3-2023

Characterizing Location-Based Electromagnetic Leakage of Computing Devices using Convolutional Neural Networks to Increase the Effectiveness of Side-Channel Analysis Attacks

Ian C. Heffron

CHARACTERIZING LOCATION-BASED ELECTROMAGNETIC LEAKAGE OF COMPUTING DEVICES USING CONVOLUTIONAL NEURAL NETWORKS

THESIS

Ian Carter Heffron, Captain, USAF

AFIT-ENG-MS-23-M-030

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-23-M-030

CHARACTERIZING LOCATION-BASED ELECTROMAGNETIC LEAKAGE OF COMPUTING DEVICES USING CONVOLUTIONAL NEURAL NETWORKS TO INCREASE THE EFFECTIVENESS OF SIDE-CHANNEL ANALYSIS ATTACKS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Science

Ian Carter Heffron, B.S.C.S.

Captain, USAF

March 23, 2023

AFIT-ENG-MS-23-M-030

CHARACTERIZING LOCATION-BASED ELECTROMAGNETIC LEAKAGE OF COMPUTING DEVICES USING CONVOLUTIONAL NEURAL NETWORKS TO INCREASE THE EFFECTIVENESS OF SIDE-CHANNEL ANALYSIS ATTACKS

THESIS

Ian Carter Heffron, B.S.C.S.
Captain, USAF

Committee Membership:

Lt Col James W. Dean, Ph.D
Chair

Scott R. Graham, Ph.D
Member

Gilbert L. Peterson, Ph.D
Member

Abstract

Side-Channel Analysis (SCA) attacks aim to recover secret information, often a cipher key, from a target device without direct interaction. Common attacks focus on power-based leakage, or electromagnetic (EM)-based leakage. Deep learning and neural networks have recently gained in popularity as successful tools in SCA attacks and research. Near-field EM probes with high-spatial resolution enable researchers and attackers to isolate physical locations above the surface of a processor. This enables attackers to exploit the spatial dependencies of algorithms processes due to the geometry of the attacked device. These spatial dependencies result in different physical locations above a chip emanating different signal strengths related to the secret information. The strengths of different locations can be mapped using the performance of a neural network trained to detect secret information on near-field leakage data. The contribution focuses on using this mapping to identify ideal near-field leakage collection locations from which to conduct an attack. This thesis demonstrates the effectiveness of this technique in reducing the time needed to conduct a successful EM SCA attack. The effectiveness of this technique is demonstrated first on the ChipWhisperer Lite Atmel XMEGA microcontroller, a platform designed as a teaching device for SCA work, and then on the Xilinx Kintex-7 field programmable gate array (FPGA) a device that makes power-based SCA less effective as it encrypts all plaintext bytes simultaneously. An increase of effectiveness of the attacks is demonstrated against the Atmel XMEGA target by 283%, and an increase in the effectiveness of our attacks against the Xilinx Kintex-7 target by 33.4% by attacking the point corresponding to the greatest performance of the Convolutional Neural Network (CNN) as opposed to the point with the worst performance of the CNN.

Table of Contents

Page

List of Figures

Figure Page

List of Tables

CHARACTERIZING LOCATION-BASED ELECTROMAGNETIC LEAKAGE OF COMPUTING DEVICES USING CONVOLUTIONAL NEURAL NETWORKS TO INCREASE THE EFFECTIVENESS OF SIDE-CHANNEL ANALYSIS ATTACKS

I. Introduction

As part of their normal operation, electronic devices emit electromagnetic (EM) energy that can be detected by, and interfere with, other electronic devices [2]. This interference can cause unintended effects, and is why you are required to turn electronic devices off on airplanes prior to takeoff and landing. In the past decade there has been increasing interest in these unintended emissions. Specifically there has been increased interest in capturing these emissions and determining if any context can be drawn from them. This increase in interest is the basis for EM Side-Channel Analysis (SCA) attacks and has potentially severe implications for the physical security of sensitive computing equipment, especially if the context drawn enables an attacker to learn specific information about the underlying operations of the sensitive equipment, including secret information such as a cryptographic key [3].

The field of SCA research is broad, however all SCA work focuses on the ability of a researcher or an attacker to infer information from a device in a manner in which they were not intended to. This information leakage can take many forms; radiated EM emissions, observed power draw over time, the time a process takes to run, listening to the vibrations created by an observed device, using the memory of a processor to sniff sensitive information, or even simple shoulder surfing techniques [3]. SCA attacks can target phone lock screen PINs, credit card numbers, passwords, cipher keys, or any other sensitive information that a user or organization wants to keep

secret. A small subset of SCA research, and the focus of this thesis is to find faster ways to ascertain a secret cipher key from the operations of a microcontroller or other computing chip.

Many factors can influence the leakage model of a given target. These factors include chip-set architecture, both physical and logical, manufacturing tolerances, fabrications techniques, surrounding components, power draw and many others [3]. This means that the physical location from which an attacker chooses to collect EM leakage data is important when conducting SCA attacks. Different electronic components within a given architecture create different signals and different noise. The point at which the leaked EM data provides the strongest signal is not obvious and thus frameworks must be created to identify said point. This research uses a neural network to identify the most effective location on the surface of a chip to extract key information, and compares that to the relative signal strengths at each collection point. This demonstrates that the identified hottest point does in fact enable quicker key recovery for an attacker.

1.1 Security Implications of Side-Channel Leakage

Side-channel leakage is concerning to researchers, device developers, and end users, as it presents an avenue for an adversary to gain access to a device or secretive information that would otherwise be unavailable to them. Data leakage via side-channels may include secret variables such as a cryptographic key, user passwords, or PIN numbers. It may also reveal information regarding the underlying implementation of the device which would allow a competitor to reverse engineer proprietary algorithms or other critical intellectual properties. For these reasons, side-channel leakage presents a practical security threat to many electronic systems, especially systems that employ cryptographic algorithms to protect information. This security threat means that de-

velopers can no longer operate under the naive assumption that only the input and output variables are present. They must look to develop more secure platforms that use side-channel countermeasures. Further, more research must be conducted into developing countermeasures to side-channel leakage.

1.2 Motivation

Modern cryptographic algorithms such as Advanced Encryption Standard (AES) operate under the assumption that the cryptographic key is completely secure. They provide security in that an attacker cannot derive the key using the known input *plaintext* and output *ciphertext*. This assumption holds true outside of the context of side-channel leakage, either in the form of the power side-channel or the EM side-channel [4]. Many researchers have already demonstrated the existence of both EM and power side-channel information leakage that enables successful retrieval of cryptographic keys [5, 6, 7, 2]. Much of current research is focused on both speeding up the retrieval timeline of SCA attacks, as well as the success rate of SCA attacks by improving side-channel attack methodologies. This includes using neural networks and other algorithms that have allowed attackers to defeat more traditional countermeasures that would have previously caused statistical analysis attacks such as the Welch's t-test or the Pearson's χ^2-test to give false negatives [8]. While power SCA attacks typically perform better than EM attacks, as power traces often have a higher signal-to-noise ratio (SNR), they also require physical modification to the device under test (DUT). This physical modification may not always be possible, and thus EM SCA attacks may be more viable in certain circumstances. Hardware implementations of AES may also enable chips to perform encryption operations on all bytes of a plaintext and key simultaneously, which would create extra noise and can prevent success when performing power-based SCA attacks. Tied to the physical

location of different computing components, the SNR of EM leakage will be different in different locations above the chip. This suggests that if hardware implementations of AES that encrypt different bytes simultaneously accomplish that encryption in different physical locations, then perhaps EM SCA attacks may outperform power SCA attacks in this instance. Additionally, most SCA attacks focus on one key byte at a time. While this is time-consuming, if an attacker is able to successfully retrieve all key bytes, they can decrypt anything on the device that has been encrypted with the secret key. However as mentioned, this is time-consuming. If it takes an attacker hours to train a model for an attack, and there are 16 key bytes to retrieve, then the process to retrieve an entire key is arduous and long. This leads researchers to attempt different methods of identifying ideal collection locations to increase the speed of and maximize the success rate of side-channel attacks. This research seeks to further this specific area of SCA research. Can we identify points where the leakage data at that point , enables both quicker and more successful key byte recovery?

1.3 Hypothesis

The hypothesis of this effort is that the performance of a neural network trained to detect secret information leaked in near-field EM radiation is correlated to the relative strength of the EM radiation associated with the secret information. As a result, this property can be leveraged to construct a heatmap that will enable attackers to identify optimal physical locations from which to collect the leaked signals and shorten the time required to successfully attack the key.

1.4 Research Objectives

The primary goal of this research is to determine whether a Convolutional Neural Network (CNN) can identify an ideal point or points to collect EM leakage in order

to improve attack success. This will be investigated by collecting EM leakage data in the form of traces in a grid pattern above a Xilinx field programmable gate array (FPGA). The leakage traces are then combined into a single large training dataset and a series of CNNs will be trained on the collective data. The resulting models will be tested for classification accuracy using data collected at each of the collection points. The results will then be combined into a heatmap and confidence intervals will be created to identify the points that are statistically significant from one another. The EM data from the identified points will then be used to conduct attacks to determine if the attacks succeed more often at the points identified as having a higher accuracy. This described framework can then be used by attackers and security professionals to identify collection points from which to conduct SCA attacks more successfully.

1.5 Our Contribution

In this research, the main objective is to develop and demonstrate the effectiveness of a framework designed to identify ideal leakage collection points against which to conduct EM SCA attacks. This framework seeks only to demonstrate that neural networks can be used, given enough input data, to classify points as being statistically stronger or weaker, with regards to leaked signal strength, in the case of an EM SCA attack. The initial research attacks a known leaky device, the ChipWhisperer Atmel XMEGA using a software implementation of AES. Once the framework was demonstrated as successful on a known target, the effort focused on a Xilinx Kintex-7 FPGA using a hardware implementation of AES. There are likely better neural network architectures for the problem, and perhaps there may be better means for determining ideal EM leakage collection locations above a computing device. However, this work demonstrates that the presented framework enables an attacker or security expert to map out the EM leakage of a computing device to identify a point or points from

which the collected leaked signal is stronger and leads to quicker SCA key recovery attacks.

1.6 Document Organization

This document is organized as follows. Chapter II provides an overview of relevant background information and previous research. Chapter III details the collection process used to obtain our trace data, the architecture used to train the CNNs on the collected data, and the methods used to compare the performance of the neural network when tested at the various collection grid points. This chapter also describes the techniques used in the SCA attack that enable individual key byte retrieval from the model. Chapter IV presents the results of the training process, and the analysis of the attack results. Finally, Chapter V discusses the conclusions drawn from the results as well as areas of interest for future research.

II. Background and Literature Review

Deep Learning-based Side-Channel Analysis (SCA) research is a multi-disciplinary field that relies on techniques pulled from other research domains including; Cryptography, Deep Learning, Electromagnetic Theory, Statistical Analysis, Profiling Attacks, Leakage Assessment.

The remainder of this chapter is outlined as follows. Section 2.1 outlines leakage assessment methodologies, including statistical analysis tools such as the Welch's t-test and the Pearson's χ^2-test. Next, the idea of *Perceived Information*, and *Mutual Information* is presented. Then,profiled SCA attacks are discussed in Section 2.2 to demonstrate how two separate devices are used in many SCA attacks, including those presented in this work. electromagnetic (EM) theory is briefly discussed in Section 2.3, as this theory is what enables EM-based SCA work. Section 2.4 discusses cryptography, including an introduction to and motivation for the use of the Advanced Encryption Standard (AES) encryption algorithm for this research, including the differences between hardware-based and software-based encryption. Next, Section 2.5 discusses relevant deep learning techniques and their use in leakage assessment. Finally, Section 2.7 discusses various research that has already been accomplished regarding the field of SCA work as well as the use of Convolutional Neural Network (CNN)s for the purpose of SCA work.

2.1 Leakage Assessment

This section presents a set of techniques that can be used to assess a given target for information leakage. A device is said to "leak" information if a correlation can be found between the data being processed and an observable external phenomenon. This research focuses on information leaked through near-field electromagnetic ra-

diation. Essentially, leakage assessment is the set of techniques used to determine if a collected observed phenomena is data- or information-dependent. All electronic equipment draws power in an amount that is dependent upon the operations occurring on the device. Additionally, all electronic equipment emits some amount of EM signal due to the nature of the manufacturing process and the physical implementation of the underlying components making up the device under test (DUT).

Leakage assessment is used to determine if any information can be extracted from a collected EM or power signal. The beginning of leakage assessment involved an exhaustive verification of the ability of the DUT to resist known attacks while simultaneously covering a broad range of intermediate values and hypothetical leakage models. This became less feasible over time as leakage models became more complex and countermeasures were introduced to defeat known SCA attacks. Additionally, this exhaustive approach often produced false negatives when applied to newer leakage models [8]. At the National Institute of Standards and Technology (NIST) Non-Invasive Attack Testing Workshop in 2011, the Welch's t-test was suggested for leakage detection [9]. The idea being that statistical analysis could determine much more quickly and with fewer false negatives, if there was a difference between collected leakage data generated with known input data and collected leakage data generated with randomized input data. Seven years later, the Pearson's χ^2-test was suggested to replace the Welch's t-test [10].

2.1.1 Statistical Analysis

Statistical analysis is a process used to determine if two or more datasets have the same or different statistical distributions. More broadly, this process tests if the datasets differ significantly from one another. More specifically, the purpose of statistical analysis tests is to find evidence that enables rejection of a null-hypothesis.

Null hypotheses can range from stating that two datasets are indistinguishable, to a sample mean being the same as a population mean, to two different variables being independent. The alternate hypothesis is the opposite of the null hypothesis. For the examples, these would be that the two datasets are distinguishable, the sample mean is not the same as the population mean, and the two variables are dependent.

2.1.2 Welch's t-Test

The aim of the t-test is to provide a quantitative probability of whether or not the mean μ of two sets are different. In this sense, the hypothesis for the t-test is that the μ for the two sets is in fact different. This means the null hypothesis would be that the μ for the sets is indistinguishable [11]. In the case of SCA, these two datasets are the observed leakage traces compared when supplying the tested device with either two sets of fixed input, or one set of fixed input and one set of randomized input. For the t-test, we have sets Q_0 and Q_1 with means μ_0 and μ_1, standard deviations s_0 and s_1, and cardinalities n_0 and n_1. The t-test statistic t and the degrees of freedom v is calculated as:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}} \tag{1}$$

$$v = \frac{(\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1})^2}{\frac{(\frac{s_0^2}{n_0})^2}{n_0 - 1} + \frac{(\frac{s_1^2}{n_1})^2}{n_1 - 1}} \tag{2}$$

By using the calculated v, and the t-distribution table mentioned in 2.1.1 the critical value for the t-test can be found. To determine the critical value, the degrees of freedom v and the alpha level, or confidence level, α are intersected on the t-distribution table. The most common alpha level is 5% or 0.05, indicating 95% certainty in the result. The critical value is the value used to determine whether to

reject the null hypothesis or not. If the absolute value of the test statistic $|t|$ is greater than the critical value returned by the t-distribution table, then the null hypothesis can be rejected, leading to acceptance of the alternate hypothesis that the two sets are distinguishable [8, 11].

2.1.3 Pearson's Chi-Squared-Test

The Pearson's χ^2-test is a statistical analysis test that allows for higher-order analyses and can provide evidence that one dataset is dependent on another dataset. While the t-test captures information in a single moment and focuses on the difference in means of two sets of data, the χ^2-test can capture information that exists in multiple statistical moments. A moment refers to a quantitative measure of a function's graph. The Pearson's χ^2-test is used to evaluate whether there is enough evidence to reject the null hypothesis stating that two categorical variables are independent. Thus the alternate hypothesis would be that the two categorical variables are dependent [8].

To calculate the χ^2 test statistic, one must first construct a contingency table F from the two (or more) sets Q_0 and Q_1.

$$F = \begin{bmatrix} Q_{0_0} & Q_{0_1} & \dots & Q_{0_c} \\ Q_{1_0} & Q_{1_1} & \dots & Q_{1_c} \end{bmatrix} \tag{3}$$

The numbers of rows is the number of compared sets, and is denoted as r. The number of columns is the number of bins in the histogram created from the compared sets, and is denoted as c. This table enables a greater look into the variance of the categorical variables. The test statistic χ^2 and the degrees of freedom v are computed as:

$$\chi^2 = \sum_{i-0}^{r-1} \sum_{j=0}^{c-1} \frac{(F_{ij} - E_{ij})^2}{E_{ij}} \tag{4}$$

$$v = (r-1)(c-1) \tag{5}$$

Where E_{ij} denotes the theoretical frequency for a cell, assuming the hypothesis that each cell is independent. E_{ij} is calculated

$$E_{ij} = \frac{(\sum k = 0^{c-1} F_{ik})(\sum_{k=0}^{r-1} F_{kj})}{N} \tag{6}$$

The confidence p to accept or reject the null hypothesis for the χ^2-test is given from the χ^2 probability density function, where Γ denotes the gamma function [8].

$$p = \int_x^\infty f(x, v)dx \tag{7}$$

$$f(x,v) = \begin{cases} \frac{x^{\frac{v}{2}-1}e^{-\frac{x}{2}}}{2^{\frac{v}{2}}\Gamma(\frac{v}{2})} & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The benefit of the χ^2-test is its ability to extend to more datasets than just the two that the t-test can work with. Generally, the chi^2-test outperforms the t-test in cases where the DUT uses a masked implementation with low noise levels, or where masking schemes cause leakage in multiple moments due to physical defects [8][10][12].

2.2 Profiled Side-Channel Attacks

This section discusses a form of SCA attack, known as a profiling attack. Profiling refers to the ability of an attacker to identify points in a trace, be it power or EM, that correspond to certain parts of a cryptographic algorithm. The attacker can separate out parts of the trace with high variance as they are more data dependent, and parts of the trace that have a low variance as they are more likely key dependent as the key is constant. An attacker can then use this knowledge, or profile, to perform inferential

power analysis. Inferential analysis takes a profile and extends it to characterize the device in question. An attacker can reveal the location of key bit manipulations in a trace by comparing the power consumption of the different rounds as opposed to the power consumption of individual operations, and this can help them determine which power level corresponds to which logic value of the key bit. In other words they can profile a device and then build a template for the key bits [13].

Profiling SCA attacks are split into two phases: a profiling phase, and an attack phase. These phases are also referred to as training and matching phases respectively. During the profiling phase, the attacker entirely controls all operations occurring on the profiled device and generates either EM or power traces by running randomized plaintexts and keys through an encryption algorithm and observing the operations of the profiled device. Profiling attacks can be broken down into the following steps:

- A set of N_p profiling traces are gathered and are considered the realization of the random variable $S_p \triangleq \{(x_1, z_1), (x_2, z_2), ..., (x_{N_p}, z_{N_p})\}$ where x_i is the i.i.d. realization of the observed random variable (X), and z_i is the i.i.d. realization of the discrete random variable, or the output of the cryptographic primitive (Z). S_p is represented by the probability mass function $Pr[X, Z]^{N_p}$.

- The gathered observed traces N_p are then used to build a model M that will return a set of scores for each hypothetical Z. This model can then be used in the attack phase.

- A set of N_a attack traces are captured on the target device and these traces are seen as a realization of $S_a \triangleq (k^*, \{(x_1, p_1), (x_2, p_2), ..., (x_{N_a}, z_{P_a})\})$ where k^* is the correct key byte value, and $k^* \in K$. K represents the set of all possible key byte values. For all $i \in [1, N_a]$, the plaintext used in the ith trace $p_i \sim Pr[P]$, and $x_i \sim Pr[X|Z = C(p_i, k^*)]$ where $C(p_i, k^*)$ represents the output of the

cryptographic primitive given p_i, and k^*.

- The model M is used to create a prediction vector for every attack trace, $q_i = M(x_i), i \in [1, N_a]$. For each trace, the model assigns a score to each possible key byte value, that is, for every $j \in [0, |K|]$, the value of the jth index in q_i corresponds to the likelihood assigned to that byte value by the model.

- The output scores are then combined over all given attack traces to output a final likelihood for each key byte candidate in K. The candidate with the highest final likelihood is the predicted correct key byte. This maximum likelihood score for each key candidate is calculated $l[k] = \sum_{i=1}^{N_a} log(q_i[z_i])$ where z_i is the output of the cryptographic primitive given p_i and the key candidate k. Based upon the final scores from the previous equation, the key candidates are then ranked in decreasing order and the attacker chooses the candidate ranked first in the sorted vector.

The goal during the profiling collection phase is to gather traces for as many possible key and plaintext combinations as possible to ensure the attacker has a more complete picture of the search space, and can create a model that better estimates the probability distribution function for the leakage [2].

Models used in the profiling phase can range from t-distribution to trained neural network models. As described in the equation in final step above, the most efficient manner of determining the correct key byte is by following a maximum likelihood strategy. This means the attacker generates a likelihood for each possible key byte value and chooses the value that maximizes the likelihood. This is often performed using Bayes' theorem under the assumption that individual trace acquisition events are independent [14].

2.3 Electromagnetic Theory

Electromagnetic waves are created through the acceleration of charged particles. More aptly, they are generated from the variation in electric currents over time. The value of these waves can be derived from Maxwell's equations [15]. These waves are the combination of electric and magnetic fields moving through free space. These waves are considered EM emissions. In some cases, such as radio broadcasting, WiFi, and medical imaging, these emissions occur to serve a purpose. In other cases they occur as a result of particles and electricity moving within an electronic component, such as the charging and discharging of a capacitor, or a current passing through a resistor. Modern computer technologies rely on a large number of components that depend on electric pulses and alternating currents. These pulses and alternating currents cause unintended EM waves to be generated.

There are two regions we consider with respect to EM radiation; near-field, and far-field. In the near-field region the electric and magnetic fields can be measured separately. Additionally, one field may dominate the other field based upon underlying features and geometry. In the far-field the electric and magnetic fields equalize, meaning it does not matter what type of antenna you choose to use to receive the waves. All EM waves can be collected given the right equipment. Radio transceivers enable the sending and receiving of purposeful EM waves. The transceivers are able to take information, perhaps an audio source as input, encode that input into EM waves, and send those waves out to receivers that subsequently decode the signal, perhaps reproducing the audio that was originally captured. The unintended EM waves generated by computing devices can be captured using EM probes. Based on previous research an electric field probe was chosen to collect leakage data. In this work, an EM probe captured the generated signals through digital sampling. The sampling rate of the EM probe must be at least twice the frequency of the signals

that must be captured [16]. The most common collection equipment uses EM probes attached to high sample rate oscilloscopes that digitize the EM waves for further analysis by some software. The output of the oscilloscope is proportional to the electric field strength at the tip of the probe.

2.4 Cryptography

Cryptography is the act of creating or solving codes. In computer terms this most often refers to securing information by encoding it either during transport or while at rest. The purpose of cryptography is to ensure that information is not accessed by people who do not need to have access. Encryption is the process by which a plain piece of information called a *plaintext* is converted to *ciphertext* by using an encoding scheme, often composed of an high-level encryption algorithm with one or more cryptographic primitives. A cryptographic primitive is a low-level encryption algorithm that is used to build high-level encryption protocols. These include simple one-way hashing functions, and encryption functions that perform simple swaps or shifts in data [17]. High-level encryption algorithms include Rivest-Shamir-Adleman (RSA), AES, and data encryption standard (DES).

2.4.1 Advanced Encryption Standard

The AES is a specification for a standard used for the encryption of electronic data established by the U.S. NIST in 2001 and is widely used today [18]. AES is a symmetric block cipher and uses key lengths of 128-bits, 192-bits, or 256-bits. A symmetric encryption algorithm uses the same key for encryption and decryption. A block cipher encrypts data in fixed-size blocks instead of one bit at a time like a stream cipher would. AES-128 encrypts 128-bit blocks of plaintext with a 128-bit key. Figure 1 is a diagram displaying the different operations within AES, and where

they fall in order. AES-128 performs its encryption in 10 rounds, with each round aside from the last including four cryptographic primitives: SubBytes, ShiftRows, MixColumns, and AddRoundKey. The final round does not perform the MixColumns operation. AES-128 performs each of its operations on a $4x4$ matrix of bytes called the state. The SubBytes operation is a non-linear substitution where each byte in the state is replaced with another value according to a lookup table. With AES this lookup table is called the SBox table. The ShiftRows operation is a transposition step that shifts the rows in the state 0, 1, 2, or 3 times depending upon the row number. The MixColumns step combines the four bytes in each column in a linear mixing operation. The AddRoundKey operation occurs in each round as well as once prior to the first round and combines a round key with each byte in the state using a bit-wise XOR operation. The round key is derived from the encryption key and a key schedule algorithm.

2.4.2 Hardware versus Software Encryption

Encryption can occur via software or hardware implementations. Software encryption is generally more cost effective for smaller manufacturers, however it typically has lower performance as the processing will be shared between the encryption algorithm and all other programs on the device. Hardware encryption is more cost effective for larger manufacturers, typically involving a dedicated processor. This provides more security over software encryption as the keys are not stored in main memory. Additionally, because it runs on a dedicated processor it does not need to share resources with other programs which increases overall performance [19].

Hardware encryption is generally more secure than software. Software encryption can be disabled by users or malware, however hardware encryption is on a separate device and all information on the device is always encrypted. Further, soft-

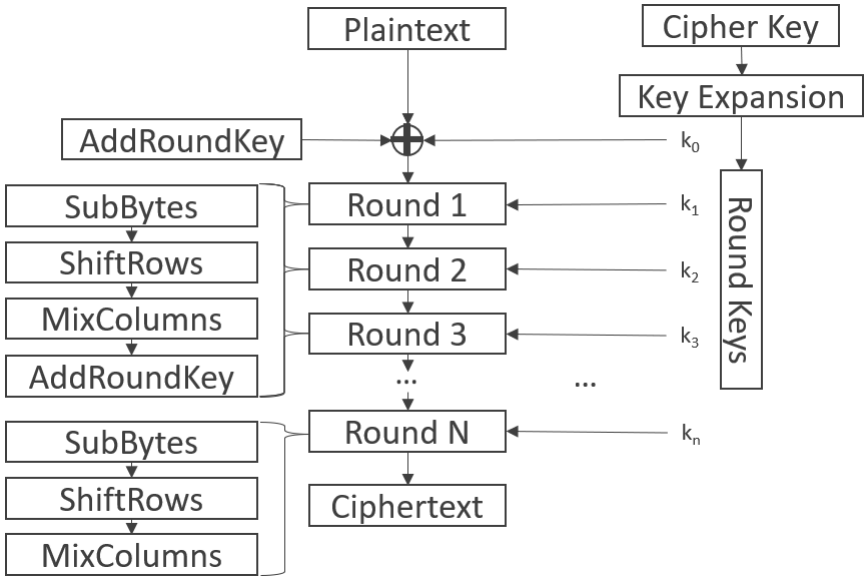

Figure 1: String Diagram displaying the rounds in AES, including the steps in each round, and the round keys used in each round.

ware encryption uses pseudo-randomly generated numbers to create asymmetric keys, whereas hardware encryption generates asymmetric keys using physical attributes of the device. Pseudo-randomly generated keys can be vulnerable to a sophisticated and dedicated enough attacker, where keys generated from some physical attribute are much more secure[1] [19].

Hardware encryption is also often implemented as part of a processors instruction set. For example, AES can be implemented using the AES instruction set on multiple architectures including the x86 architecture and the ARM architecture. One advantage of being implemented in this way is the ability of hardware encryption to process the encryption sets for all bytes simultaneously in each round. This is opposed to software encryption, which must perform encryption in a byte-wise manner during each round due to the performance trade off of software encryption. The ChipWhisperer Lite XMEGA microcontroller uses a software implementation of AES[2] and the

[1]https://oa.mo.gov/sites/default/files/CC-CryptoHardwarevsSoftwareEncryption041304ARCapp.pdf
[2]https://github.com/newaetech/chipwhisperer/blob/develop/hardware/victims/firmware/crypto/mbedtls/library/aes.c

ChipWhisperer Bergen Board setup, which uses a Xilinx FPGA, implements a hardware version of AES[3] From a side channel perspective, this makes software encryption easier to observe and attack as the information in each clock cycle pertains to an individual byte. Hardware becomes harder to attack as there is more information being processed at each clock cycle and thus the noise increases.

2.5 Machine Learning

For much of computing history and traditional programming, devices were created with a set of rules. Those devices were then fed information and the rules calculated a set of output which was returned to the user. Machine learning is a paradigm that instead gives a computing device inputs with no explicit instructions, and then asks the device to analyze and draw inferences based upon recognized patterns in the input data. There are three types of machine learning; supervised, unsupervised, and reinforcement learning. In supervised learning, inputs and outputs are given to an algorithm, and the algorithm is expected to learn rules that enable it to get from the given input to the given output. The output is usually in the form of a label, and the amount of error between the given label and the label generated by the algorithm is used to shift the algorithm to ensure it does better next time. Unsupervised learning is often used in predictive models as the algorithm is only given input data and asked to draw conclusions or to group input by patterns it finds in the data. This form of machine learning is often used to create predictive models and is used when output data does not exist. An example might be giving an algorithm information regarding weather patterns using historical examples, and then asking it to predict tomorrow's weather based upon current information and the historical examples. Reinforcement learning closely mirrors human learning in that depending upon how

[3]https://github.com/newaetech/cw310-bergen-board

close the algorithm is to being correct, it gets a positive or negative reward. Using the previous example with weather prediction, if the algorithm correctly predicts it will be sunny and in the mid 70s tomorrow, then it will be met with a positive reward. However if it predicts that warm sunny day and instead it is overcast and in the low 40s, then the algorithm receives a negative reward [20]. Figure 2 shows a comparison between traditional programming and the three forms of machine learning. This research uses supervised machine learning to perform SCA work.

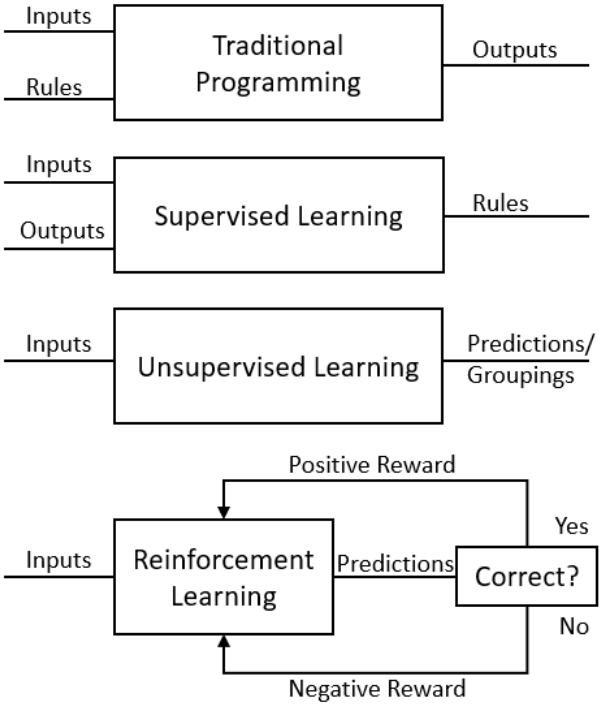

Figure 2: The differences between Traditional Programming, Supervised, Unsupervised, and Reinforcement Machine Learning

All supervised machine learning efforts require at least four essential elements: input data, expected output data, a computational tool, and a way to measure the effectiveness of the algorithm in order to update its rules. Through these rules, updates, and measures of effectiveness, the algorithm "learns". The central problem in supervised machine learning is for a computer to learn *meaningful transformation*

of data, or to learn useful *representations* of the input data that allows the algorithm to get to the expected output [1]. In this case, a representation simply refers to a manner in which data is represented, or encoded. Supervised machine learning looks to represent raw data in a way that helps to classify the input into an output. However machine learning can only search through a given space of representations, known as a *hypothesis space*.

2.5.1 Deep Learning

Deep Learning is a sub-field of machine learning that places an emphasis on successive layers of data transformations that attempt to create increasingly meaningful representations of input data. The number of layers included in the model is called the model *depth*. Deep learning can include anywhere from tens to hundreds of layers. These layers are stacked on top of one another in a model called a *neural network*. The concept of a neural networks arises from a reference to neurobiology, however they do not pretend to represent the actual inner workings of a human brain. In neural networks, all of these many layers between the input and output layers are called *hidden layers*. Each hidden layer of a neural network is made up of a matrix of neurons that take input from the previous layer, and perform nonlinear transformations on the input to create output for the next layer. In short, the output of a hidden layer becomes the input for the following layer.

Figure 3 visually demonstrates how these layers learn data representations to classify the input, a hand drawn number 7. As you can see, the representations become increasingly different from the original image, however they also become increasingly more effective at informing the output layer of the nature of the input in question. The specific transformations that occur in each layer of a deep learning model are calculated from the layer inputs, or the outputs of the previous layer, and

the *weight* stored in each neuron. The job of the neural network is to determine weights for each neuron that allow the model to turn input data into output data that accurately classifies the inputs.

In order for a deep learning algorithm to learn during training, there must be a metric that measures the error, or how far off the predicted output was from the truth value. This metric is called *loss*, and the task of calculating the loss is accomplished by including a *loss function*. A loss function takes the truth value and the output determined by the model, and calculates the loss between the two. This loss value, or *loss score* is then used by the model, in a process known as *backpropagation* to calculate the gradient of the loss function with respect to the weights of the network. Then, an *optimization function* takes the calculated gradient, learning rate, and other factors and updates the weights of the neurons in the models. At the initiation of training of a neural network, all neuron weights are initialized with random values. This means that for the first training cycle, called an *epoch*, the accuracy of the model will be equal to the probability of randomly guessing the output for each input as during this first epoch, the model has learned nothing yet and will be randomly guessing. Each successive epoch, the model will update the weights and the loss value should decrease and the accuracy should increase. Note that training accuracy does not directly translate to overall model accuracy. During training, models can be fed two sets of data; training data that is used to learn and update weights, and validation data that is not used for training and instead used to demonstrate how much the model has learned about data it has not yet seen. Models that begin to memorize the training data will see their training accuracy go up, however their validation accuracy may stagnate or even begin to decrease. Once validation accuracy hits this stagnation point or begins to decline, training should be terminated.

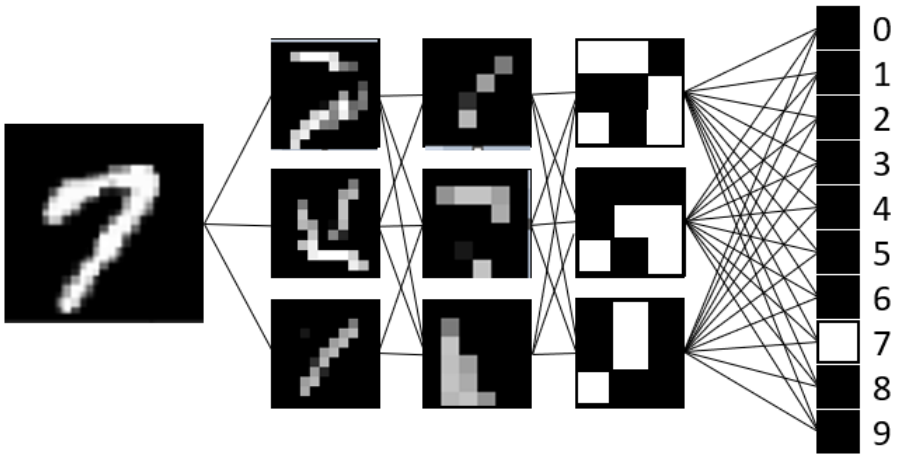

Figure 3: Data representations learned in a deep learning neural network. This figure is based on a figure found in the book Deep Learning with Python by François Chollet [1]

2.5.2 Convolutional Neural Networks

CNNs are a deep learning algorithm that include convolutional layers. Convolutional layers use a mathematical operation called a convolution that takes a larger matrix and places a moving filter, called a *kernel*, on that matrix and performs element-wise multiplication within the kernel area, summing the results and placing the sum into an element of a smaller matrix. This convolutional layer allows for sharing of weights between neurons and reduces the number of parameters in the final model. Essentially, the convolutional layer summarizes the features present in an input. CNNs also often have pooling layers that perform down-sampling. These two features, feature summarization and down-sampling, give CNNs their most important property; translation invariance. A CNN is able to learn features that appear in different locations across a given input.

CNNs also include pooling layers, non-linear activation functions, batch normalization functions, and fully connected dense layers to assist with classification. CNNs

are typically organized in blocks of layers. These blocks will often feature a convolutional layer, a non-linear function, a pooling layer, and a batch normalization function. However each block may be organized differently. Pooling layers are used to down sample the input features. Down sampling is a process that creates a lower resolution version of the given input. However despite being lower resolution, the key structural elements of the features are maintained. Pooling layers accomplish down sampling by sliding a two-dimensional filter over the feature map. This filter sliding summarizes the underlying feature map to assist with feature recognition regardless of feature size. Non-linear activation functions apply a non-linear activation function such as rectified linear unit (ReLU), Leaky ReLU, or a tangential sigmoid function. This function is applied to the entire feature map in an element-wise fashion, which means that the input and output of the non-linear layer will always have the same dimension Non-linear activation functions allow the model to learn non-linear representations of the data. The fully connected, otherwise known as *dense* layers in a CNN are always placed after the convolutional blocks and prior to the output. These dense layers are used to enable the final classification decision. In CNN models, prior to the dense layers, there is usually a flattening function that takes the multidimensional feature maps and flattens them into a one-dimensional vector that will be used by the dense layers for the final classification. The output layer of a CNN is a one-dimension vector with a size determined by the number of possible classification decisions. For the example in Figure 3, the size is ten as there are ten possible digits that can be handwritten. The value output in each of the vector's indices is the probability of the classification being the correct classification according to the trained model [21].

2.5.3 Adam Optimization Function

The Optimization function in machine learning determined how much to adjust the weights in a model based upon several factors. These factors include the error gradient calculated by the backpropagation algorithm, the learning rate, and training momentum. The purpose of these weight updates is to decrease the loss, and thus increase the accuracy of the model. Gradient descent algorithms are commonly used in neural network optimization. Gradient descent algorithms use calculus to determine the *gradient*, or the direction in one axis in which the greatest change will occur in the other axis. These algorithms continually move the weights in the direction determined by the gradient to find a local or global minima. The stochastic gradient descent (SGD) algorithm improves upon normal gradient descent by introducing a learning rate and only searching through a portion of the dataset instead of the entire dataset. Searching through only a portion of the data at each iteration results in an increase in the number of iterations to reach the minima, however the overall computational cost of SGD is lower in cases where the dataset is large.

Adam is an extension of the stochastic gradient descent optimization algorithm. Where SGD uses first, second, and other high order gradients, Adam only uses first order gradients. This reduces the memory requirement of the neural network. The Adam algorithm computes estimates of the first and second moments of the gradients and uses those estimates to perform backpropagation. Advantages of Adam are the parameter updates are invariant to the scaling of the gradient, step sizes are bounded, it works with sparse gradients, and it naturally does step size annealing [22].

2.5.4 Softmax Activation Function

When working with convolutional neural networks, the softmax function is often used as a final layer classification function. The softmax activation function converts

a vector of real numbers with length N into a probability distribution of N possible outcomes. Prior to this conversion to a probability distribution, the output vector of a neural network may include positive numbers greater than one, negative numbers, or the sum of the vector may be greater than one. After applying softmax, the resulting distribution will cause each index of the vector to fall between zero and one, and all the sum of all indices will be one. This means the output vector can be interpreted as the probability of that output being the correct output according to the model. These probabilities are easier to work with in CNN models.

2.5.5 Negative Log Likelihood Loss Function

In Convolutional Neural Network (CNN), one oft-used case involves categorical classification. This classification seeks to have a one-hot encoded vector of each of the potential classes. However, the output layer of a model does not produce a one-hot vector. Note, you can think of a one-hot encoded vector as a probability distribution where there is no uncertainty. When using the softmax activation function, the output layer of a model will instead output the probability distribution determined by the model for all possible classes. The goal in categorical classification is to maximize the likelihood of correctly categorizing the input given a model and a set of parameters. The likelihood of guessing the correct classification in this sense can be calculated by summing the unit-wise multiplication of the one-hot encoded vector corresponding to the correct output, and the probability distribution vector output by the model. Maximizing the likelihood of guessing the correct category can be accomplished by minimizing the loss function. One such loss function is negative log likelihood (NLL), which uses likelihood and the negative logarithm of small numbers to create a loss function. This can be represented in the following equation: $NLL = \sum_{i=0}^{j} k_i s_i$ where j is the number of possible classes, k is the one-hot encoded vector,

and s is the probability distribution output by the model. This summation function can be reduced to taking the probability output by the model for the correct class. This results in a small number, and the logarithm of a small number, such as the probabilities of each category produced by a model that all need to sum to 1, will be negative, so taking the negative will produce a positive number where the higher the resulting number, the lower the probability produced by the model for that specific category. Herein lies the need to minimize. The NLL can be calculated by taking the negative natural log of the likelihood. The lower the result, the closer the model is to correctly classifying the input data.

In neural network training, minimizing NLL loss has been demonstrated as being asymptotically equivalent to maximizing the perceived information (PI) [2]. The PI serves as a lower bound for the Mutual Information (MI). The MI is the true statistical distribution model of the leakage of the DUT, and is computationally intractable. Using NLL loss enables neural network models to efficiently and effectively estimate the PI, which then allows said models to closely approximate the MI [23]. As NLL enables us to closely approximate the true leakage distribution of a target device, NLL may be preferable to Categorical Crossentropy, a loss function commonly used in image classification problems, in SCA work.

2.6 Field Programmable Gate Arrays

Field programmable gate array (FPGA)s are silicon semiconductor devices created to be configurable by the customer or other end user after the manufacturing process is complete. FPGAs are often configured using hardware design language (HDL) and the chip is able to be reconfigured as it contains an array of programmable logic units and re-configurable connections, allowing the units to be wired together as desired. The configurable nature of an FPGA may serve as a passive SCA countermeasure,

if it can be demonstrated that modifying the layout of the components responsible for performing encryption algorithms causes side-channel leakage to become less consistent when attacked using a model trained on data pulled from a device with a different configuration. The target of this research is an FPGA device, however no effort was made to obfuscate the encryption algorithm using alternate bitstreams. This can become a direction for future work.

2.7 Previous Research

Much of recent SCA research focuses on retrieving the correct key or key byte in as few traces on the DUT as possible with some papers demonstrating successful attacks using as little as one trace [7][5]. Deep learning techniques have been demonstrated as quite effective when used in SCA work [8]. They showed deep learning to be more successful in detecting leakage over more traditional statistical analysis attacks such as the t-test and the χ^2-test. Specifically they found deep learning outperforms such statistical analysis attacks when the leakage is non-trivial, that is the leakage data is misaligned or the leakage model exhibits a multivariate distribution [8].

Additional research considered the effectiveness of multi-device profiling attacks, in which the profiling phase consists of training a neural network on leakage traces captured from multiple devices. One paper demonstrated the effectiveness of a neural network model trained on power data traces captured both from multiple homogeneous devices and multiple non-homogeneous devices [7]. With the same number of traces captured either all from one board, or split equally across nine homogeneous boards, they demonstrated an increase from 39.95% to 86.07% success rate when executing an attack with only one trace. They discussed the accuracy drop that occurs when attacks occur against a device that is different than the profiled device and demonstrated the effectiveness of training a model on data pulled from multiple

non-homogeneous devices. They showed that attack accuracy can be improved from under 14% to over 55% in such cases [7]. Their research is based upon the idea that increasing signal and noise diversity can help lead to higher classification accuracy as the neural network has been given more diverse data to learn information from. If a network is trained on traces collected from only one device, the resulting model will be largely unable to correctly classify trace data collected from another dissimilar device. However if the model is trained on data pulled from multiple devices, it will have been exposed to more of the total sample population and have a better change of correctly classifying data it has not seen before. This diversity principle is also demonstrated using EM trace data [6]. This research trained a Deep Neural Network (DNN) on EM traces pulled from 10 homogeneous microcontroller devices and achieved an average success rate across 10 attack devices of 91.5% [6]. Both papers demonstrate the effectiveness of diversity in attack success rate.

Other investigators have sought to identify an ideal location to collect EM leakage data to lower the number of required traces and ensure a higher likelihood of attack success. Schlösser et al. demonstrated a technique to identify the SRAM location accessed during the activation of the SBox lookup operation of the AES algorithm. They analyzed photonic emissions collected using a silicon-based charged coupled device during the SBox lookup operation and successfully located the physical location of SRAM containing the SBox lookup table. They were able to do so because the spatial resolution of the emissions collected allowed them to clearly identify memory access locations even when they are adjacent [24].

Andrikos, et al. extended this idea of identifying memory access locations, using an information-based approach to retrieve the AES key-byte after performing sequential accesses to a continuous region of SRAM by loading data from all memory positions. This was accomplished by accessing 4096 32-bit words in order to load the entire

16KB region of SRAM. They divided the resulting traces into two classes, those collected using the first 2048 words and those collected using the last 2048 words. They then performed a Welch's t-test and found the location on the chip that had the most significant location-based leakage. This allowed them to identify the SRAM location of the SBox lookup table and determine the secret key [25]. Das & Sen used the signal-to-noise ratio (SNR) or test vector leakage assessment (TVLA) methods, such as the Welch's t-test or Pearson's χ^2-test, to identify a possible ideal leakage collection location from which to perform an SCA attack. They used a framework called SCNIFFER [26] in conjunction with either the TVLA or the SNR to perform a greedy gradient-search heuristic that converges on an ideal point to collect EM leakage traces. This framework is intended to reduce the correlational EM leakage analysis time by a factor of n where the chip dimension is $n \times n$ [4].

2.8 Conclusion

This chapter introduced key concepts this research is built on. It discussed leakage assessment through statistical analysis, profiling SCA attacks and how they work, the EM theory that makes EM-based SCA attacks possible, cryptography, and the AES algorithm attacked in this research, machine learning, deep learning, and key deep learning concepts such as CNNs, the Adam optimization function, the Softmax activation function, and NLL as a loss function. These specific functions and terms represent the functions used in this research. Finally, the chapter presented research conducted by several different authors with the main goal being the advancement of SCA techniques. This includes the introduction of CNNs for leakage assessment, using heterogeneity of devices when collecting leakage signals, and a system for finding an ideal location from which to collect EM leakage to conduct SCA attacks.

III. Methodology

This chapter describes the research methodology used to perform electromagnetic (EM) Side-Channel Analysis (SCA) using Convolutional Neural Network (CNN)s to identify ideal points from which to conduct SCA attacks. The chapter is laid out as follows: Section 3.1 discusses the collection methodology, Section 3.2 discusses the architecture of the CNNs, Section 3.4 discusses methodology used to identify the ideal points. Finally, Section 3.5 presents the attack methodology used for determining the success rate for the models trained at each of the identified points.

The research consists of three phases. The first phase is a pilot study to determine whether a CNN can identify points that enable an attacker to extract a key byte more quickly. This study uses the ChipWhisperer-Lite platform and an XMEGA microcontroller, designed to leak information as a teaching tool for those interested in SCA work [1], and commonly used in side-channel research [5, 7]. Once results demonstrated the effectiveness of this methodology, additional research is accomplished to determine if the experiment can be accomplished with less data, thereby reducing the time required to train CNN models. The final experiments involve utilizing the demonstrated framework's effectiveness on a more secure device. The device chosen is a Xilinx Kintex-7 410T field programmable gate array (FPGA), fitted on the ChipWhisperer 310 platform[2].

Python is used with multiple specialized libraries throughout this work mainly for its compatibility with the TensorFlow and Keras libraries which perform much of the underlying work for the CNNs[3]. Additionally, python is the language used by the United States Air Force (USAF) in their collection tool named *WarChest*. WarCh-

[1] https://www.newae.com/chipwhisperer
[2] https://www.xilinx.com/products/silicon-devices/fpga/kintex-7.html
[3] https://www.python.org/

est uses several python libraries including ChipWhisperer[4], python-ivi[5], h5py[6], and PySerial[7]. These libraries enable WarChest to send instructions to each of the components in the collection process. For the collection, we use an three axis linear translation stage connected to a Duet WiFi translation stage controller, an oscilloscope, a ChipWhisperer target, and a local data-store[8]. WarChest connects to the Duet WiFi via serial connection, and sends commands via a serial bus. WarChest generates the keys and plaintexts used in the cryptographic operations and passes them to the ChipWhisperer target using a separate serial connection. WarChest also communicates via Ethernet to the oscilloscope and sets the collection parameters using the python-ivi library to ensure collection is consistent across all collection points and collection runs. The oscilloscope connects via a coaxial cable to the EM probe. When the ChipWhisperer board is ready to perform the cryptographic operations, it sends a trigger pulse to the oscilloscope to begin the collection and performs the operation. The captured trace is then passed back via the Ethernet connection to the WarChest platform which then saves them on the local data storage.

The research uses profiled SCA attacks, as mentioned in Section 2.2. As a brief refresher, this form of SCA attack uses two phases, a profiling phase and an attack phase. The profiling phase consists of an adversary first collecting leakage data from a device equivalent to the target device, but which the adversary has full control over, during operations of interest, such as encryption operations. Following which, the adversary trains a model on the collected leakage data. For the attack phase, the adversary collects leakage data on a target device over which they do not have full control. The adversary collects this data by observing the device while it performs an

[4]https://www.newae.com/chipwhisperer
[5]https://github.com/python-ivi/python-ivi
[6]https://www.h5py.org/
[7]https://pyserial.readthedocs.io/en/latest/
[8]https://www.duet3d.com/DuetWifi

operation. In many cases this operation is an encryption algorithm. During the attack phase the secret information is assumed to be fixed. The attack data is fed into the previously trained model and an attempt is made to extract the secret information. In many cases, to include this case, the secret information is a cipher key.

3.1 Side-Channel Leakage Collection Methodology

This section presents6 the collection processes used during each of the collection phases of the research. Section 3.1.1 discusses the setup used during the preliminary research, conducted using the ChipWhisperer-Lite and an XMEGA ARM microprocessor using a software implementation of Advanced Encryption Standard (AES). Section 3.1.2 details the collection setup used for the main body of research, using the ChipWhisperer 310 platform and a Xilinx FPGA performing hardware-based AES encryption. A full diagram for the collection setup, used for both stages of research, is displayed in Figure 4.

3.1.1 ChipWhisperer XMEGA Collection Setup

A collection setup for initial research into the effectiveness of CNNs for identifying ideal leakage collection locations consists of an three-axis translation station, a Riscure Version 4 EM probe connected to a LeCroy WavePro 725Zi oscilloscope with a maximum sampling rate of 40 giga-samples per second (GS/s), set above a ChipWhisperer-Lite ARM board attached to a 16-bit Atmel XMEGA D4 microcontroller utilizing a software implementation of the AES algorithm [9]. The EM probe is configured to move in a grid pattern, demonstrated in Figure 5, above the XMEGA microcontroller. The XMEGA microcontroller is 10mm $\times$ 10mm in size and the pattern consisted of 25 collection points arranged in an 5×5 grid with each point set

[9]https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8135-8-and-16-bit-AVR-microcontroller-ATxmega16D4-32D4-64D4-128D4_datasheet.pdf

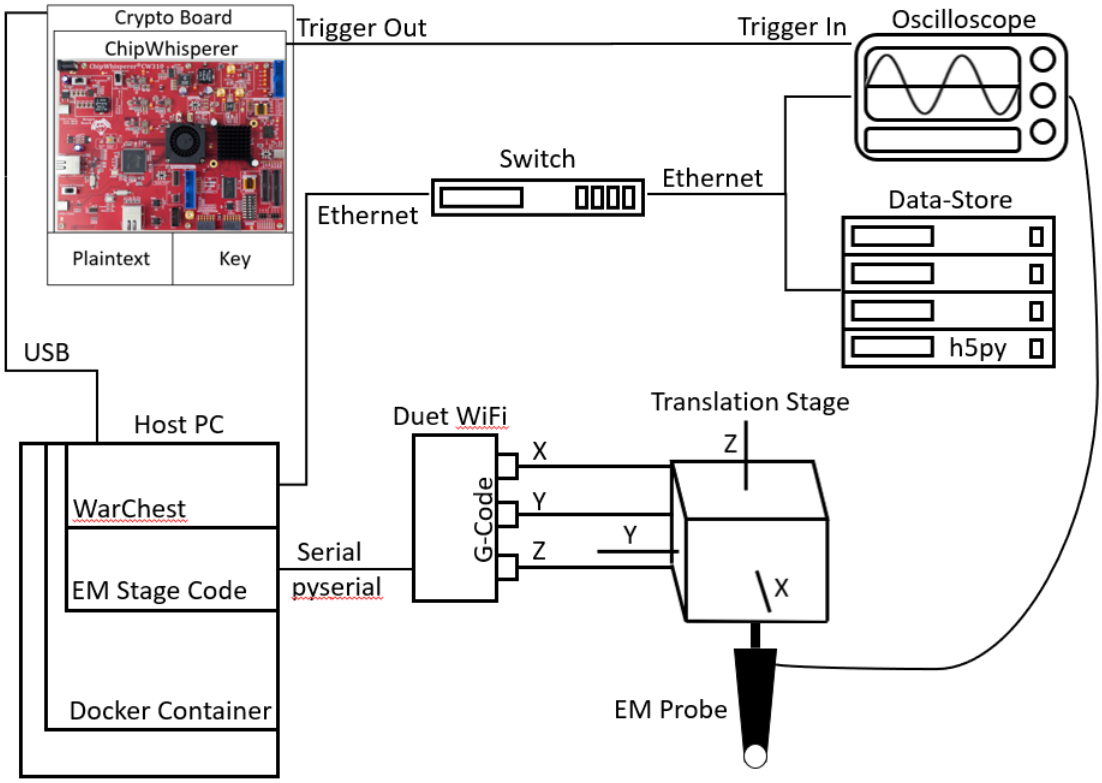

Figure 4: Full collection setup diagram displaying the connection types between different components in the setup and the protocols and libraries used to control the system.

2.5mm apart. The probe moved in a raster scan manner, moving across five collection points before moving to the next row and collecting at the next five points in the opposite direction. At each point, the EM probe collected 33,000 traces, each with 500,000 samples. The first 30,000 traces are allocated for training and testing a profile model, and 3,000 are reserved for the attack. The training and testing traces are collected with randomized plaintext and randomized keys. This is done to ensure the trained models are exposed to a larger set of possibilities within the search space. The 3,000 attack traces are collected with randomized plaintext and constant key. The attack traces use a constant key as in a real world SCA attack the key would change according to a key schedule. This key schedule makes it imperative that an attacker be able to retrieve the correct key byte in as few traces as possible.

To collect leakage data, the microcontroller must execute the AES encryption algorithm and use a trigger timer to synchronize the oscilloscope and the microcontoller to capture the leakage data during the course of the encryption operation. This entire process is performed using the WarChest tool developed by the USAF. WarChest sets the plaintext and key metadata, loads the firmware onto the device, then issues a command to the microcontroller to trigger the key update and encryption operations, all using a *.hex* file created using instructions found on the ChipWhisperer website[10]. Then WarChest sets the trigger for both the microcontroller and the oscilloscope. The traces collected are 5ms in length, and the oscilloscope collected samples at a rate of 100mega-samples per second (MS/s). This results in 500,000 samples per trace. These 500,000 samples included 150,000 samples prior to the beginning of the AES encryption process, and 130,000 traces following the completion of the encryption process. This leaves 220,000 samples related to the AES process. However, due to the rounds used in the AES encryption algorithm, the region between samples

[10]https://wiki.newae.com/index.php?title=V4:Tutorial_B1_Building_a_SimpleSerial_Project

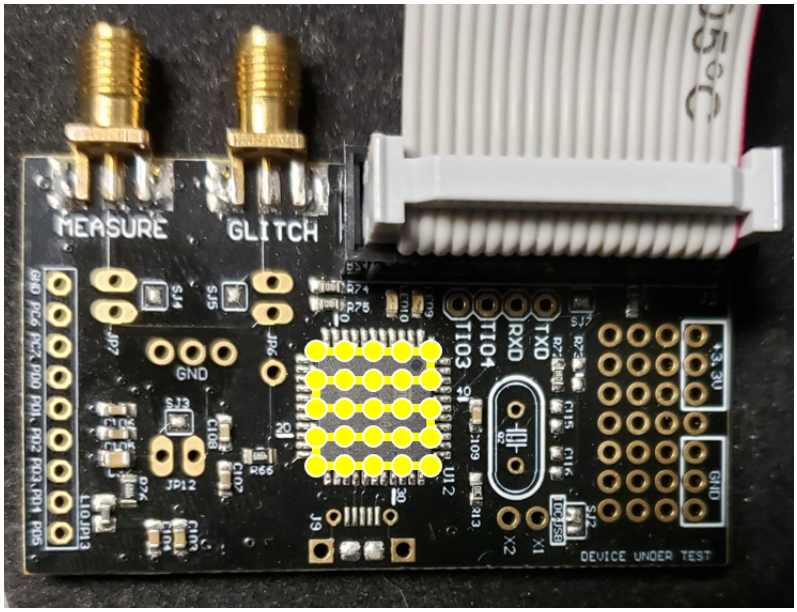

Figure 5: Atmel XMEGA D4 microcontroller displaying 25 grid points used for leakage collection. The lines between points demonstrate the raster scan movement of the collection probe

150,000 and 180,000 are identifiable as the time period in which the first round of the algorithm occurred. The region corresponding to the time the algorithm begins and the end of the first round is chosen for the attack point, as shown in Figure 6. This attack point is chosen because the time between the initiation of the AES algorithm and then completion of the first round is the only time in which the original key is exposed, as the key undergoes a key expansion heuristic during each round of the algorithm. This isolation ensured that only 30,000 samples per trace are needed to retrieve the key byte, and can be seen in Figure 8 by the red bounding box displaying the first round.

The collected traces are separated out into three groups. The first group contained 30,000 traces where both the plaintexts and keys are randomly chosen. This group is then split into groups of size 27,000 traces and 3,000 traces. The first 27,000 are used for model training, and the last 3,000 are used for individual point testing, once a model is trained. The 27,000 training traces collected at each of the 25 points

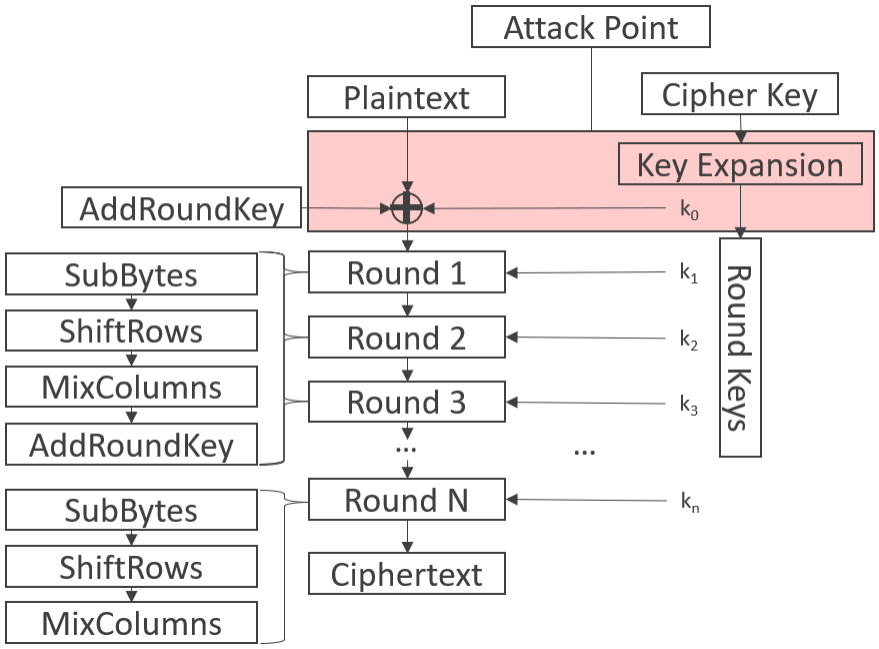

Figure 6: String Diagram displaying the chosen attack point for the AES algorithm. This attack point is chosen due to it being the only period where the cipher key is exposed in the clear before being modified by the Key Expansion algorithm.

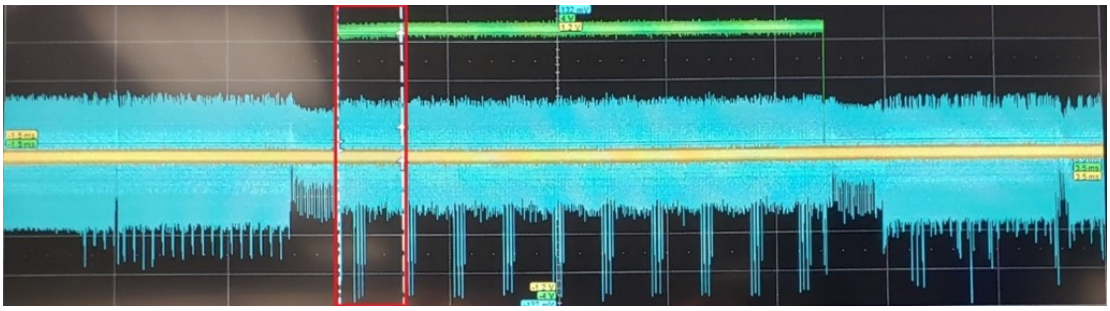

Figure 7: Oscilloscope output for the Atmel XMEGA microcontroller demonstrating the location of the isolated samples. Notice the 10 spikes corresponding to the 10 rounds of encryption.

are combined into a single large training set consisting of 675,000 traces. This large training set is provided to the neural networks to train the models. The 3,000 points are then used for testing to determine the accuracy of the model at each of the individual points.

3.1.2 Xilinx FPGA Collection Setup

The collection setup consists of the three-axis linear translation station, a Riscure Version 4 EM probe connected to a LeCroy WavePro 760Zi-A oscilloscope with a maximum sampling rate of 40 GS/s, set above a ChipWhisperer CW310 fitted with a Xilinx Kintex-7 410T FPGA. The EM probe is programmed to move in a grid pattern, shown in Figure 8 above the FPGA. The pattern consists of 80 collection points set up in an 8×10 grid with each point set 2mm apart. The probe is moved across the shorter axis of the board from point to point until it reached the edge of the chip. Once it reached the edge of the chip, the translation table moved the probe back to the initial edge. Then it moved one point along the longer axis and began working across the shorter axis once more. It continued this E-shaped scanning pattern in order to ensure each collection point is reached from the same direction. The intermediary research discussed in Section 3.8 demonstrated that it is possible to train the CNN models with fewer traces. Thus, at each point above the Xilinx Kintex-7 FPGA, the EM probe collected 23,000 traces split into 20,000 samples with randomized plaintexts and keys, and 3,000 with randomized plaintexts and a single constant key. Each collected leakage trace has 20,000 samples. However, the number of testing and attack traces between both phases of the experiment is held constant.

To collect leakage data, the FPGA is observed during the AES encryption algorithm and with a trigger signalling the oscilloscope to capture the leakage data during the course of the encryption operation. This entire process is performed using the

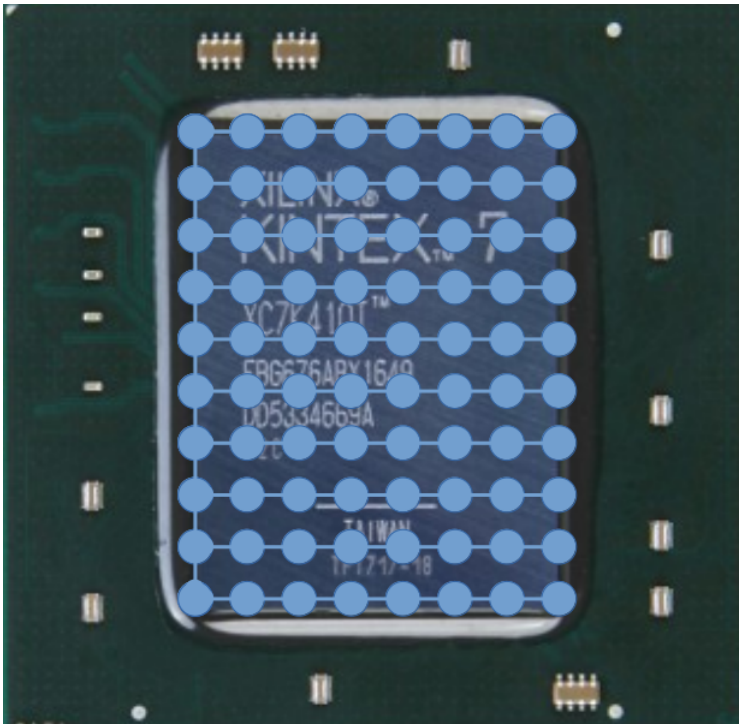

Figure 8: Xilinx Kintex-7 410T FPGA device with 80 leakage collection points. The lines between points demonstrate the pattern of collection

WarChest tool developed by the USAF. WarChest sets the plaintext and key metadata, directs the microcontroller to perform the encryption operation based upon a bitstream generated by the code shown in Appendix A, and sets the trigger for both the operation and the oscilloscope. The oscilloscope captured a window of 1,000ns, however data is collected during the 50ns before and the 75ns following the AES encryption process. The oscilloscope captured samples at a rate of 20GS/s which resulted in a total of 20,000 samples per trace. As with the XMEGA experiment, the focus is on the window of time between the beginning of the algorithm and the end of the first round as it is the only window containing the original key. Figure 9 displays this region of interest, which corresponds to the samples between the 1,000th and the 3,000th samples. This results in 2,000 samples per trace required to successfully retrieve the key.

As with the preliminary research, the collected traces are separated out into three

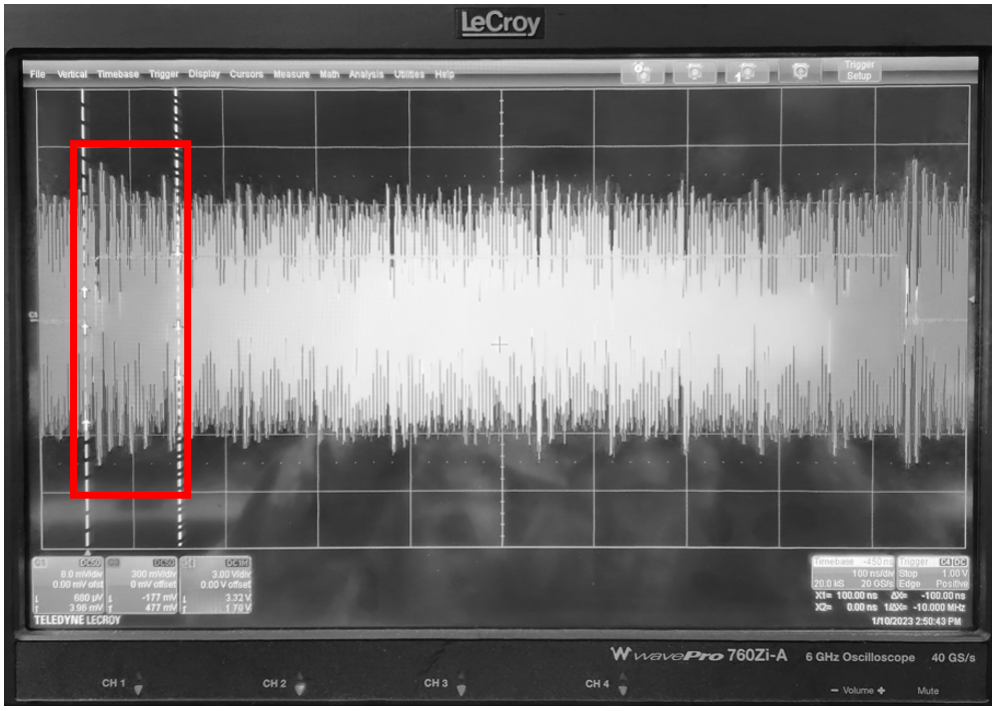

Figure 9: Oscilloscope output for the Xilinx Kintex-7 FPGA demonstrating the location of the isolated samples.

groups. The 20,000 traces with randomized plaintexts and keys are split into groups of size 17,000 traces and 3,000 traces. The first 17,000 are used for model training, and the last 3,000 are used for individual point testing, once a model had been trained. The 17,000 training traces collected at each of the 80 points are combined into a single large training set consisting of 1,360,000 traces. The 3,000 points used for testing enable determination of the accuracy of the trained model at each of the individual collection points.

3.2 Convolutional Neural Network Architecture

This section will discuss the architectures used for both major phases of the research. The neural network architecture is based on the architecture used in the research published by Prouff, Strullu, Benadjila, Cagli, and Dumas (2018) [14]. This model has been proven successful when used in EM SCA research. The model is

scaled down for these attacks as the input layer size causes a larger parameter count which initially led to overfitting. The model architectures used against the different targets differ slightly due to changes in device hardware, newly discovered research, and size of the input.

The neural network architecture consisted of five convolutional blocks of increasing filter size. Each block consisted of a convolutional layer build using Tensorflow's built-in *Conv1D()*[11] with a kernel size of 11, followed by a batch normalization function, an activation function, and a one-dimension average pooling layer built with Tensorflow's built-in *AveragePooling1D*[12] function with a pool size of 2 and a stride of 2. Following these convolutional blocks, the output of the last convolutional block is flattened to create a one-dimensional vector and then sent into two dense layers used to assist final classification. As the goal is to retrieve an individual key byte with a value ranging from 0 to 255, softmax is employed as a final layer activation function to output the probabilities of each key byte candidate. The labels are converted from integer form to a categorical one-hot encoded vector of length 256 used for comparison.

In the initial research on the XMEGA target, *ReLU* is used as the activation function as it is a commonly used activation function for neural networks used in SCA research. *Categorical Crossentropy* is employed as the loss function as it is also commonly used in SCA research, and we had not yet discovered the research pointing to negative log likelihood (NLL) as the better loss function [2]. *Adam* is chosen as the model optimizer and *softmax* as the activation function for the final dense layers.

Upon pivoting to research on attacking the Xilinx FPGA, the network architecture changed slightly. The five convolutional block architecture of increasing size is retained with most of the same parameters, kernel size of 11, pool size of 2 and stride of 2, however a leaky ReLU activation function is implemented at each layer. The

[11]https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D
[12]https://www.tensorflow.org/api_docs/python/tf/keras/layers/AveragePooling1D

built-in Tensorflow LeakyReLU activation function[13] is again used, as opposed to the normal ReLU activation function, to allow negative values to impact training, instead of removing them entirely as the normal ReLU function does. It would have been desirable to study the effects of using the LeakyReLU activation function versus the normal ReLU function, however there was not enough time to test this hypothesis. Following the convolutional blocks, the output data is flattened and sent as input into two dense layers consisting of 512 neurons each. The filter sizes for this phase of research are increased due to the decrease in the input layer size. The size of the Dense layer is also increased, again this is due to the decrease in size of the input layer which decreases the overall parameter count and enables a larger network without the fear of overfitting. The output layer is the same as the output layer from the previous research. The output needed to consist of a one-hot encoded vector where the hot value represented the expected label value. Figures 10 and 11 display the architectures with the differences noted.

[13]https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU

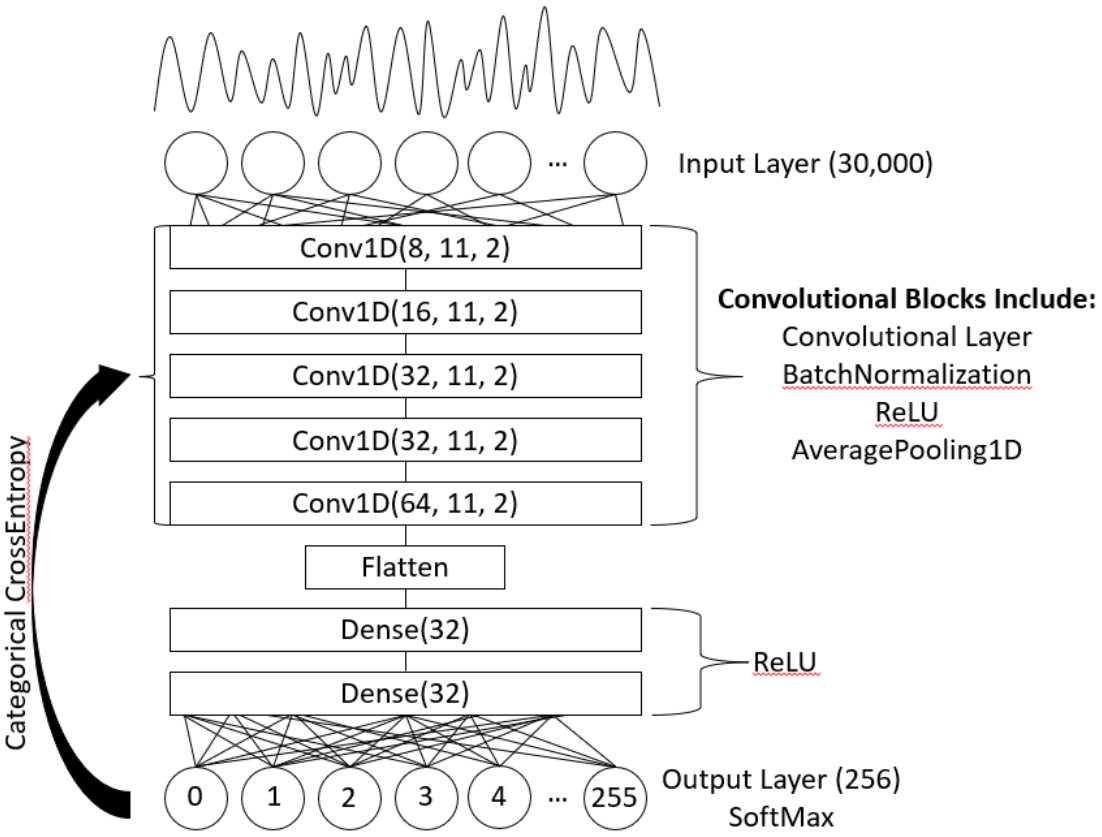

Figure 10: CNN Architecture used on the XMEGA microcontroller dataset. The network consists of five convolutional blocks, each with a convolutional layer, a batch normalization function, an activation function, and an average pooling function. The output of the final block is flattened and then used as input to two dense classification layers and the final layer is a one-hot encoded vector of length 256, corresponding to the 256 possible values of an individual key byte. Categorical Crossentropy is used as the loss function and Softmax is used as a final layer activation function.

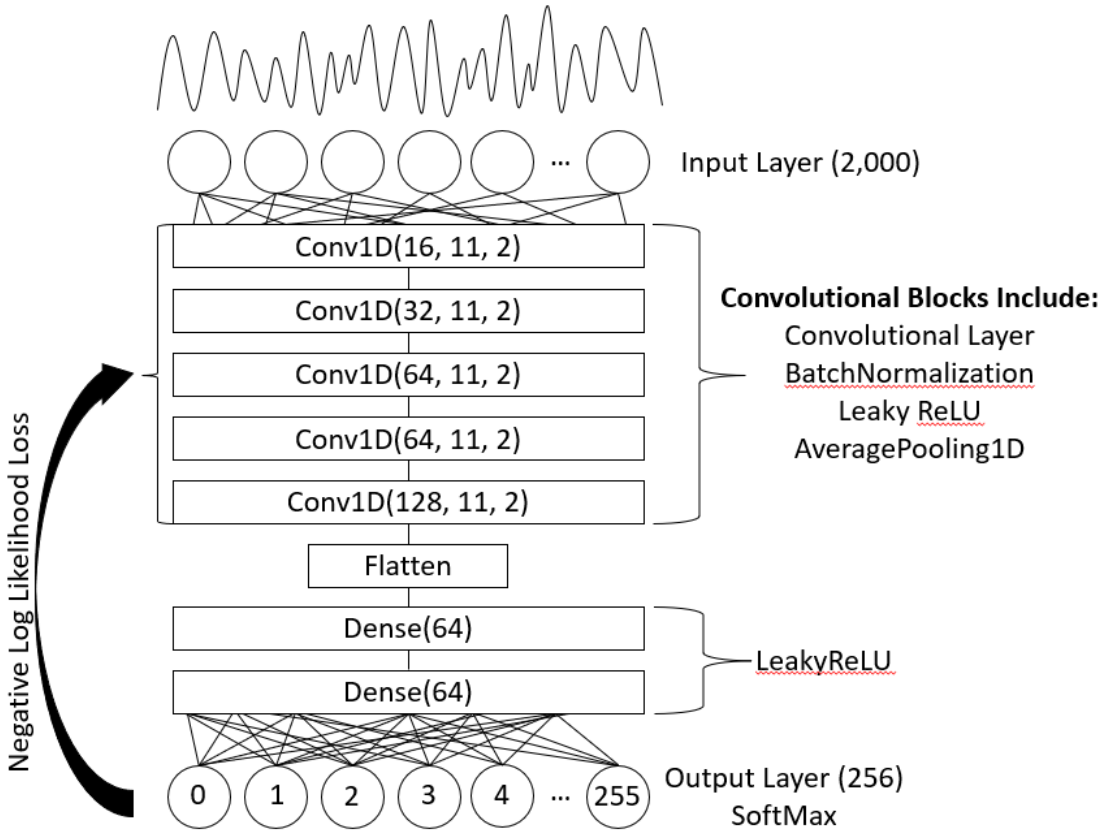

Figure 11: CNN Architecture used on the Xilinx FPGA dataset. The network consists of five convolutional blocks, each with a convolutional layer, a batch normalization function, an activation function, and an average pooling function. The output of the final block is flattened and then used as input to two dense classification layers and the final layer is a one-hot encoded vector of length 256, corresponding to the 256 possible values of an individual key byte. Negative Log Likelihood is used as the loss function and Softmax is used as a final layer activation function.

3.3 Convolutional Neural Network Training

To train the models, Tensorflow is chosen for its neural network training flexibility, and its ability to use the Keras Functional API. It performs all training within a Docker container. To build and train the CNNs used for the research, a graphics processing unit (GPU)-cluster device is fitted with four GPUs. While training models on the data from the XMEGA microcontroller and performing attacks on the target, the cluster is set up with four TITAN V GPUs[14]. Training is performed using the built-in Tensorflow *Mirrored Strategy* which enables the use of all four GPUs at once to train a single model[15]. This enables much quicker training for the models, however due to the input size and the scale of the models, even with all of the processing power from these GPUs, training took roughly four and a half hours per model. This is one of the driving factors in the number of models used in the statistical analysis during the initial research on the XMEGA target. We ran out of time to train more networks. During the second phase of the research, attacking on the Xilinx Kintex-7 FPGA, the GPU cluster is upgraded to have four Quadro A5000 GPUs[16]. Unlike with the data collected from the XMEGA microcontroller, a mirrored strategy is not employed. Instead training occurred with one model per GPU, which enabled training of up to four models at once.

After training a few models initially, it was observed that the heatmaps did not always reveal a similar pattern. One hypothesis is that this may be a result of the stochastic initialization of weights and order of input data when using CNNs. A number of models are then trained in order to better identify the ideal leakage location using statistical analysis of the performance of the many models. While training models on the XMEGA microcontroller data, 10 models are trained as a

[14]https://www.nvidia.com/en-us/titan/titan-v/
[15]https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy
[16]https://www.nvidia.com/en-us/design-visualization/rtx-a5000/

result of time constraints. Training on data captured from the Xilinx FPGA includes 40 models to get a better idea of the statistical distribution of the leakage at each collection point.

During training on the XMEGA dataset, a batch size of 200 is used, and during the training on the FPGA dataset, a batch size of 100 is used. The batch size is lowered in the second phase of training because training is much faster due to the increase in computing power from the newer GPUs and the decrease in model size. Smaller batch sizes help to increase the ability for a model to generalize about the input data. To build the batches, rather than relying on the built-in ability of Tensorflow to randomly assign traces to batches, a custom generator is created, which normalizes the input data and turns the integer labels into a categorical one-hot encoded vector using the *to_categorical* function within the utilities library of Keras.

Training on the XMEGA dataset included a learning rate of 0.00001 over 50 epochs. Training on the Xilinx FPGA target used a learning rate of 0.000003 over 100 epochs.

3.4 Model Testing and Heatmap Analysis

The 40 trained models are then tested against the 3,000 testing traces gathered at each of the leakage collection points. The tests are accomplished by loading the trained models into Tensorflow using a generator similar to the one described in Section 3.2, the data is loaded, the integer labels are transformed into a one-hot encoded vector, and the signal data is normalized, after which model.evaluate() is executed using the generator. The results for each test point for a given model are saved in an h5 file the entire test results are saved in an array format according to the number of testing points in the grid, e.g. 5×5 for the XMEGA tests and 8×10 for the Xilinx tests.

The heatmap analysis for both research phases is relatively similar. The main difference being the number of models used for the analysis. The heatmaps are generated first by running the 3,000 testing traces, mentioned in Sections 3.1.1 and 3.1.2 through the trained models using the tensorflow *model.evaluate()* function. Once the accuracies are returned, the 10 and 40 accuracies for each of the collected points for the XMEGA and Xilinx FPGA respectively are averaged and plotted using python's plotly Version 2.17 [17]. This provides a visual representation to identify and areas that present with higher averaged accuracies.

The ChipWhisperer XMEGA had only 10 trained models, weakening the statistical analysis as compared with the testing on the Xilinx FPGA, which used 40 trained models. Unfortunately, the time required to train a model with an input size of 30,000 is too long and as a result only 10 models are trained. Despite this, two points were discovered to be statistically significant from one another when discussing test performance. Further the intent of using multiple models, averaging the accuracies, and using statistical significance, is to remove any noise that can cause issues with classification and reduce the variance that may be created as a result of the stochastic nature of neural networks.

Heatmaps can be created by executing each of the trained models against each of the 25 and 80 testing trace datasets collected on the XMEGA microcontroller and the Xilinx FPGA respectively. Accuracy metrics for each specific pairing could then be obtained by processing each of the traces in the model and calculating the accuracy for a given model and a given dataset.

Confidence intervals are calculated to determine if the average accuracies returned for all points from all trained models are distinguishable from one another. This may allow rejection of the null hypothesis in which the location of EM collection is not

[17] https://github.com/plotly/plotly.py

important, indicating that all points leak similarly. To determine if the resulst of data collected from one point is statistically significant from another point of collection, confidence intervals are calculated on the returned accuracies for each point over the trained models, using the t-distribution table and a confidence level of 95%[18]. If the entirety of the two-tail confidence intervals for a given confidence level do not overlap, then the two datasets are considered statistically significant from one another which provides enough information to either accept or reject the null hypothesis. Confidence intervals are calculated according the following equation, where CI is the confidence interval, $\bar{x}$ is the sample mean, n is the number of samples $t_{\alpha/2}$ is the confidence level value, provided by the t-distribution table by using the degrees of freedom (n-*1*) and the confidence value (in this case, 95%), and s is the standard deviation of the sample population:

$$CI = \bar{x} \pm t_{\alpha/2} s/\sqrt{n} \tag{9}$$

The t-test statistic is used to determine whether the null hypothesis can be rejected in favor of the alternate hypothesis. The t-test uses a confidence value, $t_{1-\alpha,n}$ retrieved by using the table referenced above to identify a confidence threshold. Above this threshold the t-value can be calculated for the desired point using the following equation, where $\bar{x}$ is the population mean, μ_0 is the sample mean, S is the population standard deviation, and n is the degrees of freedom, or the number of observations minus 1:

$$t_0 = \frac{\bar{x} - \mu_0}{\frac{S}{\sqrt{n}}} \tag{10}$$

The null hypothesis can be rejected if $t_0 > t_{1-\alpha,n}$.

[18]Table can be found in Design and Analysis of Experiments by Douglas C. Montgomery

3.5 Side Channel Attack Methodology

The attack methodology between the two experiments is very similar. Two new neural networks are trained, one at the identified "stronger leakage" point and one at the point identified as being statistically significant from the "stronger leakage" point. The same architecture is used to identify the points, with an increase in the number of epochs by a factor of two. Following training of these two networks, the attacks are executed. However during the research attacking the FPGA, the point identified with its average accuracy being the median of all average accuracies also has a model trained on its leakage data and is also attacked.

3.6 XMEGA SCA Attack

The attack began with training two models, one with the 30,000 collected traces from the point identified as having statistically better leakage, and one with the 30,000 traces collected from the point identified as having statistically worse leakage. As mentioned in Section 3.5 the epoch count is doubled while training in order to increase the total accuracy of the model. This results in an epoch count of 100.

Execution of the attack includes sending the 3,000 attack traces, all with the same key, through the two point-wise trained models in a randomized order. The order of the traces is randomized in each attack to prevent a situation where attacks always succeed or fail based upon the predetermined order of the traces used in the attack. Each trace is then run through the trained model using Tensorflow's *model.predict()* function. The output of this function is a list of probabilities corresponding to the likelihood of different key byte values being the true key byte.

Once the model has output a list of the probabilities of each of the key byte values, the probabilities are used in Bayesian analysis until the correct key byte is determined to be the most probable result. Bayesian analysis is a process by which the predicted

output of a system is updated as more information is presented to the system. In this case, this increase of information is the addition of more leakage traces. The Bayesian process only works if the individual leakage events, traces, are independent.

Once the predictions are created, the traces are randomized and split into 10 groups of 300 traces, enabling 10 attacks, as opposed to one. Cursory analysis determined that if the model did not converge on the correct key byte after 300 traces, it would not converge at all. In each of these attacks, an individual traces is used to determine the ranking of each of the key byte possibilities. In order to perform the Bayesian part of the analysis, the logarithm of each of the probabilities is taken. If the probability of a key byte candidate is 0, then the logarithm of a number near zero ($1e^{-76}$) is substituted instead, as the logarithm of 0 is undefined. The log of the probabilities is then added to the summation of the logged probabilities of all previous traces. The array of key byte candidates is sorted and a result is returned. If at the end of the 300 trace attack, the predicted key byte is the correct key byte, then the attack is considered successful.

3.7 Xilinx FPGA SCA Attack

The attacks against the Xilinx FPGA are similar to the attack against the Atmel XMEGA. Two models are trained on leakage data collected from the points identified as having statistically better and statistically worse leakage. However, unlike with the Atmel XMEGA, the number of traces used at each of the points is deliberately increased to 65,000, and an additional model is trained on leakage data collected from the point with it's mean leakage being in the median of all eighty points. With 65,000 traces 99% of the search space is observable for a single key byte and plaintext byte combination. As the values for each byte can be from 0 to 256, that yields a search space of 256×256 or $65,536$. This increase could have been chosen as 2^16 (65,536),

but was not, just to keep it as a round number that would more easily fit into batches. Once again, as mentioned in Section 3.5 the number of epochs used in training the large point identification model is doubled. In other words, 200 epochs are used to train these three models to achieve higher accuracies and increase the effectiveness of the attacks.

With three trained models, all three models undergo the same attack process. The process begins with randomly separating the 3,000 attack traces into 10 sets of 300 traces. This is to avoid a situation where the order of the traces being fed into the model affects the likelihood of success. The 300 attack traces, again generated with random plaintext and constant key, are presented to the trained models using the Tensorflow library's model.predict() function. This function outputs an array of size 256, representing the predicted probability of each index value being the true key byte according to the model and the given input trace.

Bayesian analysis is then performed on these probability arrays, leading to maximum likelihood guessing for each key byte value, given more information over time. Each time a trace is provided for analysis, the logarithm of the probabilities of each of the key byte values is added to the summed logarithms of the key byte candidate from previous traces. These summed log probabilities are then sorted to provide the most likely key byte candidate. Note, if the probability of a key byte candidate is zero, the logarithm of a number very close to zero ($1e^{-76}$) is taken instead, thereby preventing an error. The sorted array is then returned. If, after all 300 attack traces are fed through the Bayesian process, the correct key byte is returned as the most likely key byte candidate, then the attack is considered a success.

3.8 Intermediate Research Methodology

The first round of research using the Atmel XMEGA was accepted for publication in International Conference on Cyber Warfare and Security (ICCWS) detailing the results. The success rates and accuracy metrics were slightly lower than desired. Given more time, the CNN design could have been altered further to see if better results could have been achieved. Prior to beginning the research on the Xilinx Kintex-7 FPGA, more CNNs were trained using different parameters and lower trace counts to determine if the results would suffer given 10,000 traces per point as opposed to 27,000 traces per point. The model test accuracies produced by models trained with fewer traces were not noticeably different than the test accuracies produced by the models trained with more traces. This enabled further model training time reduction, from four hours per model to 30 minutes per model. This further enabled training more models to achieve better statistical results during the research using the Xilinx FPGA.

IV. Results and Analysis

Preamble

This chapter discusses the results from both phases of research, including the neural network training, the heatmap analyses, and the results of the attacks based upon the heatmap analysis. Of note, our neural networks have an output layer size of 256, coinciding with the 256 possible classes. The likelihood of randomly guessing the correct class, or key byte is $\frac{1}{256}$, or 0.3906%. This is why you will see accuracy values that seem low.

4.1 Atmel XMEGA Training Results

The training accuracies of each of the ten trained models is displayed in Table 1. The average of all 10 models was 0.862% with a variance of 2.265e−8. Figures 12 and 13 display the losses and accuracies of each model respectively. All 10 plots are superimposed over one another. More training may have been required, as the loss curve does not appear to level off on any of the plots.

Table 1: Final Training Accuracy for each of the 40 CNN Models Trained on the XMEGA microcontroller leakage data

Model	Accuracy (%)
1	0.854
2	0.867
3	0.863
4	0.869
5	0.858
6	0.860
7	0.874
8	0.859
9	0.829
10	0.891

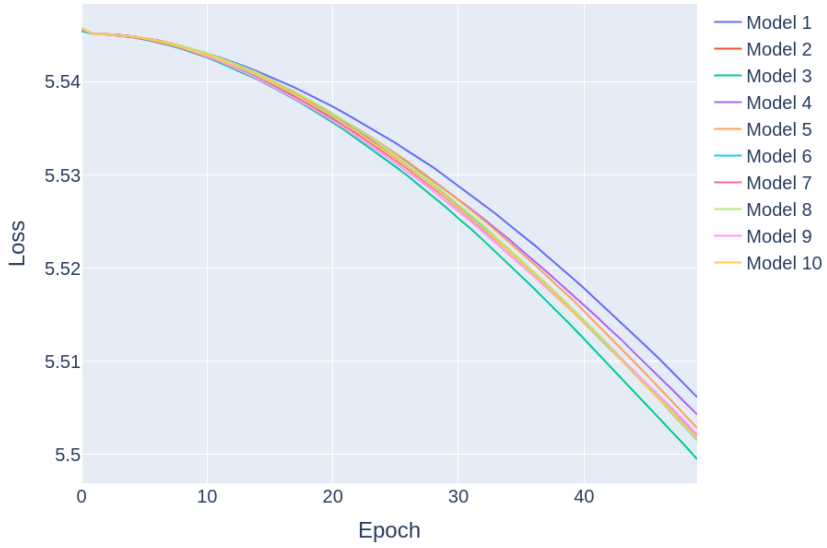

Figure 12: Loss plots for each of the 10 CNN models trained over 50 epochs on the leakage data from the XMEGA microcontroller superimposed over one another

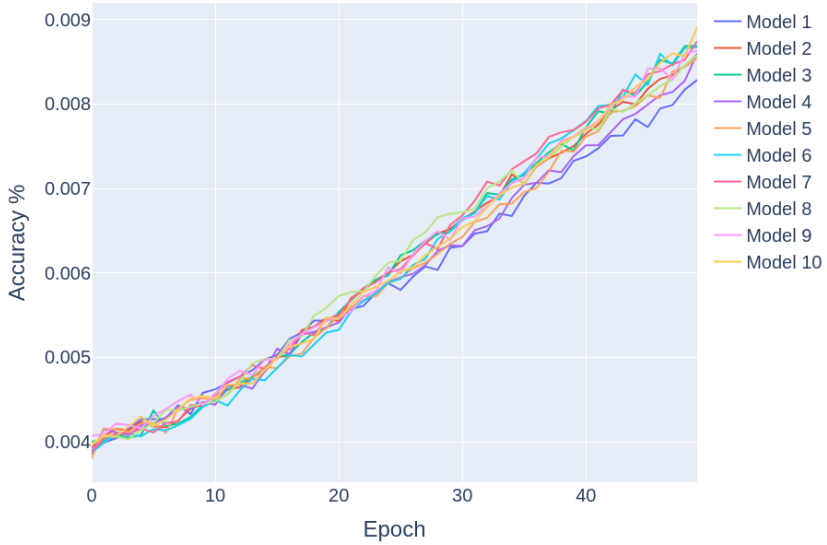

Figure 13: Loss plots for each of the 40 CNN models trained over 100 epochs on the leakage data from the XMEGA microcontroller superimposed over one another

4.2 Xilinx Kintex-7 FPGA Training Results

The training accuracies of each of the forty trained models is displayed in Table 2. The average of all 40 models was 0.752% with a variance of 1.45e−7. Figures 14 and 15 display the losses and accuracies of each model respectively. All 40 plots are superimposed over one another. More training may again be needed as the loss curve here also does not appear to level off.

Table 2: Final Training Accuracy for each of the 40 CNN Models Trained on the Xilinx Kintex-7 FPGA leakage data

Model	Accuracy (%)	Model	Accuracy (%)	Model	Accuracy (%)	Model	Accuracy (%)
1	0.587	11	0.743	21	0.73	31	0.756
2	0.72	12	0.851	22	0.75	32	0.761
3	0.777	13	0.767	23	0.786	33	0.729
4	0.718	14	0.769	24	0.738	34	0.732
5	0.783	15	0.727	25	0.774	35	0.715
6	0.778	16	0.758	26	0.797	36	0.725
7	0.792	17	0.783	27	0.736	37	0.742
8	0.74	18	0.728	28	0.721	38	0.773
9	0.76	19	0.784	29	0.76	39	0.735
10	0.747	20	0.765	30	0.757	40	0.784

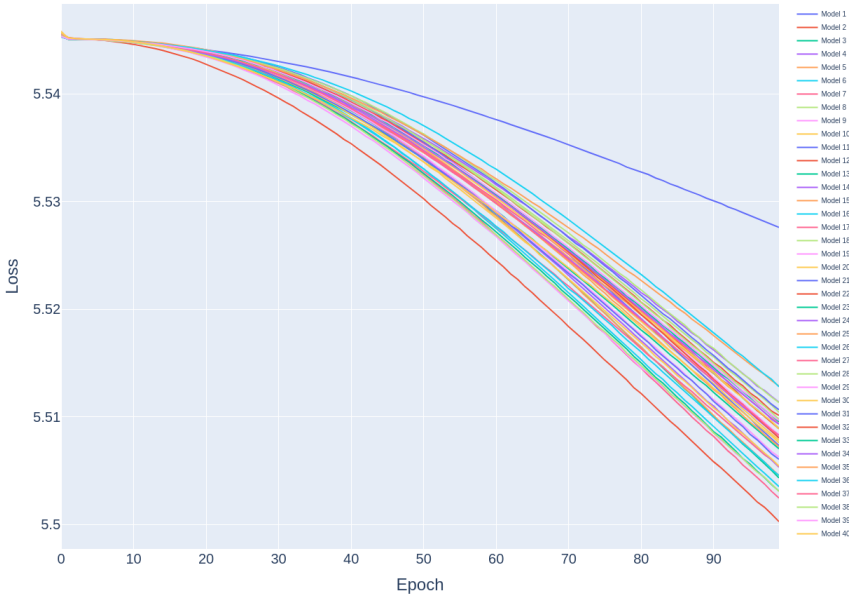

Figure 14: Loss plots for each of the 40 CNN models trained over 100 epochs on the leakage data from the Xilinx Kintex-7 FPGA superimposed over one another

Figure 15: Accuracy plots for each of the 40 CNN models trained over 100 epochs on the leakage data from the Xilinx Kintex-7 FPGA superimposed over one another

4.3 Atmel XMEGA Microcontroller Heatmap Results

With ten neural network models trained on the collected electromagnetic (EM) data, models were tested for accuracy against the 3,000 traces set aside collected from each point. These accuracies for each of the 10 models were then averaged and displayed in the heatmap in Figure 16.

Note that point 6 has the highest average accuracy and point 10 has the lowest average accuracy. Their accuracies were 0.45% and 0.33% respectively. Recall that the likelihood of a random guess of the key byte being correct is 0.39%. However, these numbers alone aren't helpful; the statistical significance must also be determined, that is that they did not occur by chance. When using the t distribution to determine if the results are statistically significant, points 6 and 10 were indeed statistically significant from one another. This is demonstrated in Figure 17, by considering at the confidence intervals of points 6 and 10, and observing no overlap. Additionally, points 8 and 12 have confidence intervals that do not overlap with point 10, thus demonstrating that location does indeed matter when collecting leakage data for side-channel purposes. As displayed in Figure 17 no single point has it's entire confidence interval above the random guess line. This make it impossible to determine with 95% confidence that any of the points are guaranteed to do better than random guessing given a model trained on data gathered from all 25 collection points. However this does not imply that a model trained on data collected only from a single point will not be guaranteed to perform better than random chance. One interesting finding, after collecting all 250 accuracies, is that the overall average accuracy of all data points over all models was 0.391% with a variance of 1.391e−6. This average corresponds to the likelihood of success from randomly guessing the key byte. This test accuracy comes despite the average final training accuracy of the models being 0.862% with a variance of 2.265e−8. This may point to overfitting, but another hypothesis is that this may

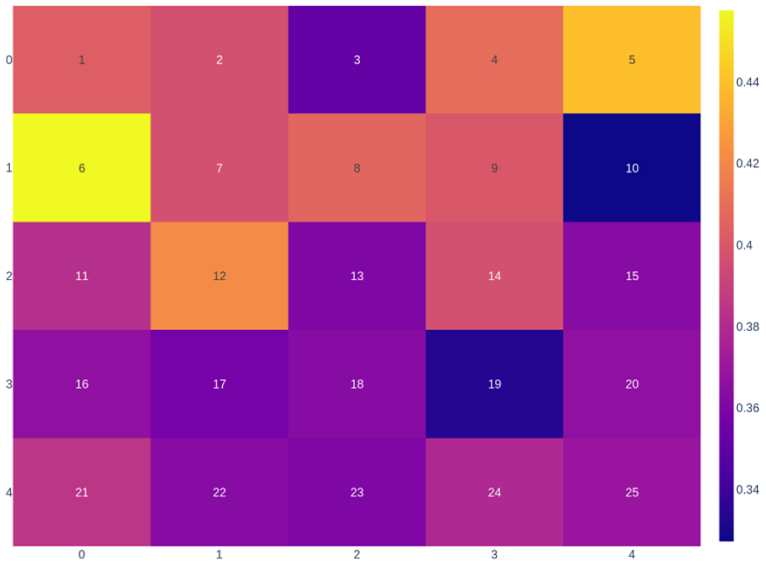

Figure 16: Heat map using averaged accuracies from each of the 10 trained models reflecting the average accuracy at each of the 25 collection points above the Atmel XMEGA Microcontroller

come as a result of the model learning generalized information from the points that have higher average accuracies during testing, and simply memorizing information from the point that have average accuracies lower than the likelihood of randomly guessing. However there is no data to support this claim, which may be a meaningful research objective to explore in future research.

To test the null hypothesis and determine if the accuracies returned by points 6 and 10 were statistically different, a right-tailed t-analysis is performed with an α of 0.05, 9 degrees of freedom, and means of 0.45% for point 6 and 0.33% for point 10. Using the critical value table introduced in Section 3.4 along with this alpha and degrees of freedom, results in a critical value of 1.833. Using the right-tailed t-test results in a t value for this test of 2.828 which is higher than the critical value. Therefore, the null hypothesis, that position does not matter when collecting EM Side-Channel Analysis (SCA) leakage data, is rejected.

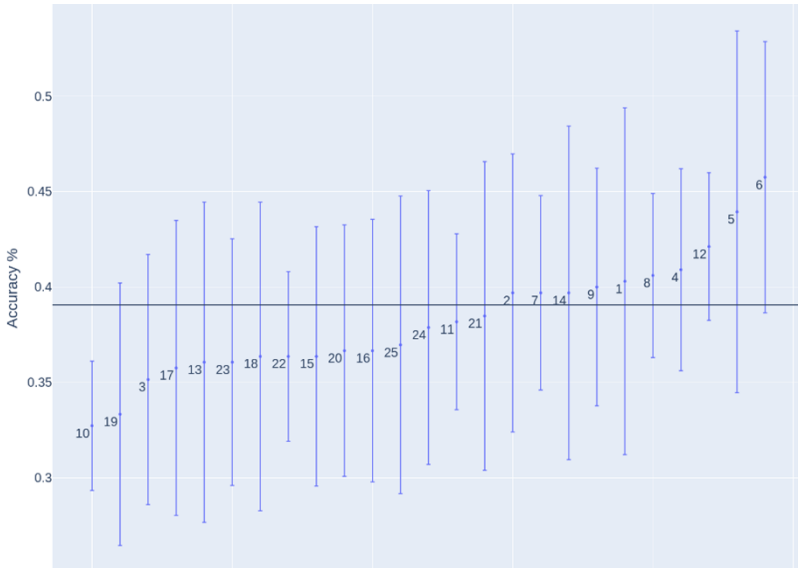

Figure 17: Confidence intervals for the accuracies at each of the 25 collection points. The horizontal line displays the likelihood of success when randomly guessing.

4.4 Atmel XMEGA Microcontroller Attack Results

Determining average test success does not adequately demonstrate the success of a methodology to identify ideal leakage collection points. SCA attacks are also required to determine if the points with higher test accuracy on the larger model indeed retrieve the key byte more successfully than the points identified as having lower test accuracy. To this end, the models trained on data collected from points 6, identified as having more effective leakage, and 10, identified as having less effective leakage did not perform the same when used in SCA attacks. After training these models, the model trained on data from point 6, henceforth referred to as model 6, had a final training accuracy of 2.98%. The model trained on data from point 10, henceforth referred to as model 10, had a final training accuracy of 2.28%. Figure 18 displays the loss over each epoch of model 6, and Figure 19 displays the accuracy over each epoch. Figure 20 displays loss over time of model 10, and Figure 21 displays the corresponding accuracy over time for model 10. These two trained models were

then tested using the same 3,000 trace test data used to determine the best and worst points for the large whole microcontroller model. The test results had model 6 return an accuracy of 3.56% and model 10 return an accuracy of 2.40%.

After running a few attacks, it was observed that if the model was unable to successfully retrieve the key byte prior to the 300th trace, it was never able to retrieve the correct key byte at all. Thus, the attacks were modified to randomly separate the 3,000 collected attack traces into 10 sets of 300 traces, with 10 attacks executed simultaneously. The intent was not just to see which model could return its first successful attack, or which successful attack returned the key byte in fewer traces. Rather, the intent was to increase overall understanding of the model accuracy when running attacks with a variety of traces, input in random order. Thus, attacks continued until each model had executed 30 successful attacks. Model 6 was able to garner 30 successes in 194 attacks, and model 10 was able to garner 30 successes in 549 attacks. This translates to a success rate of 15.49% for model 6 and 5.46% for model 10. Further, this translates to the model trained on data pulled from point 6 successfully retrieving a key byte 2.83 times more often than the model trained at point 10. From this, and from the data demonstrating that points 6 and 10 performed statistically significantly from one another when data was fed into our large model, it is reasonable to conclude that the location of EM collection does matter when performing SCA attacks on the Atmel XMEGA Microcontroller.

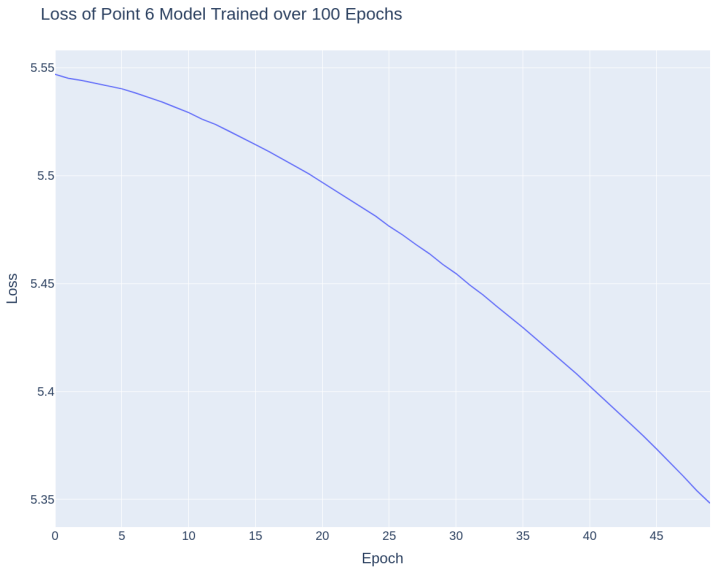

Figure 18: Loss plot for the model trained on the leakage data pulled from point number 6. Note that the loss does not appear to level off toward the last epochs, indicating that a lower loss could have been achieved with more epochs.

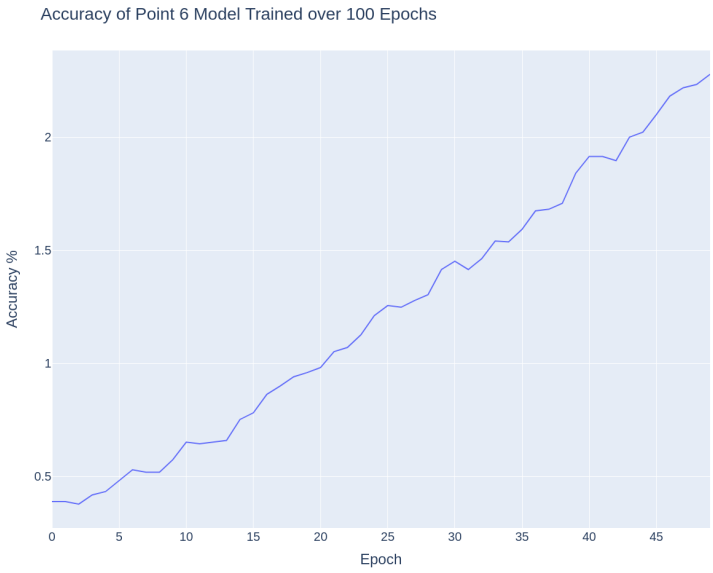

Figure 19: Accuracy plot for the model trained on the leakage data pulled from point number 6. Note that the loss does not appear to level off toward the last epochs, indicating that a more accuracy model could have been achieved with more epochs given enough time.

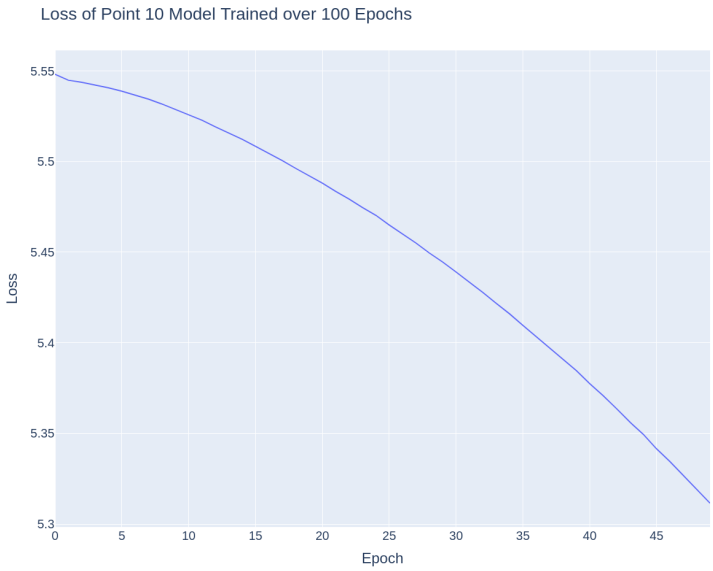

Figure 20: Loss plot for the model trained on the leakage data pulled from point number 10. Note that the loss does not appear to level off toward the last epochs, indicating that a lower loss could have been achieved with more epochs.

Figure 21: Accuracy plot for the model trained on the leakage data pulled from point number 10. Note that the loss does not appear to level off toward the last epochs, indicating that a higher accuracy model could have been achieved with more epochs given enough time.

4.5 Xilinx Kintex-7 FPGA Heatmap Results

After training 40 Convolutional Neural Network (CNN) models on the accumulated data from all 80 leakage collection locations, each was tested on the test data set aside previously, and discussed in Section 3.1.2. These 80 data sets consisted of 3,000 traces, and the average accuracy for each point across all 40 models is displayed in Figure 22.

While it is not immediately obvious from this heatmap, point 61 has the highest average accuracy and point 25 has the lowest average accuracy. Additionally point 40 has an average accuracy that is the median average accuracy of all 80 collection points. The average accuracies for each of these three points is 0.339%, 0.389%, and 0.427% for points 25, 40, and 61 respectively. However, as mentioned in Section 4.3, these numbers alone are not particularly significant, as the statistical significance of the averages and the accuracy data from each point have not yet been determined. The t-distribution was used to determine whether the results given by the models and the tests had any statistical significance; and the results conclude that they were indeed statistically significant. As seen in Figure 23, there are several points that are statistically significant from point 25; 6 points in fact. Points 10, 53, 61, 70, 72, and 76 all performed statistically better than point 25. Point 61 had the highest average, and point 76 had its entire confidence interval perform better than the likelihood of success when randomly guessing, which once again is 0.39%. Despite this point 61 was still chosen as the attack point, as it offered the highest demonstrated average accuracy.

As was the case with the test results for the Atmel XMEGA microcontroller, after collecting all 3,200 accuracies for all 80 points across all 40 models, the overall average accuracy for all 3,200 data points was 0.385% with a variance of 1.244e−6. This average accuracy fell once again right around the percent likelihood of randomly

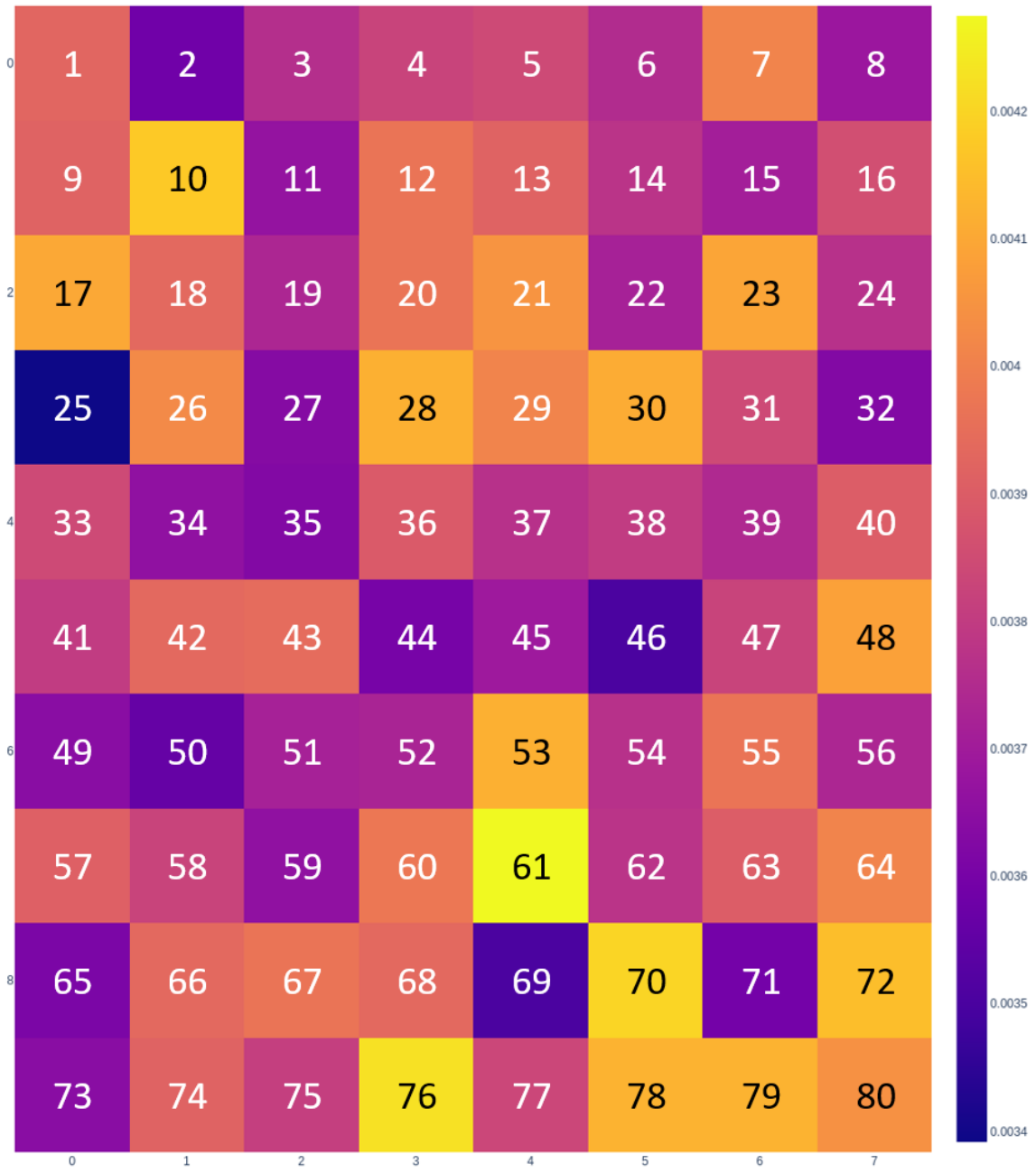

Figure 22: Heat map using averaged accuracies from each of the 40 trained models reflecting the average accuracy at each of the 80 collection points above the Xilinx Kintex-7 FPGA

guessing the correct key byte. This average testing accuracy is reported despite the average accuracy of each of the 40 trained models at the end of training being 0.752%. Again this may point to overfitting, but this is likely a result of the model learning generalized information about the signals that enable classification from the traces at the points that perform well during testing, and memorizing the signal data at the points the perform poorly when tested. This generalizability and memorization combine to result in an overall testing accuracy of 0.385%. However as stated before, there is no data to back up this hypothesis and this is left as a potential research objective in the future.

To test the null hypothesis further and to determine if point 61 or point 76 are statistically different from the population mean, a right-tailed t-test analysis was performed with an α of 0.05, 39 degrees of freedom, and a population mean of 0.385%. The sample mean of point 61 was 0.428%. Using the critical value table introduced in Section 3.4 leads to a critical value of 1.685. Using the right-tailed t-test results in a t value of 2.551 which is higher than the critical value. Therefore the null hypothesis, that position does not matter when collecting EM SCA leakage data, is rejected. Further, when comparing to points, point 25 and point 61, a two-sample t-test reveals a p of 0.001, which is smaller than the alpha of 0.05 which again leads to rejecting the null hypothesis that the two points have the same mean. Again this points to position being important when collecting EM leakage data. If instead, a two-sample t-test is performed on point 61 and point 40, where 40 is the median point with respect to its average accuracy over 40 trained models, this leads to a p of 0.1 which is too high to reject the null hypothesis that points 40 and 61 have different means. The obvious conclusion is that some points perform better than one another definitively, but many other points do not.

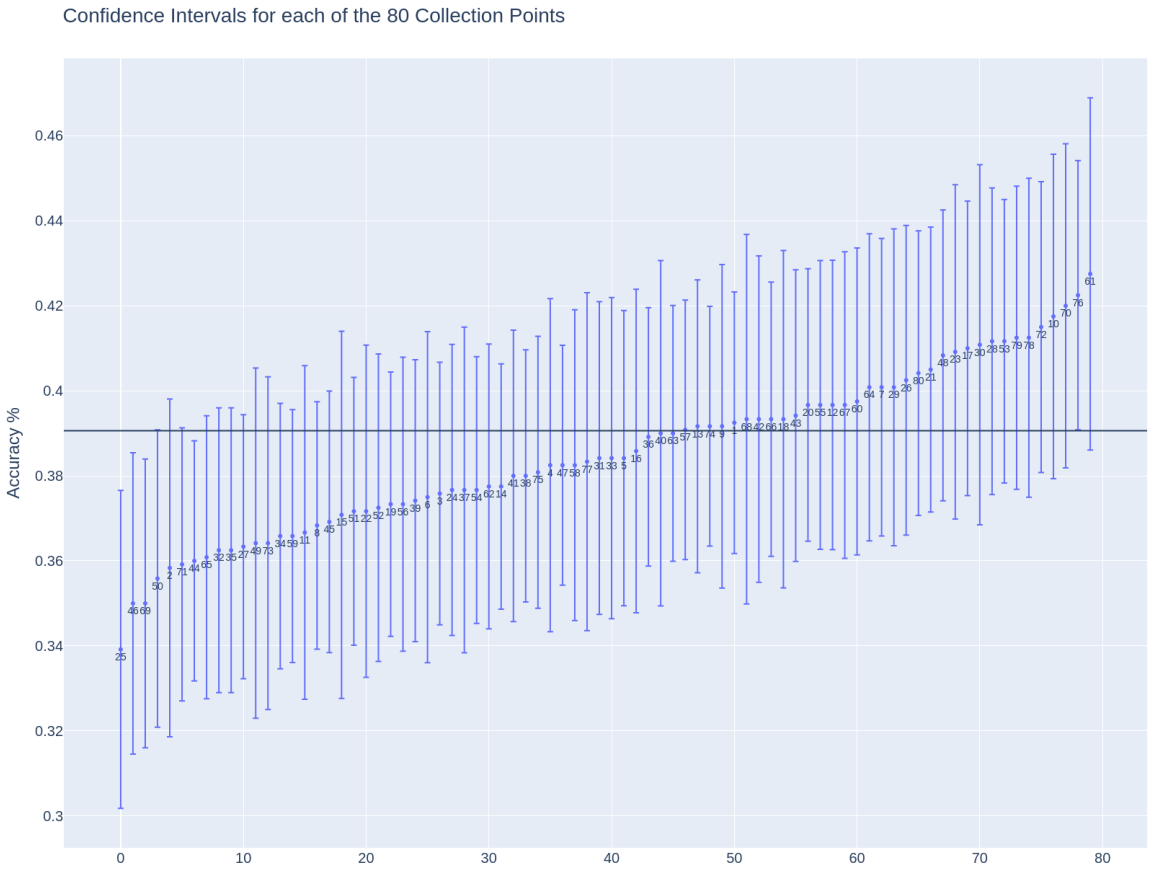

Figure 23: Confidence intervals for the accuracies at each of the 80 collection points above the Xilinx Kintex-7 FPGA. The horizontal line displays the likelihood of success when randomly guessing.

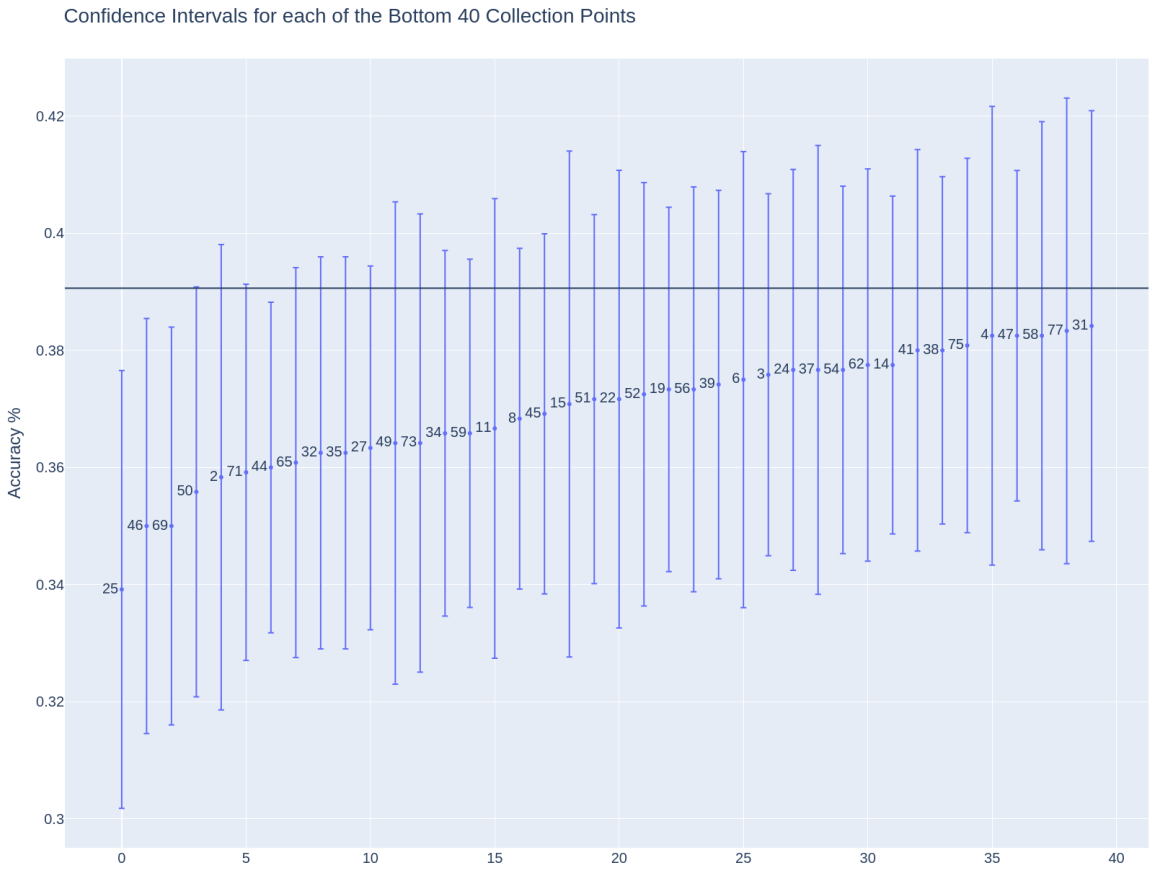

Figure 24: Confidence intervals for the accuracies for the lower performing 40 points above the Xilinx Kintex-7 FPGA. The horizontal line displays the likelihood of success when randomly guessing.

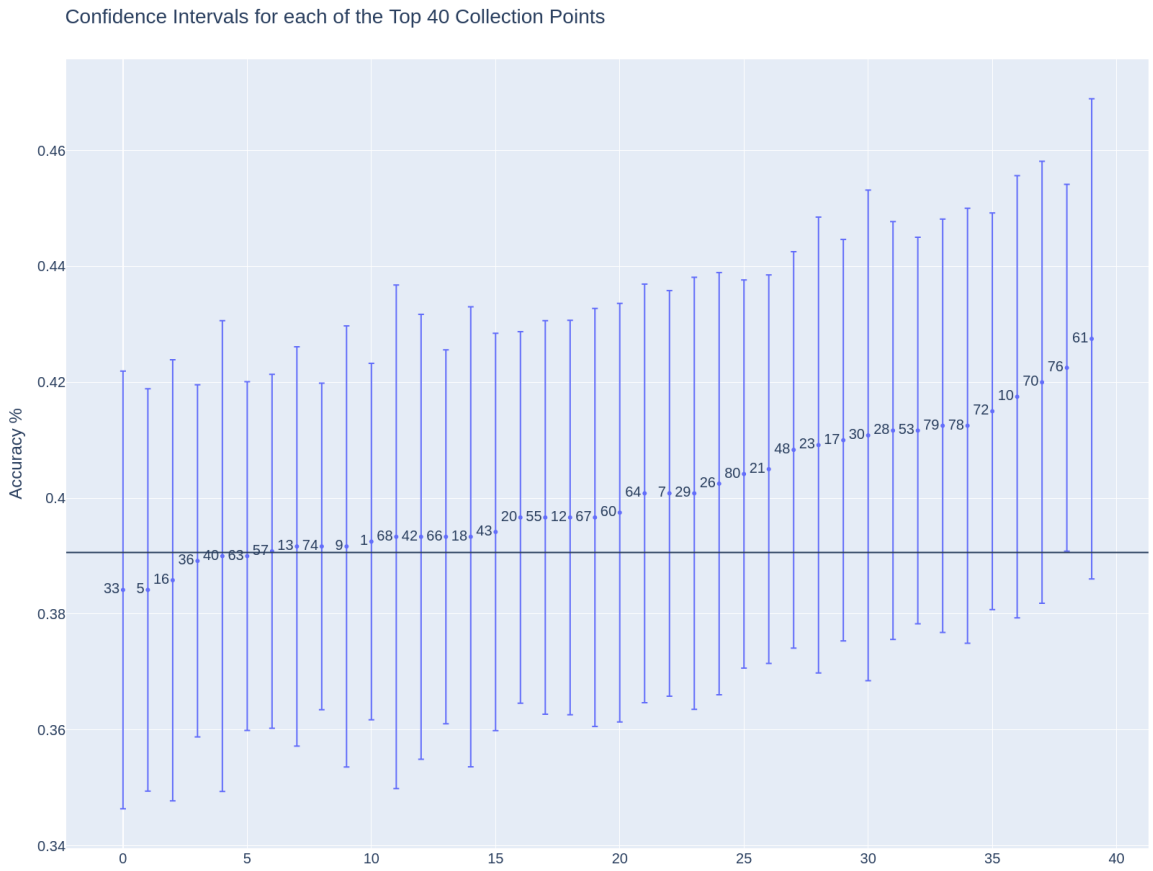

Figure 25: Confidence intervals for the accuracies for the upper performing 40 points above the Xilinx Kintex-7 FPGA. The horizontal line displays the likelihood of success when randomly guessing.

4.6 Xilinx Kintex-7 FPGA Attack Results

Despite having test successes that demonstrate certain points do leak a stronger signal, the average test successes do not adequately demonstrate the effectiveness of our framework in identifying ideal attack locations. Actual attacks are required to determine if the points with higher or lower test accuracy perform similarly when under attack. That is, do the points with higher test accuracy retrieve the key byte correctly more often? To do so, models are trained on data pulled from the three identified points. Point 25, identified as having the lowest accuracy, point 40, identified as having the median average accuracy across all 80 collection points, and point 61, identified as having the highest average accuracy. Once these models had been trained on the 65,000 collected traces for each point, the model trained at point 25, henceforth referred to as model 25, had a final training accuracy of 5.82%. The model trained at point 40, henceforth referred to as model 40, had a final training accuracy of 5.05%. And the model trained at point 61, henceforth referred to as model 61, had a final training accuracy of 5.13%. Figure 26 displays the loss over each epoch of model 25, and Figure 27 displays the accuracy over each epoch. Figure 28 displays loss over time of model 40, and Figure 29 displays the corresponding accuracy over time for model 40. Figure 30 displays the loss over each epoch of model 61, and Figure 31 displays the accuracy over each epoch. Each trained model was then tested with the same test data used to determine the best and worst points on the model trained on data from all 80 points, and the test results had model 25 return an accuracy of 0.27%, model 40 return an accuracy of 0.53%, and model 61 return an accuracy of 0.57%.

When executing these attacks, it was observed that by 300th trace, the model converged on a singular guess. Armed with this observation, the attacks were then modified to randomly separate the 3,000 collected attack traces into 10 sets of 300

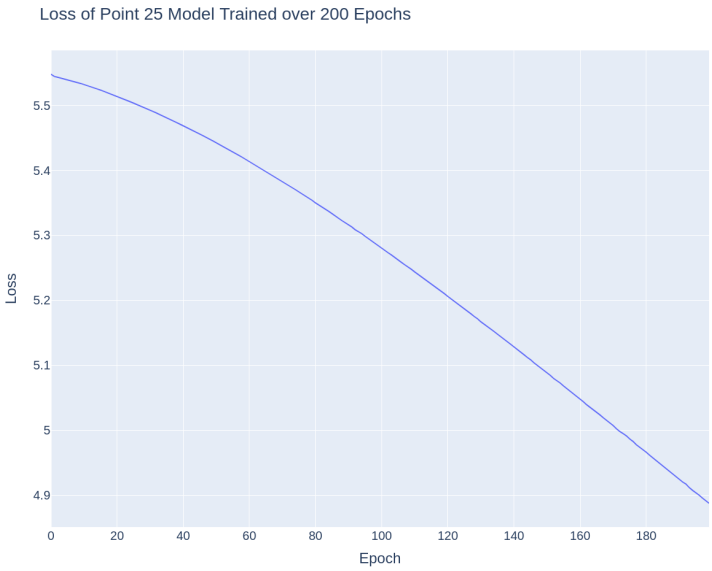

Figure 26: Loss plot for the model trained on the leakage data pulled from point number 25. Note that the loss does not appear to level off toward the last epochs, indicating that a lower loss could have been achieved with more epochs.

Figure 27: Accuracy plot for the model trained on the leakage data pulled from point number 25. Note that the loss does not appear to level off toward the last epochs, indicating that a more accuracy model could have been achieved with more epochs given enough time.

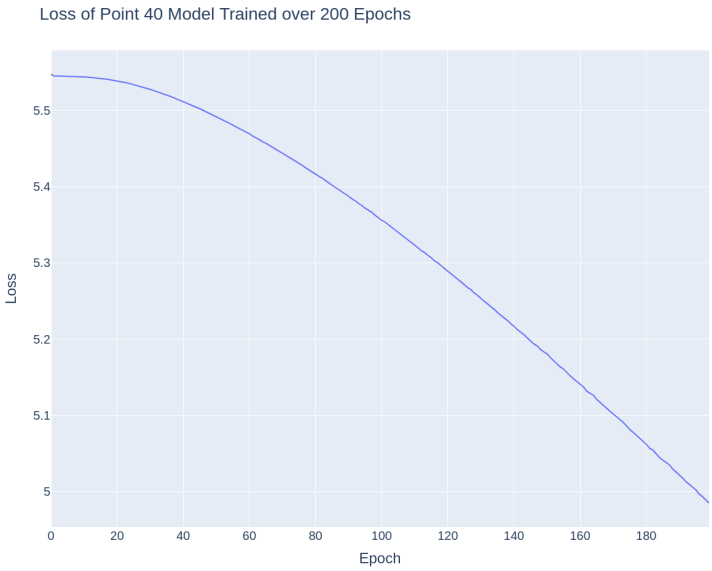

Figure 28: Loss plot for the model trained on the leakage data pulled from point number 40. Note that the loss does not appear to level off toward the last epochs, indicating that a lower loss could have been achieved with more epochs.

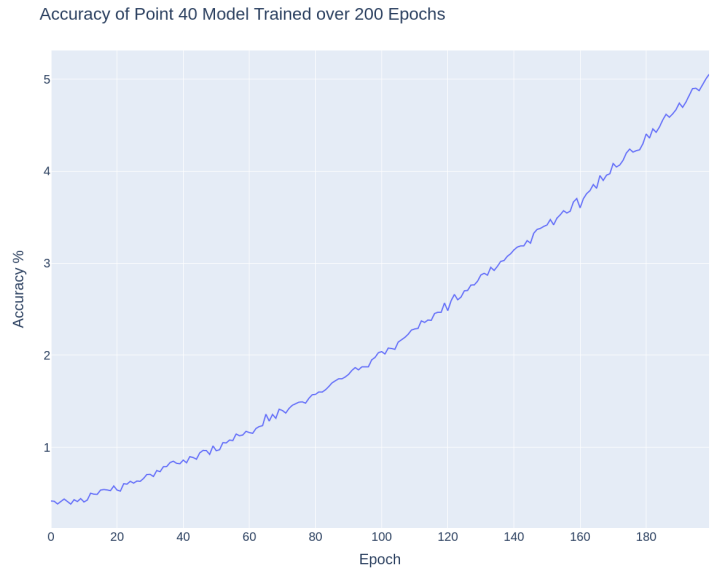

Figure 29: Accuracy plot for the model trained on the leakage data pulled from point number 40. Note that the loss does not appear to level off toward the last epochs, indicating that a higher accuracy model could have been achieved with more epochs given enough time.

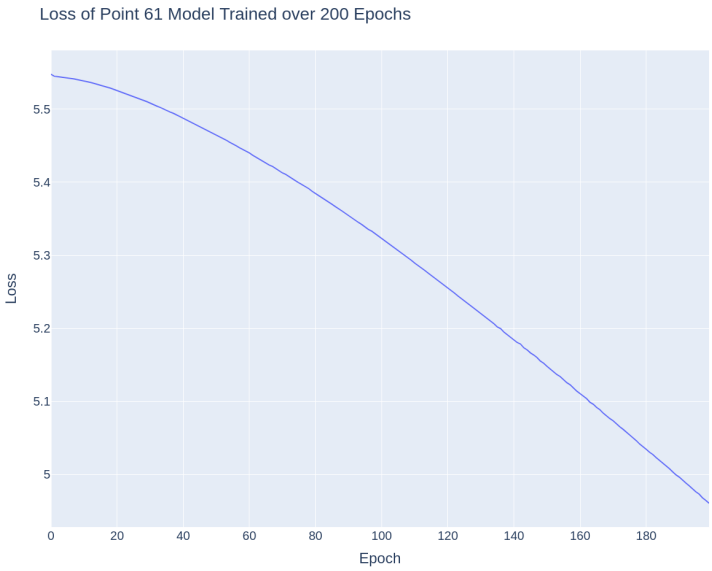

Figure 30: Loss plot for the model trained on the leakage data pulled from point number 61. Note that the loss does not appear to level off toward the last epochs, indicating that a lower loss could have been achieved with more epochs.

Figure 31: Accuracy plot for the model trained on the leakage data pulled from point number 61. Note that the loss does not appear to level off toward the last epochs, indicating that a more accuracy model could have been achieved with more epochs given enough time.

traces and thus execute 10 attacks simultaneously. An attack was considered successful if after all 300 traces were input, the model had converged on the correct key byte. The intent is not just to see which model could return a successful attack first, or the fewest number of traces required by each model for a successful attack. Instead was extract understanding of each of the models' abilities to run successful attacks when executing attacks with traces input in random order. As a result attacks executed until each model had executed 40 successful attacks. Model 25 was able to execute 40 successful attacks after 419 attacks, model 40 was able to execute 40 successful attacks after 347 attacks, and model 61 was able to execute 40 successful attacks after only 314 attacks. This translates to a success rate of 9.546% for model 25, 11.527% for model 40, and 12.739% for model 61. Further, the number of traces required for the models to converge on the correct key byte for each model is 78.025 for model 25, 84.2 for model 40, and 54.925 for model 61. This demonstrates that model 61 successfully retrieves the key byte more often and more quickly during individual attacks than both the median point and the worst point. From this it is clear that physical location of EM collection does matter when performing SCA attacks on the Xilinx Kintex-7 field programmable gate array (FPGA) running a hardware implementation of the Advanced Encryption Standard (AES) encryption scheme which encrypts each byte of the plaintext simultaneously.

4.7 Conclusion

Both the research focusing on the XMEGA microcontroller and the research focusing on the Xilinx FPGA device demonstrated that a heatmap can be created and used to determine an attack location that speeds up key retrieval times. The framework used CNNs to create the heatmap and to perform the attacks. The CNN architecture used could have certainly been improved as the final results were not astounding,

however these results do demonstrate that the hypothesis holds true; a device can be analyzed and a heatmap can be generated using a CNN trained on leakage data. Further that map can then be used to identify ideal attack locations.

V. Conclusions

In this research, a framework was introduced to map the surface of a computing device which can be used to identify ideal Side-Channel Analysis (SCA) attack points. This framework used convolutional neural networks trained on trace data captured above the target devices. Using this framework, this research has been able to demonstrate that neural networks can be used to develop a heatmap that can identify points with a leaked signal more strongly correlated to the secret information being processed by the device in question. It further demonstrates that certain points over two different ChipWhisperer platforms, the Atmel XMEGA microcontroller and a more advanced Xilinx Kintex-7 FPGA, emit stronger SCA leakage and that this leakage can be mapped using a tool such as a Convolutional Neural Network (CNN). Further the maps created using this method can be used by an attacker to identify points from which collecting leakage data and conducting an attack can enable quicker and more successful key recovery. However, even the points identified as having weaker leakage were able to perform successful attacks. This points to a need for an increase in the effectiveness of SCA countermeasures.

5.1 Future Work

This research demonstrated the ability to map out ideal locations for collecting side channel leakage using a neural network. However as referenced in Section 2.7, another method for identifying an ideal leakage collection location has already been created in the SCNIFFER framework [26]. Further, despite demonstrating that leakage collected from different points perform differently when tested against a model trained on leakage data collected from all points, the average success rate for all points when tested for a given model always hovered right around the likelihood of randomly

guessing the correct key. Along this same line of thought, even points that ended with high average accuracies over all models, sometimes performed incredibly poorly on a single model. This may be due to the stochastic nature of CNNs, and is why multiple models were constructed to get a better idea of the distribution. This line of effort should be explored in future work.

This framework was only tested on the first byte of a secret key. However, as the field programmable gate array (FPGA) encrypts all bytes at once, there is a potential for each byte to be encrypted in such a way that the ideal leakage collection point may be different. As FPGA devices can be modified using constraints included in the bitstream to change the bank locations for certain operations, it may be possible to alter the device in such a way that the ideal leakage collection location is different for each FPGA depending upon the bitstream used.

Additionally, while the samples in each trace correspond to the entirety of the first round of Advanced Encryption Standard (AES) encryption, it is not known whether that full time period is necessary to retrieve the secret key. As mentioned in Section 2.2 an attacker must determine which parts of a trace are important in retrieving the secret key. In this case, a neural network was able to perform this task, however it is not clear what exactly the neural network focused on. Perhaps the samples in both experiments could have been trimmed if it were possible to identify the exact window of time that the neural network learned from. Perhaps of the 30,000 and 2,000 samples used in each experiment, only 5,000 and 400 respectively actually mattered. If this is the case then research focusing on identifying the location of the information of importance in the actual trace could be beneficial.

Based upon all of the above, the following are suggestions for future work regarding this field of research:

- A comparison between the framework presented in this thesis to map out the

ideal leakage locations against the SCNIFFER framework.

- A deep dive into the reason that a model trained on data from all collection points with a training accuracy above that of random chance, performing exactly at random chance in test cases.
- Why average top performing points may perform poorly on a given trained model.
- Do different bytes cause different leakage patterns that would result in a different leakage heat map, and thus can an attacker potentially collect from 8 points simultaneously to reduce the time needed to retrieve the entire key?
- Does modifying an FPGA with constraints cause the heat map to change? Or is the leaked signal based more upon physical pin locations than logic operation locations?
- A sensitivity study to determine the location of the critical information related to the secret key in each trace or in each round of encryption. This could speed up training as the input size drops, or allow for models with a larger set of hyper-parameters as the parameter count would not balloon up as fast with a smaller input size.

5.2 Conclusion

The purpose of SCA attacks is to recover secret information from a target device. This research looked to improve this attack process by providing a framework that can be used to map a given device to identify ideal points for collecting electromagnetic (EM) leakage data using a near-field EM probe. The spatial resolution of EM probes enable such a mapping. This framework used the relative performance of a CNN

model trained to detect leakage on data collected from the target device to create a leakage heatmap. This heatmap has been demonstrated as successful in enabling an attacker to identify ideal attack points from which to collect leakage on the target device. The effectiveness of this framework is demonstrated both on the ChipWhisperer Lite Atmel XMEGA microcontroller, and the Xilinx Kintex-7 FPGA. This framework can enable an attacker to see an increase in attack effectiveness of 283% and 33.4% on the XMEGA microcontroller and the Xilinx FPGA targets respectively. This increase in effectiveness comes as attackers attack points corresponding to the highest accuracy as opposed to attacking the point with the lowest accuracy performance of the CNN models.

Appendix A. FPGA Bitstream Generation Code

This is the code originally created with Vivado to generate the bitstreams required to perform the AES encryption on the Xilinx Kintex-7 FPGA device:

```
#*****************************************************************
# Vivado (TM) v2019.2 (64-bit)
#
# cw310-aes128.tcl: Tcl script for re-creating project 'cw310-aes128'
#
# Generated by Vivado on Wed Nov 17 15:33:10 -0500 2021
# Modified by Matthew Dallmeyer - AFIT
# IP Build 2700528 on Thu Nov  7 00:09:20 MST 2019
#
# This file contains the Vivado Tcl commands for re-creating the
# project to the state when this script was generated. In order to
# re-create the project, please source this file in the Vivado Tcl
# Shell.
#
# * Note that the runs in the created project will be configured the
# same way as the original project, however they will not be launched
# automatically. To regenerate the run results please launch the
# synthesis/implementation runs as needed.
#
#*****************************************************************

# Set the reference directory for source file relative paths (by
# default the value is script directory path)
```

```
set origin_dir "."

# Set the project name
set _xil_proj_name_ "cw310-aes128"

variable script_file
set script_file "cw310-aes128.tcl"

# Set the directory paths
set orig_proj_dir "[file normalize "$origin_dir/../vivado"]"
set src_dir [file normalize "${origin_dir}/../src"]
set constr_dir [file normalize "${origin_dir}/../constraints"]
set sim_dir [file normalize "${origin_dir}/../sim"]

set fpga_part "xc7k410tfbg676-2"

# Create project
create_project ${_xil_proj_name_} $orig_proj_dir -part $fpga_part
            -force

# Set the directory path for the new project
set proj_dir [get_property directory [current_project]]

# Set project properties
set obj [current_project]
set_property -name "default_lib" -value "xil_defaultlib" -objects $obj
```

```
set_property -name "enable_vhdl_2008" -value "1" -objects $obj
set_property -name "ip_cache_permissions" -value "read write" -objects
            $obj
set_property -name "ip_output_repo" -value
            "$proj_dir/${_xil_proj_name_}.cache/ip" -objects $obj
set_property -name "mem.enable_memory_map_generation" -value "1"
            -objects $obj
set_property -name "part" -value $fpga_part -objects $obj
set_property -name "sim.central_dir" -value
            "$proj_dir/${_xil_proj_name_}.ip_user_files" -objects $obj
set_property -name "sim.ip.auto_export_scripts" -value "1" -objects $obj
set_property -name "simulator_language" -value "Mixed" -objects $obj

# Create 'sources_1' fileset (if not found)
if {[string equal [get_filesets -quiet sources_1] ""]} {
  create_fileset -srcset sources_1
}

# Set 'sources_1' fileset object
set obj [get_filesets sources_1]
add_files -fileset $obj $src_dir

# IP
create_ip -name xadc_wiz -vendor xilinx.com -library ip -module_name xadc_wiz_0
set obj [get_ips xadc_wiz_0]
set_property -name CONFIG.XADC_STARUP_SELECTION -value "channel_sequencer"
```

```
            -objects $obj
set_property -name CONFIG.CHANNEL_ENABLE_CALIBRATION -value "false" -objects $obj
set_property -name CONFIG.CHANNEL_ENABLE_TEMPERATURE -value "true" -objects $obj
set_property -name CONFIG.AVERAGE_ENABLE_TEMPERATURE -value "true" -objects $obj
set_property -name CONFIG.CHANNEL_ENABLE_VCCINT -value "true" -objects $obj
set_property -name CONFIG.AVERAGE_ENABLE_VCCINT -value "true" -objects $obj
set_property -name CONFIG.CHANNEL_ENABLE_VCCAUX -value "true" -objects $obj
set_property -name CONFIG.AVERAGE_ENABLE_VCCAUX -value "true" -objects $obj
set_property -name CONFIG.CHANNEL_ENABLE_VP_VN -value "false" -objects $obj
set_property -name CONFIG.CHANNEL_ENABLE_VBRAM -value "true" -objects $obj
set_property -name CONFIG.AVERAGE_ENABLE_VBRAM -value "true" -objects $obj
set_property -name CONFIG.CHANNEL_ENABLE_VAUXP0_VAUXN0 -value "true" -objects $obj
set_property -name CONFIG.AVERAGE_ENABLE_VAUXP0_VAUXN0 -value "true" -objects $obj
set_property -name CONFIG.CHANNEL_ENABLE_VAUXP1_VAUXN1 -value "true" -objects $obj
set_property -name CONFIG.AVERAGE_ENABLE_VAUXP1_VAUXN1 -value "true" -objects $obj
set_property -name CONFIG.CHANNEL_ENABLE_VAUXP8_VAUXN8 -value "true" -objects $obj
set_property -name CONFIG.AVERAGE_ENABLE_VAUXP8_VAUXN8 -value "true" -objects $obj
set_property -name CONFIG.ENABLE_VBRAM_ALARM -value "true" -objects $obj
set_property -name CONFIG.SEQUENCER_MODE -value "Continuous" -objects $obj
set_property -name CONFIG.EXTERNAL_MUX_CHANNEL -value "VP_VN" -objects $obj
set_property -name CONFIG.SINGLE_CHANNEL_SELECTION -value "TEMPERATURE"
            -objects $obj

# Set 'sources_1' fileset properties
set obj [get_filesets sources_1]
set_property -name "top" -value "cw310_top" -objects $obj
```

```
set_property -name "top_auto_set" -value "0" -objects $obj

# Create 'constrs_1' fileset (if not found)
if {[string equal [get_filesets -quiet constrs_1] ""]} {
  create_fileset -constrset constrs_1
}

# Set 'constrs_1' fileset object
set obj [get_filesets constrs_1]
add_files -fileset $obj $constr_dir

# Create 'sim_1' fileset (if not found)
if {[string equal [get_filesets -quiet sim_1] ""]} {
  create_fileset -simset sim_1
}


# Set 'sim_1' fileset object
set obj [get_filesets sim_1]
add_files -fileset $obj $sim_dir

# Set 'sim_1' fileset properties
set obj [get_filesets sim_1]
set_property -name "top" -value "cw310_top" -objects $obj
set_property -name "top_lib" -value "xil_defaultlib" -objects $obj
```

```
# Create 'synth_1' run (if not found)
if {[string equal [get_runs -quiet synth_1] ""]} {
    create_run -name synth_1 -part $fpga_part -flow {Vivado Synthesis
                2019} -strategy "Vivado Synthesis Defaults"
                -report_strategy {No Reports} -constrset constrs_1
} else {
  set_property strategy "Vivado Synthesis Defaults" [get_runs synth_1]
  set_property flow "Vivado Synthesis 2019" [get_runs synth_1]
}
set obj [get_runs synth_1]
set_property set_report_strategy_name 1 $obj
set_property report_strategy {Vivado Synthesis Default Reports} $obj
set_property set_report_strategy_name 0 $obj

set obj [get_runs synth_1]
set_property -name "part" -value $fpga_part -objects $obj
set_property -name "strategy" -value "Vivado Synthesis Defaults" -objects $obj

# set the current synth run
current_run -synthesis [get_runs synth_1]

# Create 'impl_1' run (if not found)
if {[string equal [get_runs -quiet impl_1] ""]} {
    create_run -name impl_1 -part $fpga_part -flow {Vivado Implementation
            2019} -strategy "Vivado Implementation Defaults" -report_strategy
            {No Reports} -constrset constrs_1 -parent_run synth_1
```

```
} else {
  set_property strategy "Vivado Implementation Defaults" [get_runs impl_1]
  set_property flow "Vivado Implementation 2019" [get_runs impl_1]
}

set obj [get_runs impl_1]
set_property set_report_strategy_name 1 $obj
set_property report_strategy {Vivado Implementation Default Reports} $obj
set_property set_report_strategy_name 0 $obj

set obj [get_runs impl_1]
set_property -name "part" -value $fpga_part -objects $obj
set_property -name "strategy" -value "Vivado Implementation Defaults" -objects
            $obj
set_property -name "steps.phys_opt_design.is_enabled" -value "1" -objects
            $obj
set_property -name "steps.write_bitstream.args.readback_file" -value "0" -objects
            $obj
set_property -name "steps.write_bitstream.args.verbose" -value "0" -objects
            $obj

# set the current impl run
current_run -implementation [get_runs impl_1]

# generate the bitstream
launch_runs impl_1 -to_step write_bitstream -jobs 4
```

```
#generate bin for flash chip
wait_on_run impl_1
write_cfgmem -format bin -interface SPIx1 -size 32 -loadbit "up 0x0
            $proj_dir/${_xil_proj_name_}.runs/impl_1/cw310_top.bit"
            -file cw310_top.bin -force
```

Bibliography

1. François Chollet. *What is Deep Learning*, page 1–25. Manning Publications, second edition, 2017.

2. Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. Cryptology ePrint Archive, Paper 2019/439, 2019. `https://eprint.iacr.org/2019/439`.

3. Gavin Wright and Alexander S Gillis. What is a side-channel attack?, Apr 2021.

4. Debayan Das and Shreyas Sen. Electromagnetic and power side-channel analysis: Advanced attacks and low-overhead generic countermeasures through white-box approach. *Cryptography*, 4:1–19, 12 2020.

5. Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. X-deepsca: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery.

6. Honggang Yu, Haoqi Shan, Maximillian Panoff, and Yier Jin. Cross-device profiled side-channel attacks using meta-transfer learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 703–708, 2021.

7. Huanyu Wang, Sebastian Forsmark, Martin Brisfors, and Elena Dubrova. Multi-source training deep-learning side-channel attacks. In *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 58–63, 2020.

8. Thorben Moos, Felix Wegener, and Amir Moradi. Dl-la: Deep learning leakage assessment a modern roadmap for sca evaluations, 2019.

9. Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for sidechannel resistance validation, 2011.

10. Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the x2-test. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018:209–237, 2018.

11. Roxy Peck, Chris Olsen, and Jay L. Devore. *Comparing Two Populations or Treatments*, page 583–646. Brooks/Cole Cengage Learning, 2012.

12. Roxy Peck, Chris Olsen, and Jay L. Devore. *The Analysis of Categorical Data and Goodness-of Fit Tests*, page 647–688. Brooks/Cole Cengage Learning, 2012.

13. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *5.5: Simple Power Analysis*, page 115–116. Springer, 2007.

14. Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ascad database. *Journal of Cryptographic Engineering*, 10:163–188, 6 2020.

15. Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. A survey of electromagnetic side-channel attacks and discussion on their case-progressing potential for digital forensics. 3 2019.

16. Steven W. Smith. *Chapter 3: ADC and DAC; The Sampling Theorem*, page 39–44. California Technical Pub., 1997.

17. Abhishek Sachdeva. A study of encryption algorithms aes, des and rsa for security. *Global Journal of Computer Science and Technology*, 13(15):32–40, Mar. 2013.

18. Morris J Dworkin, Elaine B Barker, James R Nechvatal, James Foti, Lawrence E Bassham, E Roback, and James F Dray. Advanced encryption standard (aes). *Federal Information Processing Standards Publication*, 197, Nov 2001.

19. Michael Healy, Thomas Newe, and Elfed Lewis. *Analysis of Hardware Encryption Versus Software Encryption on Wireless Sensor Network Motes*. Springer Berlin Heidelberg.

20. Sara Brown. Machine learning, explained, Apr 2021.

21. Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

22. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

23. Olivier Bronchain, Julien M Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations, 2019.

24. Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, and Jean-Pierre Seifert. Simple photonic emission analysis of aes. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 41–57, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

25. Christos Andrikos, Lejla Batina, Lukasz Chmielewski, Liran Lerman, Vasilios Mavroudis, Kostas Papagiannopoulos, Guilherme Perin, Giorgos Rassias, and

Alberto Sonnino. Location, location, location: Revisiting modeling and exploitation for location-based side channel leakages.

26. Josef Danial, Debayan Das, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. Scniffer: Low-cost, automated, efficient electromagnetic side-channel sniffing. *IEEE Access*, 8:173414–173427, 2020.

Acronyms

AES Advanced Encryption Standard. 3, 5, 7, 15, 16, 17, 18, 28, 29, 32, 34, 35, 37, 38, 72, 75, 78

CNN Convolutional Neural Network. iv, 4, 5, 6, 7, 22, 23, 25, 29, 30, 32, 37, 44, 51, 62, 72, 73, 74, 75, 76, 77

DES data encryption standard. 15

DNN Deep Neural Network. 28

DUT device under test. 3, 8, 11, 26, 27

EM electromagnetic. iv, 1, 2, 3, 4, 5, 7, 8, 11, 12, 14, 15, 28, 29, 30, 31, 32, 34, 37, 39, 46, 56, 57, 59, 64, 72, 76

FPGA field programmable gate array. iv, 5, 26, 27, 30, 37, 40, 44, 45, 48, 51, 72, 75, 76, 77

GPU graphics processing unit. 44, 45

GS/s giga-samples per second. 32, 37, 38

HDL hardware design language. 26

ICCWS International Conference on Cyber Warfare and Security. 51

MI Mutual Information. 26

MS/s mega-samples per second. 34

NIST National Institute of Standards and Technology. 8, 15

NLL negative log likelihood. 25, 26, 29, 40

PI perceived information. 26

ReLU rectified linear unit. 23

RSA Rivest-Shamir-Adleman. 15

SCA Side-Channel Analysis. iv, 1, 2, 3, 5, 6, 7, 8, 9, 11, 12, 19, 26, 27, 29, 30, 31, 34, 39, 40, 57, 58, 59, 64, 72, 74, 76

SGD stochastic gradient descent. 24

SNR signal-to-noise ratio. 3, 4, 29

TVLA test vector leakage assessment. 29

USAF United States Air Force. 30, 34, 38

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
23–03–2023	Master's Thesis	Sept 2021 — Mar 2023

4. TITLE AND SUBTITLE

Characterizing Location-Based Electromagnetic Leakage of Computing Devices using Convolutional Neural Networks to Increase the Effectiveness of Side-Channel Analysis Attacks

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Heffron, Ian, Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-23-M-030

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

SCA attacks aim to recover some sort of secret information, often in the form of a cipher key, from a target device. Some of these attacks focus on either power-based leakage, or EM-based leakage. Neural networks have recently gained in popularity as tools in SCA attacks. Near-field EM probes with high-spatial resolution enable attackers to isolate physical locations above a processor. This enables attackers to exploit the spatial dependencies of algorithms running on said processor. These spatial dependencies result in different physical locations above a chip emanating different signal strengths. The strengths of different locations can be mapped using the performance of a neural network trained to detect secret information on near-field leakage data. Our contribution uses this mapping to identify ideal near-field leakage collection locations from which to conduct an attack. This paper demonstrates the effectiveness of this technique in reducing the time needed to conduct a successful EM SCA attack.

15. SUBJECT TERMS

convolutional neural network (CNN), deep learning, machine learning, side-channel analysis (SCA), electromagnetic leakage, field-programmable gate array (FPGA), side-channel attacks

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Lt Col James Dean, AFIT/ENG
U	U	U	UU	104	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x9999; james.dean@afit.edu