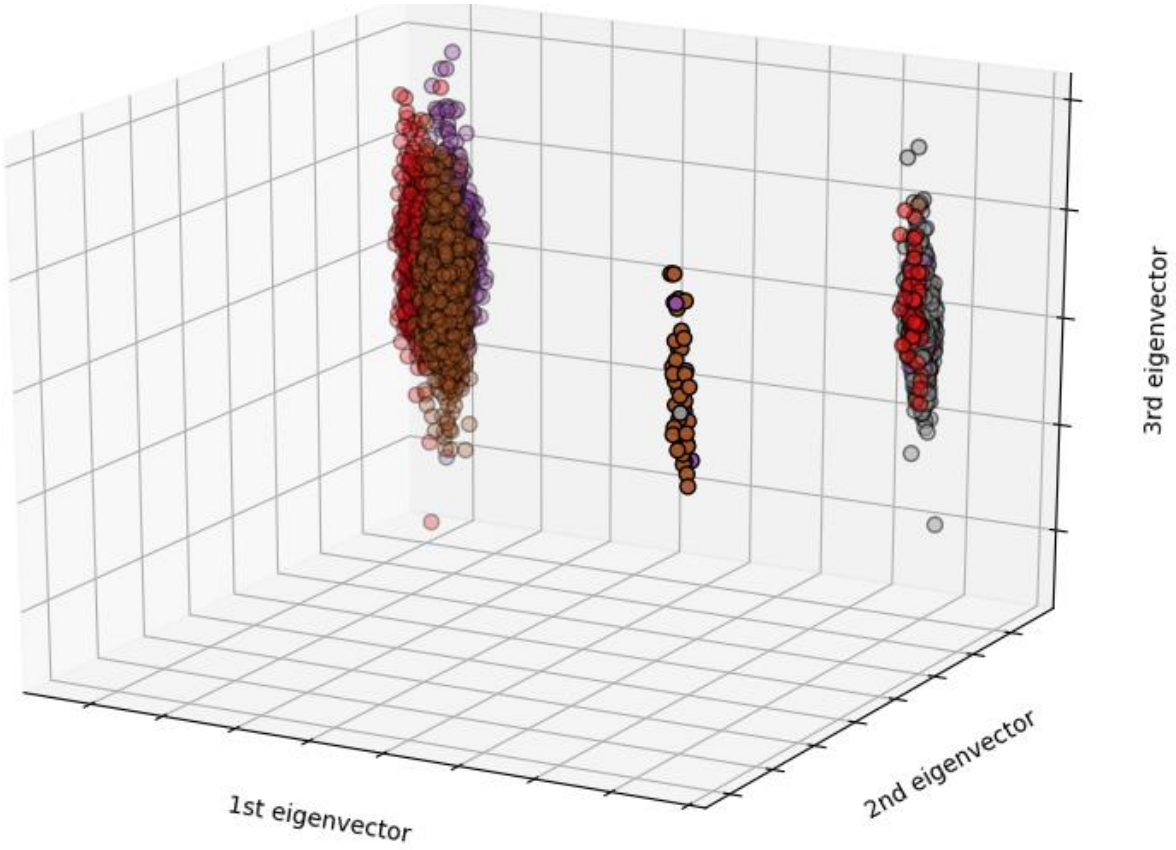

Figure 4.6: First Three PCA Directions Iteration 5

The logistic regression algorithm was trained with the four-label data, the model was tested on new data, and finally cross-validation was performed. The results obtained can be found in Table 4.8. The confusion matrix compares the predictions on the test data to their actual labels. The diagonal of the confusion matrix represents the correct predictions. Next, the f1-score for every label was computed: it was 0.94 for label 0 and label 2, 0.91 for label 1, and 0.97 for label 3. Moreover, the average f1-score was 0.94. Finally, a 10-fold cross-validation was performed and 10 scores were obtained and then averaged to get a value of 0.952. These results are high and approach the maximum possible value of 1.00, hence they are satisfactory for this proof of concept project.

Table 4.8: Classification Results Iteration 5

		Predicted				
		Label 0	Label 1	Label 2	Label 3	
Actual	Label 0	177	9	3	0	
	Label 1	7	169	4	6	
	Label 2	3	7	166	4	
	Label 3	0	0	1	174	
	F1-score	0.94	0.91	0.94	0.97	Average = 0.94
	10-Fold Cross-Validation Average Score					0.952

The final iteration was performed to compare the performance of logistic regression to nine other classifiers while having multi-labels instead of just two.

Iteration 6: Final Test of Classifiers

The final step was to compare the performance of logistic regression, with the four-label data, when using nine different classifiers: "KNN", "Linear SVM", "RBF SVM", "Decision Tree", "Random Forest", "Neural Net", "AdaBoost", "Naive Bayes", "QDA". The results of the comparison are summarized in Table 4.9. As observed, logistic regression is the classifier with the second highest 10-fold cross-validation average score of 0.952 and third highest f1-average score of 0.94. The highest results (validation of 0.963 and f1-score of 0.97) are obtained by using neural networks. Consequently, logistic regression was chosen as the final machine learning algorithm because it reached the wanted results.

Table 4.9: Classier Comparison Results Iteration 6

	10-Fold Cross-Validation Average Score	F1-score Average
Logistic Regression	0.952	0.94
KNN	0.925	0.92
Linear SVM	0.951	0.95
RBF SVM	0.671	0.64
Decision Tree	0.900	0.90
Random Forest	0.868	0.86
Neural Network	0.963	0.97
AdaBoost	0.646	0.54
GaussianNB	0.712	0.60
QDA	0.943	0.94

Analysis of Final Model

To evaluate the final model, learning curves were plotted along with the Receiver Operating Characteristic (ROC) metrics. The learning curve in Figure 4.7 compares the model's performance on training and testing data over a varying number of examples. Theoretically, the model is said to have learned as much as it could when the training error (bias) and testing error (variance) are at a low value [45]. The maximum learning happened since the error values are around 0.05 out of 1. Consequently, this model had a low training error while being able to generalize on new data (low testing error).

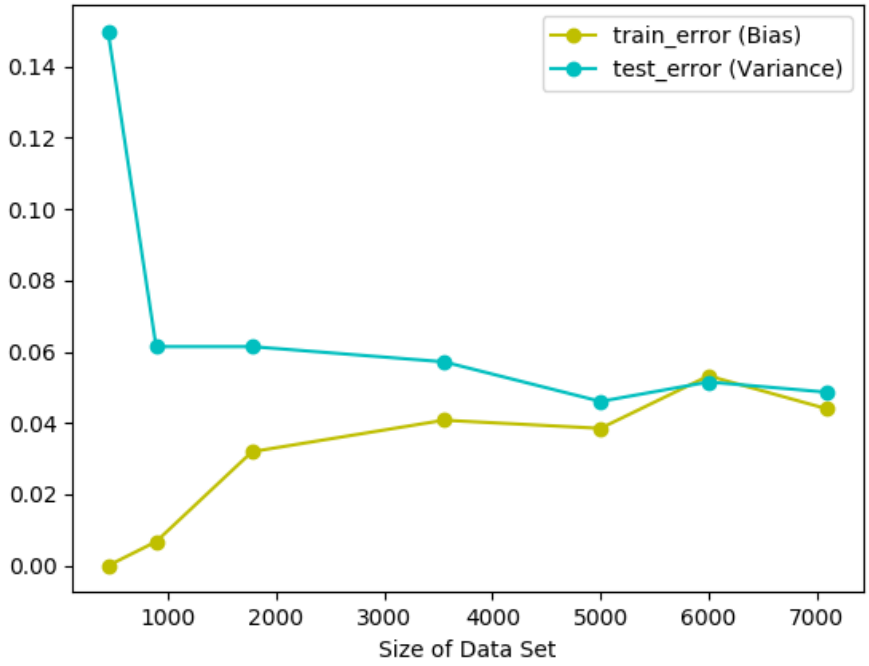

Figure 4.7: Learning Curve

Next, the train scores and test scores were plotted while varying the number of features in Figure 4.8. It can be concluded that the best scores (around 0.95) are obtained when using all 44 features.

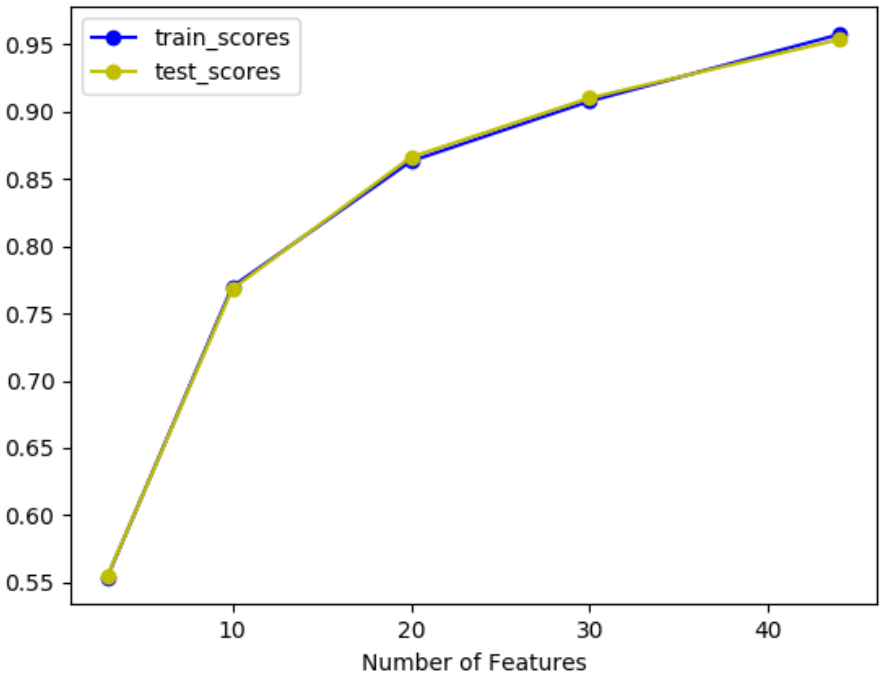

Figure 4.8: Varying number of Features

Finally, the Receiver Operating Characteristic (ROC) curve were plotted in Figure 4.9. The ROC curves display the true positive rate on the Y axis, and the false positive rate on the X axis while varying the discrimination threshold. A classifier usually outputs the probability that an input belongs to a specific class. If the probability for the input at a specific class x is higher than a certain discrimination threshold, the classifier predicts that the input belongs to that class x. While plotting the ROC curve, the discrimination threshold's value is changed, and true and false positive rates are computed for every threshold value. The top left corner of the plot is the "ideal" point, having a false positive rate of zero, and a true positive rate of one. In other words, a larger area under the curve (AUC) is usually better [48]. The "steepness" of ROC curves is also important since it is ideal to maximize the true positive rate while minimizing the false positive rate. The plot in Figure 4.9 has an AUC of 0.99 for label 0, 0.98 for labels 1 and 2, and 1.00, which is the maximum value, for label 3. Hence the proposed model has an excellent performance.

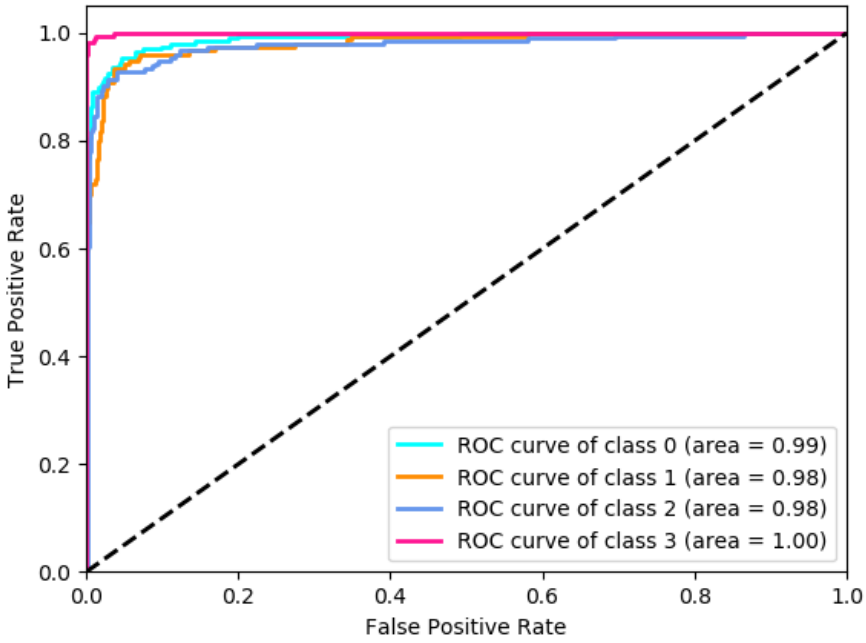

Figure 4.9: ROC Curves

The final test performed on this model was using HTH-infected power data; presented in the following section.

Model Prediction on HTH Data

To conclude this work, the model trained on golden (HTH-free) power data was used to test power data coming from the HTH-infected circuit. The data from the HTH-infected circuit was composed of 21 examples in total collected from encryption operations using K1, K2, K3, K4, and labeled 0, 1, 2, 3 respectively. The prediction of keys obtained from the model were compared to the actual keys used in the AES circuit as shown in Table 4.10. The model was only able to correctly label 19% of the examples from the HTH-infected circuit. More precisely, 75% of the data coming from an encryption using K4 was incorrectly labeled as coming from K1. The incorrect prediction is consistent with the implementation of the HTH described previously, where the HTH was triggered when the encryption operation used K4 and the effect was to use K1 instead. The model was able to detect this change of keys, hence indicating the presence of the triggered HTH. In addition to that, even when the HTH was not triggered (when using K1, K2, and K3) the model incorrectly predicted 81% of the labels. The incorrect predictions by the model allowed to detect the presence of the HTH even when it was not triggered. The incorrect predictions can be explained by the differences in power the HTH produced when constantly checking if the trigger was satisfied or not.

Table 4.10: HTH Prediction Results

Actual Label	Predicted label	Result	Probability of prediction
0	3	FALSE	0.605
0	0	TRUE	0.626
0	0	TRUE	0.815
0	3	FALSE	0.647
0	3	FALSE	0.969
1	3	FALSE	0.505
1	2	FALSE	0.361
1	3	FALSE	0.506
1	2	FALSE	0.855
1	2	FALSE	0.682
1	2	FALSE	0.999
2	3	FALSE	0.676
2	3	FALSE	0.545
2	2	TRUE	0.903
2	3	FALSE	0.532
2	3	FALSE	0.481
3	3	TRUE	0.901
3	0	FALSE	0.926
3	0	FALSE	0.785
3	0	FALSE	0.629
3	1	FALSE	0.564

CHAPTER 5: Summary and Future Work

A side-channel power analysis technique using machine learning was presented for HTH detection. The approach used the power traces from a golden implementation of the AES encryption algorithm on an FPGA to train a logistic regression model. The obtained model was then tested on new power data and was able to make correct predictions with a 95% accuracy. Next, an HTH (of a few gates) was implemented in the AES circuit which was triggered when K4 was used in the encryption and its effect was denial-of-service along with breach of plaintext secrecy. The power data from the HTH-infected circuit were collected and tested on the trained logistic regression model. The predictions made by the model allowed the detection of the HTH even when it was not triggered. In fact, 81% of the HTH-infected data was detected as flawed by the logistic regression model. This work was a proof of concept that verified that machine learning along with side-channel power data provided enough information for HTH detection. The work can be developed further to include more features namely in the frequency domain. Also, more keys could be considered to emulate real life encryption situations. Moreover, different types and sizes of HTH could be implemented then tested to figure out the threshold at which the trained model stops being able to detect an HTH.

Finally, an initial phase can be added to the mainstream manufacturing flow as shown in Figure 5.1. This “motivation” stage should be added before the specification stage and hence start the flow. This additional step would take into consideration the strategies and incentives of the different key players in the manufacturing flow and hence complete the picture. The “motivation” phase would act as a defensive stage where the IC designer takes into consideration malicious incentives of an adversary before starting the process. As a

result, the designer would have an advantage over the adversary instead of waiting for an attack to then start the recovery and detection process.

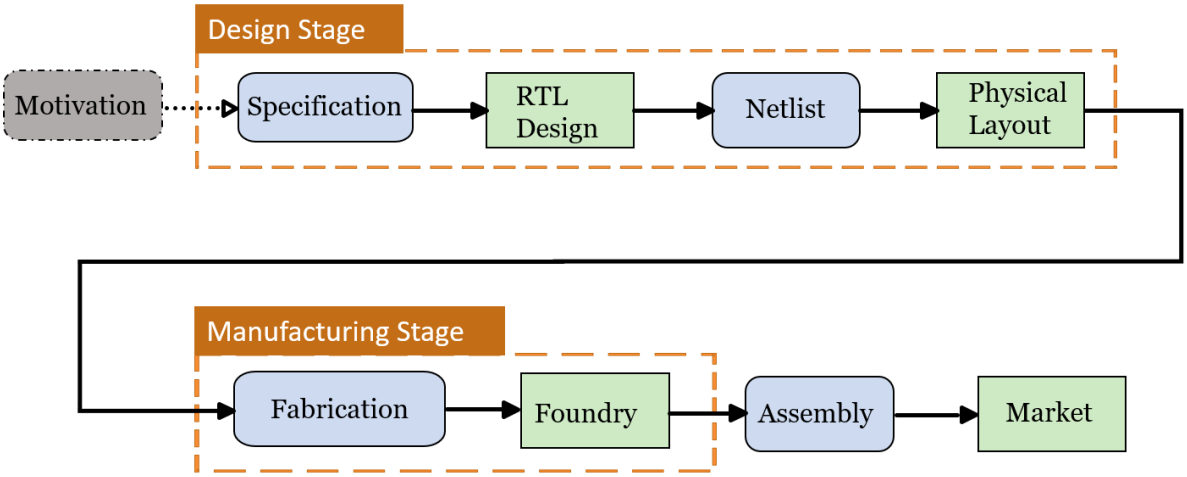

Figure 5.1: Additional "Motivation" Stage to Mainstream Manufacturing Flow

This idea was inspired by [29] where the author took into consideration the strategies and incentives of the adversary and the designer using game theory.

This additional step would mean that the designer will take into consideration what information is critical and would interest an adversary. The mainstream chip production would then start while having these malicious incentives in mind. As a result, IC design happens in a defensive manner as opposed to waiting for an attack to start the recovery and detection process. As the authors in [4] specify, designers make the first and last move in the manufacturing process which gives them an advantage over adversaries.

References

[1] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, Orlando, FL, 2008.

[2] D. F. Y. J. R. K. S. B. a. M. T. K. Xiao, "Hardware Trojans: Lessons learned after one decade of research," in *ACM Transactions on Design Automation of Electronic Systems*, 2016.

[3] S. Bhunia, M. Hsiao, M. Banga and S. Narasimhan, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," in *Proceedings of the IEEE*, 2014.

[4] M. Potkonjak , A. Nahapetian, M. Nelson and T. Massey, "Hardware Trojan Horse Detection Using Gate-Level Characterization," in *Design Automation Conference*, San Francisco, California, USA , 2009.

[5] M. Tehranipoor, *Protecting Electronics Supply Chain from Design to Resign,* Washington DC: IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2017.

[6] B. H. T. S. H. e. a. Shakya, "Benchmarking of Hardware Trojans and Maliciously Affected Circuits," *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85-102, 2017.

[7] D. J. Greaves, "cl.cam.ac.uk," University of Cambridge, Computer Laboratory, 2011. [Online]. Available: http://www.cl.cam.ac.uk/teaching/1011/SysOnChip/socdam-notes1011.pdf.

[8] X. Zhang and M. Tehranipoor, "Case Study: Detecting Hardware Trojans in Third-Party Digital IP Cores," in *IEEE International Symposium on Hardware-Oriented Security and Trust*, San Diego, California, USA, 2011.

[9] J. A. Roy, F. Koushanfar and I. . L. Markov, "Extended Abstract: Circuit CAD Tools as a Security Threat," in *IEEE International Workshop on Hardware-Oriented Security and Trust*, Anaheim, CA, USA, 2008.

[10] S. BHASIN, J.-L. DANGER, S. GUILLEY, X. T. NGO and L. SAUVAGE, "Hardware Trojan Horses in Cryptographic IP Cores," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Santa Barbara, CA, USA, 2013.

[11] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Cryptographic Hardware and Embedded Systems Workshop (CHES)*, Leuven, Belgium, 2012.

[12] Chief, Access and Target Development (S3261), NSA/CSSM 1-52, "spiegel.de," 2010. [Online]. Available: http://www.spiegel.de/media/media-35669.pdf.

[13] "Trust Hub," [Online]. Available: http://trust-hub.org/benchmarks.php.

[14] D. Forte, C. Bao and A. Srivastava, "Temperature tracking: An innovative run-time approach for hardware Trojan detection," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, USA, 2013.

[15] A. Kulkarni, Y. Pino and T. Mohsenin, "Adaptive Real-time Trojan Detection Framework through Machine Learning," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, USA, 2016.

[16] C. Lamech, R. M. Rad, M. Tehranipoor and J. Plusquellic, "An Experimental Analysis of Power and Delay Signal-to-Noise Requirements for Detecting Trojans and Methods for Achieving the Required Detection Sensitivities," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3, pp. 1170 - 1179, 2011.

[17] S. Wei and M. Potkonjak, "Scalable Hardware Trojan Diagnosis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1049 - 1057, 2011.

[18] . S. Narasimhan, D. Du, R. S. Chakraborty, S. Paul, F. G. Wolff, C. A. Papachristou, K. Roy and S. Bhunia, "Hardware Trojan Detection by Multiple-Parameter Side-Channel Analysis," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 2183 - 2195, 2012.

[19] S. Narasimhan, D. Du, R. S. Chakraborty, S. Paul, F. Wolff, C. Papachristou, K. Roy and S. Bhunia, "Multiple-parameter side-channel analysis: A non-invasive hardware Trojan detection approach," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Anaheim, CA, USA, 2010.

[20] C. Bao, Y. Xie and A. Srivastava, "A security-aware design scheme for better hardware Trojan detection sensitivity," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, Washington, DC, USA, 2015.

[21] A. Waksman, M. Suozzo and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in *ACM SIGSAC conference on Computer & communications security*, Berlin,

Germany, 2013.

[22] M. Banga, M. Chandrasekar, L. Fang and M. S. Hsiao, "Guided test generation for isolation and detection of embedded trojans in ics," in *18th ACM Great Lakes symposium on VLSI*, Orlando, Florida, USA, 2008.

[23] S. Narasimhan, X. Wang, D. Du, R. S. Chakraborty and S. Bhunia, "TeSR: A robust Temporal Self-Referencing approach for Hardware Trojan detection," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, San Diego CA, USA, 2011.

[24] I. Wilcox, F. Saqib and J. Plusquellic, "GDS-II Trojan detection using multiple supply pad VDD and GND IDDQs in ASIC functional units," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, Washington, DC, USA, 2015.

[25] I. Exurville, L. Zussa, J.-B. Rigaud and B. Robisson, "Resilient hardware Trojans detection based on path delay measurements," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, Washington, DC, USA, 2015.

[26] L.-w. Wang, H. Xie and H.-w. Luo, "A novel analysis method of power signal for integrated circuits Trojan detection," in *20th IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*, Suzhou, China, 2013.

[27] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi and B. Sunar, "Trojan Detection using IC Fingerprinting," in *IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, 2007.

[28] J. Li and J. Lach, "At-speed delay characterization for IC authentication and Trojan Horse detection," in *IEEE International Workshop on Hardware-Oriented Security and Trust*, Anaheim, CA, USA, 2008.

[29] J. Graf, "Trust games: How game theory can guide the development of hardware Trojan detection methods," in *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, USA, 2016.

[30] H. Salmani and M. Tehranipoor, "Analyzing circuit vulnerability to hardware Trojan insertion at the behavioral level," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, New York City, NY, USA, 2013.

[31] D. R. Collins , "Trust in Integrated Circuits," in *Government Microcircuit Applications and Critical Technology Conference (GOMACTech)*, Las Vegas, NV, USA, 2008.

[32] H. Guntur, J. Ishii and A. Satoh, "Side-channel Attack User Reference Architecture Board SAKURA-G," in *IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, Tokyo, Japan, 2014.

[33] SAKURA Hardware Security Project, "SAKURA-G Side-channel AttacK User Reference Architecture - Specifications," MORITA TECH CO., LTD, 2013.

[34] XILINX INC, "ISE WebPACK Design Software," Xilinx All Programmable, 2018. [Online]. Available: https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html.

[35] Xilinx All Programmable, "ISE Design Suite," Xilinx Inc, 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/isehelp_start.htm#xpa_c_overview.htm.

[36] Tektronix, "Digital Storage Oscilloscopes TDS2000C Series Datasheet," 2013. [Online]. Available: https://www.tek.com/sites/default/files/media/media/resources/TDS2000C-Digital-Storage-Oscilloscope-Datasheet-4_1.pdf.

[37] H. Salmani, M. Tehranipoor and R. Karri, "On Design vulnerability analysis and trust benchmark development," in *IEEE Int. Conference on Computer Design (ICCD)*, 2013.

[38] E. Milanov, "The RSA Algorithm," 2009. [Online]. Available: https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf.

[39] A. S. a. L. A. R. Rivest, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, p. 120–126, 1978.

[40] Satoh Lab, "SAKURA-G," SAKURA hardware security project , 2014. [Online]. Available: http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html.

[41] . A. Kak, "Lecture 8: AES: The Advanced Encryption Standard," Purdue University, 2018. [Online]. Available: https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf.

[42] Satoh Laboratory, "Side-channel AttacK User Reference Architecture SAKURA-G Quick Start Guide," The University of Electro Communications, 2014.

[43] Online Domain Tools, "AES – Symmetric Ciphers Online," 2018. [Online]. Available: http://aes.online-domain-tools.com/.

[44] M. Joye and F. Olivier, Encyclopedia of Cryptography and Security, H. C. A. van Tilborg and S. Jajodia, Eds., Boston, MA, USA: Springer International Publishing AG, 2017.

[45] A. Ng, *Machine Learning Course*, Stanford University, Coursera, 2017.

[46] I. A. F. K. O. H. F. A. a. S. R. H. F. K. Lodhi, "A Self-learning Framework to Detect the Intruded Integrated Circuits," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, Montreal, QC, Canada, 2016.

[47] RapidMiner, Inc., "RapidMiner Studio, Visual Workflow Designer For Data Scientists," 2017. [Online]. Available: https://rapidminer.com.

[48] Blondel et al., "scikit-learn, Machine Learning in Python," 2017. [Online]. Available: http://scikit-learn.org/stable/.

[49] Pedregosa et al., "Scikit-learn: Machine Learning in Python," in *JMLR 12*, 2011.

[50] J. N. v. R. B. B. L. T. Joaquin Vanschoren, "OpenML: networked science in machine learning," in *SIGKDD Explorations 15(2)*, 2013.

[51] B. Bernd Klein, "Neural Networks with scikit," python-course.eu, 2018. [Online]. Available: https://www.python-course.eu/neural_networks_with_scikit.php.

[52] S. Joglekar, "Linear and Quadratic Discriminant Analysis for ML / statistics newbies," wordpress, 2015. [Online]. Available: https://codesachin.wordpress.com/2015/08/25/linear-and-quadratic-discriminant-analysis-for-ml-statistics-newbies/.

[53] TKJ Electronics, *FPGA Xilinx VHDL Video Tutorial*, 2011.

[54] E. Stavinov, "Using Xilinx Tools in Command-Line Mode," 2011. [Online]. Available: http://outputlogic.com/xcell_using_xilinx_tools/74_xperts_04.pdf.

[55] Xilinx, "Command Line Tools User," 2009. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/devref.pdf.

[56] ufrisk, *PCILeech SP605 / FT601 PCIe to USB3*, 2017.

[57] Doulos, *Scripting Xilinx® ISE™ using Tcl,* 2008.

[58] Tektronix INC, "TDS2000C Digital Storage Oscilloscope," [Online]. Available: https://www.tek.com/oscilloscope/tds2000-digital-storage-oscilloscope.

[59] Tektronix, "tek.com," [Online]. Available: https://de.tek.com/sites/default/files/media/media/resources/3GW_25645_3.pdf.

[60] O. F. Joye M., Side-Channel Analysis, Boston, MA: van Tilborg H.C.A., Jajodia S. (eds) Encyclopedia of Cryptography and Security, Springer, 2011.

[61] Wikipedia, the free encyclopedia, "RSA (algorithm)," 2018. [Online]. Available: https://simple.wikipedia.org/wiki/RSA_(algorithm).

Appendix A: Synthesizing Circuits on SAKURA-G

To start with synthesizing a “counter” circuit on the SAKURA-G FPGA, the following are the steps to perform [53]. First a Xilinx project was created, then a VHDL source file was added. In the source file, VHDL code was written for a “counter” application. The code lit up the FPGA’s LEDs based on the internal counter values, and the counter would reset when a certain push button was pressed. Second, a constraints file was created to map the LED and pushbutton variables used in the VHDL code to the actual hardware components on the FPGA. The pin numbers were obtained from [40].Third, the counter circuit was synthesized into a bit stream configuration file in the Xilinx tool.

To upload the bit-stream file to the FPGA using the iMPACT tool embedded within Xilinx, a USB cable is needed from the connecter CN6 on SAKURA-G to the PC, and a configuration cable to the JTAG connector CN2 for the main FPGA (if needed the controller’s connector was CN4) as shown in Figure 2.2. Finally, the iMPACT tool was used to assign the bit stream programming file as the FPGA’s configuration file, and finally program the FPGA. As a result, the LEDs started flashing giving the expected output of an incrementing counter. Also, the reset push button would reset the counter value, when pressed, and hence was functioning correctly.

Appendix B: Tcl Script for Automation of Synthesis and Implementation

Xilinx ISE is an EDA tool which can be controlled using the Tool Command Language (Tcl) and comes with a customized Tcl distribution [54]. After researching the different scripting language options, it was concluded that Tcl was an easy to use scripting language and an industry standard popular in the electronic design automation (EDA) industry [55]. As a result, a Tcl script was written to automate the creation of a configuration file for the FPGA from source and constraint files and then the upload of the generated bit stream file to the SAKURA-G board [56]. The script created a Xilinx project, set its properties, then synthesized, translated, mapped, placed and routed, and implemented the design. As the last step, the code which uploaded the configuration file to the FPGA using iMPACT [57] was integrated with the Tcl file. Consequently, the synthesis and implementation of any circuit of the FPGA only required the use of the "xtclsh" command in the ISE Design Suite Command Prompt, while specifying the name of the Tcl file. The "xtclsh" command starts the Tcl shell in Xilinx ISE [54].

Appendix C: Power Collection from Tektronix TDS2022C Oscilloscope

The Tektronix TDS2022C oscilloscope is a digital real-time sampling architecture which can accurately see small signal details [58]. The oscilloscope is packed with USB connectivity, 16 automated measurements, limit testing, and a built-in help menu. To collect power data using a Tektronix TDS2022C oscilloscope, a USB memory device is needed. The oscilloscope's USB host port on the front panel enabled the saving of power waveforms in the memory device. The built-in Data Logging feature was used to set up the oscilloscope and have it save user-specified triggered waveforms to the USB. The data recorded on the USB are saved in a CSV file which contains a maximum of 2,500 samples.

Appendix D: RapidMiner Blocks

RapidMiner is a data science platform which provides automated machine learning to accelerate building predictive models [47]. To build training and testing stages using the logistic regression algorithm, different blocks need to be used in RapidMiner. Every block has a specific functionality. In the training stage, the “Retrieve” operator was used to access the stored input training file of features. The “Set Role” operator was to set the role of the fifth column in the input file of features to be the labels’ column. Finally, the “Logistic Regression” operator was the machine learning algorithm. After training, the testing stage included another “Retrieve” operator to access the testing file of features in the repository and load it into the process. The “Apply Model” operator was used to test the already trained model on another data set and get a prediction on unseen data.