CSI NN

Reverse engineering of neural network architectures through electromagnetic side channel

Batina, Lejla; Jap, Dirmanto; Bhasin, Shivam; Picek, Stjepan

Publication date
2019

Document Version
Final published version

Published in
Proceedings of the 28th USENIX Security Symposium

Citation (APA)
Batina, L., Jap, D., Bhasin, S., & Picek, S. (2019). CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *Proceedings of the 28th USENIX Security Symposium* (pp. 515-532). USENIX Association.

CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel

Lejla Batina, ***Radboud University, The Netherlands;*** **Shivam Bhasin and Dirmanto Jap,** ***Nanyang Technological University, Singapore;*** **Stjepan Picek,** ***Delft University of Technology, The Netherlands***

https://www.usenix.org/conference/usenixsecurity19/presentation/batina

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the 28th USENIX Security Symposium is sponsored by USENIX.

CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel

Lejla Batina
Radboud University, The Netherlands

Shivam Bhasin
Nanyang Technological University, Singapore

Dirmanto Jap
Nanyang Technological University, Singapore

Stjepan Picek
Delft University of Technology, The Netherlands

Abstract

Machine learning has become mainstream across industries. Numerous examples prove the validity of it for security applications. In this work, we investigate how to reverse engineer a neural network by using side-channel information such as timing and electromagnetic (EM) emanations. To this end, we consider multilayer perceptron and convolutional neural networks as the machine learning architectures of choice and assume a non-invasive and passive attacker capable of measuring those kinds of leakages.

We conduct all experiments on real data and commonly used neural network architectures in order to properly assess the applicability and extendability of those attacks. Practical results are shown on an ARM Cortex-M3 microcontroller, which is a platform often used in pervasive applications using neural networks such as wearables, surveillance cameras, etc. Our experiments show that a side-channel attacker is capable of obtaining the following information: the activation functions used in the architecture, the number of layers and neurons in the layers, the number of output classes, and weights in the neural network. Thus, the attacker can effectively reverse engineer the network using merely side-channel information such as timing or EM.

1 Introduction

Machine learning, and more recently deep learning, have become hard to ignore for research in distinct areas, such as image recognition [25], robotics [21], natural language processing [47], and also security [53, 26] mainly due to its unquestionable practicality and effectiveness. Ever increasing computational capabilities of the computers of today and huge amounts of data available are resulting in much more complex machine learning architectures than it was envisioned before. As an example, AlexNet architecture consisting of 8 layers was the best performing algorithm in image classification task ILSVRC2012 (`http://www.image-net.org/challenges/LSVRC/2012/`). In 2015, the best performing architecture for the same task was ResNet consisting of 152 layers [15]. This trend is not expected to stagnate any time soon, so it is prime time to consider machine/deep learning from a novel perspective and in new use cases. Also, deep learning algorithms are gaining popularity in IoT edge devices such as sensors or actuators, as they are indispensable in many tasks, like image classification or speech recognition. As a consequence, there is an increasing interest in deploying neural networks on low-power processors found in always-on systems, e.g., ARM Cortex-M microcontrollers.

In this work, we focus on two neural network algorithms: multilayer perceptron (MLP) and convolutional neural networks (CNNs). We consider feed-forward neural networks and consequently, our analysis is conducted on such networks only.

With the increasing number of design strategies and elements to use, fine-tuning of hyper-parameters of those algorithms is emerging as one of the main challenges. When considering distinct industries, we are witnessing an increase in intellectual property (IP) models strategies. Basically, in cases when optimized networks are of commercial interest, their details are kept undisclosed. For example, EMVCo (formed by MasterCard and Visa to manage specifications for payment systems and to facilitate worldwide interoperability) nowadays requires deep learning techniques for security evaluations [43]. This has an obvious consequence in: 1) security labs generating (and using) neural networks for evaluation of security products and 2) they treat them as IP, exclusively for their customers.

There are also other reasons for keeping the neural network architectures secret. Often, these pre-trained models might provide additional information regarding the training data, which can be very sensitive. For example, if the model is trained based on a medical record of a patient [9], confidential information could be encoded into the network during the training phase. Also, machine learning models that are used for guiding medical treatments are often based on a patient's genotype making this extremely sensitive from the privacy perspective [10]. Even if we disregard privacy issues,

obtaining useful information from neural network architectures can help acquiring trade secrets from the competition, which could lead to competitive products without violating intellectual property rights [3]. Hence, determining the layout of the network with trained weights is a desirable target for the attacker. One could ask the following question: Why would an attacker want to reverse engineer the neural network architecture instead of just training the same network on its own? There are several reasons that are complicating this approach. First, the attacker might not have access to the same training set in order to train his own neural network. Although this is admittedly a valid point, recent work shows how to solve those limitations [49]. Second, as the architectures have become more complex, there are more and more parameters to tune and it could be extremely difficult for the attacker to pinpoint the same values for the parameters as in the architecture of interest.

After motivating our use case, the main question that remains is on the feasibility of reverse engineering such architectures. Physical access to a device could allow readily reverse engineering based on the binary analysis. However, in a confidential IP setting, standard protections like blocking binary readback, blocking JTAG access [20], code obfuscation, etc. are expected to be in place and preventing such attacks. Nevertheless, even when this is the case, a viable alternative is to exploit side-channel leakages.

Side-channel analysis attacks have been widely studied in the community of information security and cryptography, due to its potentially devastating impact on otherwise (theoretically) secure algorithms. Practically, the observation that various physical leakages such as timing delay, power consumption, and electromagnetic emanation (EM) become available during the computation with the (secret) data has led to a whole new research area. By statistically combining this physical observation of a specific internal state and hypothesis on the data being manipulated, it is possible to recover the intermediate state processed by the device.

In this study, our aim is to highlight the potential vulnerabilities of standard (perhaps still naive from the security perspective) implementations of neural networks. At the same time, we are unaware of any neural network implementation in the public domain that includes side-channel protection. For this reason, we do not just pinpoint to the problem but also suggest some protection measures for neural networks against side-channel attacks. Here, we start by considering some of the basic building blocks of neural networks: the number of hidden layers, the basic multiplication operation, and the activation functions.

For instance, the complex structure of the activation function often leads to conditional branching due to the necessary exponentiation and division operations. Conditional branching typically introduces input-dependent timing differences resulting in different timing behavior for different activation function, thus allowing the function identification. Also, we notice that by observing side-channel leakage, it is possible to deduce the number of nodes and the number of layers in the networks.

In this work, we show it is possible to recover the layout of unknown networks by exploiting the side-channel information. Our approach does not need access to training data and allows for network recovery by feeding known random inputs to the network. By using the known divide-and-conquer approach for side-channel analysis, (i.e., the attacker's ability to work with a feasible number of hypotheses due to, e.g., the architectural specifics), the information at each layer could be recovered. Consequently, the recovered information can be used as input for recovering the subsequent layers.

We note that there exists somewhat parallel research to ours also on reverse engineering by "simply" observing the outputs of the network and training a substitute model. Yet, this task is not so simple since one needs to know what kind of architecture is used (e.g., convolutional neural network or multilayer perceptron, the number of layers, the activation functions, access to training data, etc.) while limiting the number of queries to ensure the approach is realistic [39]. Some more recent works have tried to overcome a few of the highlighted limitations [49, 18].

To our best knowledge, this kind of observation has never been used before in this context, at least not for leveraging on (power/EM) side-channel leakages with reverse engineering the neural networks architecture as the main goal. We position our results in the following sections in more detail. To summarize, our main motivation comes from the ever more pervasive use of neural networks in security-critical applications and the fact that the architectures are becoming proprietary knowledge for the security evaluation industry. Hence, reverse engineering a neural network has become a new target for the adversaries and we need a better understanding of the vulnerabilities to side-channel leakages in those cases to be able to protect the users' rights and data.

1.1 Related Work

There are many papers considering machine learning and more recently, deep learning for improving the effectiveness of side-channel attacks. For instance, a number of works have compared the effectiveness of classical profiled side-channel attacks, so-called template attacks, against various machine learning techniques [30, 19]. Lately, several works explored the power of deep learning in the context of side-channel analysis [32]. However, this line of work is using machine learning to derive a new side-channel distinguisher, i.e., the selection function leading to the key recovery.

On the other hand, using side-channel analysis to attack machine learning architectures has been much less investigated. Shokri et al. investigate the leakage of sensitive information from machine learning models about individual

data records on which they were trained [44]. They show that such models are vulnerable to membership inference attacks and they also evaluate some mitigation strategies. Song et al. show how a machine learning model from a malicious machine learning provider can be used to obtain information about the training set of a model [45]. Hua et al. were first to reverse engineer two convolutional neural networks, namely AlexNet and SqueezeNet through memory and timing side-channel leaks [17]. The authors measure side-channel through an artificially introduced hardware trojan. They also need access to the original training data set for the attack, which might not always be available. Lastly, in order to obtain the weights of neural networks, they attack a very specific operation, i.e., zero pruning [40]. Wei et al. have also performed an attack on an FPGA-based convolutional neural network accelerator [52]. They recovered the input image from the collected power consumption traces. The proposed attack exploits a specific design choice, i.e., the line buffer in a convolution layer of a CNN.

In a nutshell, both previous reverse engineering efforts using side-channel information were performed on very special designs of neural networks and the attacks had very specific and different goals. Our work is more generic than those two as it assumes just a passive adversary able to measure physical leakages and our strategy remains valid for a range of architectures and devices. Although we show the results on the chips that were depackaged prior to experiments in order to demonstrate the leakage available to powerful adversaries, our findings remain valid even without depackaging. Basically, having EM as an available source of side-channel leakage, it comes down to using properly designed antennas and more advanced setups, which is beyond the scope of this work.

Several other works doing somewhat related research are given as follows. Ohrimenko et al. used a secure implementation of MapReduce jobs and analyzed intermediate traffic between reducers and mappers [37]. They showed how an adversary observing the runs of typical jobs can infer precise information about the inputs. In a follow-up work they discuss how machine learning algorithms can be exploited by various side-channels [38]. Consequently, they propose data-oblivious machine learning algorithms that prevent exploitation of side channels induced by memory, disk, and network accesses. They note that side-channel attacks based on power and timing leakages are out of the scope of their work. Xu et al. introduced controlled-channel attacks, which is a type of side-channel attack allowing an untrusted operating system to extract large amounts of sensitive information from protected applications [54]. Wang and Gong investigated both theoretically and experimentally how to steal hyper-parameters of machine learning algorithms [51]. In order to mount the attack in practice, they estimate the error between the true hyper-parameter and the estimated one.

In this work, we further explore the problem of reverse engineering of neural networks from a more generic perspective. The closest previous works to ours have reverse engineered neural networks by using cache attacks that work on distinct CPUs and are basically micro-architectural attacks (albeit using timing side-channel). Our approach utilizes EM side-channel on small embedded devices and it is supported by practical results obtained on a real-world architecture. Finally, our attack is able to recover both the hyper-parameters (parameter external to the model, e.g., the number of layers) and parameters (parameter internal to the model, like weights) of neural networks.

1.2 Contribution and Organization

The main contributions of this paper are:

1. We describe full reverse engineering of neural network parameters based on side-channel analysis. We are able to recover the key parameters such as activation function, pre-trained weights, number of hidden layers and neurons in each layer. The proposed technique does not need any information on the (sensitive) training data as that information is often not even available to the attacker. We emphasize that, for our attack to work, we require the knowledge of some inputs/outputs and side-channel measurements, which is a standard assumption for side-channel attacks.
2. All the proposed attacks are practically implemented and demonstrated on two distinct microcontrollers (i.e., 8-bit AVR and 32-bit ARM).
3. We highlight some interesting aspects of side-channel attacks when dealing with real numbers, unlike in everyday cryptography. For example, we show that even a side-channel attack that failed can provide sensitive information about the target due to the precision error.
4. Finally, we propose a number of mitigation techniques rendering the attacks more difficult.

We emphasize that the simplicity of our attack is its strongest point, as it minimizes the assumption on the adversary (no pre-processing, chosen-plaintext messages, etc.)

2 Background

In this section, we give details about artificial neural networks we consider in this paper and their building blocks. Next, we discuss the concepts of side-channel analysis and several types of attacks we use in this paper.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) is an umbrella notion for all computer systems loosely inspired by biological neural networks. Such systems are able to "learn" from examples, which makes them a strong (and very popular) paradigm in

the machine learning domain. Any ANN is built from a number of nodes called artificial neurons. The nodes are connected in order to transmit a signal. Usually, in an ANN, the signal at the connection between artificial neurons is a real number and the output of each neuron is calculated as a nonlinear function of the sum of its inputs. Neurons and connections have weights that are adjusted as the learning progresses. Those weights are used to increase or decrease the strength of a signal at a connection. In the rest of this paper, we use the notions of an artificial neural network, neural network, and network interchangeably.

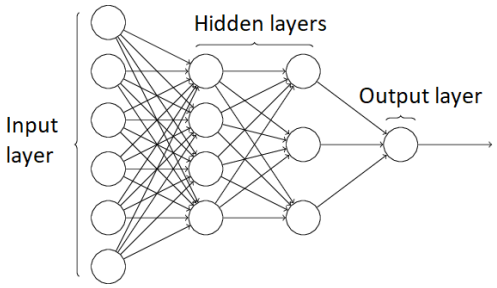


Figure 1: Multilayer perceptron.

2.1.1 Multilayer Perceptron

A very simple type of a neural network is called perceptron. A perceptron is a linear binary classifier applied to the feature vector as a function that decides whether or not an input belongs to some specific class. Each vector component has an associated weight w_i and each perceptron has a threshold value θ. The output of a perceptron equals "1" if the direct sum between the feature vector and the weight vector is larger than zero and "-1" otherwise. A perceptron classifier works only for data that are linearly separable, i.e., if there is some hyperplane that separates all the positive points from all the negative points [34].

By adding more layers to a perceptron, we obtain a multilayer perceptron algorithm. Multilayer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. It consists of multiple layers of nodes in a directed graph, where each layer is fully connected to the next one. Consequently, each node in one layer connects with a certain weight w to every node in the following layer. Multilayer perceptron algorithm consists of at least three layers: one input layer, one output layer, and one hidden layer. Those layers must consist of nonlinearly activating nodes [7]. We depict a model of a multilayer perceptron in Figure 1. Note, if there is more than one hidden layer, then it can be considered a deep learning architecture. Differing from linear perceptron, MLP can distinguish data that are not linearly separable. To train the network, the backpropagation algorithm is used, which is a generalization of the least mean squares algorithm in the linear perceptron. Backpropagation is used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function [34].

2.1.2 Convolutional Neural Network

CNNs represent a type of neural networks which were first designed for 2-dimensional convolutions as it was inspired by the biological processes of animals' visual cortex [28]. From the operational perspective, CNNs are similar to ordinary neural networks (e.g., multilayer perceptron): they consist of a number of layers where each layer is made up of neurons. CNNs use three main types of layers: convolutional layers, pooling layers, and fully-connected layers. Convolutional layers are linear layers that share weights across space. Pooling layers are non-linear layers that reduce the spatial size in order to limit the number of neurons. Fully-connected layers are layers where every neuron is connected with all the neurons in the neighborhood layer. For additional information about CNNs, we refer interested readers to [12].

2.1.3 Activation Functions

An activation function of a node is a function f defining the output of a node given an input or set of inputs, see Eq. (1). To enable calculations of nontrivial functions for ANN using a small number of nodes, one needs nonlinear activation functions as follows.

$$y = Activation(\sum(weight \cdot input) + bias). \tag{1}$$

In this paper, we consider the logistic (sigmoid) function, tanh function, softmax function, and Rectified Linear Unit (ReLU) function. The logistic function is a nonlinear function giving smooth and continuously differentiable results [14]. The range of a logistic function is $[0,1]$, which means that all the values going to the next neuron will have the same sign.

$$f(x) = \frac{1}{1+e^{-x}}. \tag{2}$$

The tanh function is a scaled version of the logistic function where the main difference is that it is symmetric over the origin. The tanh function ranges in $[-1,1]$.

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1. \tag{3}$$

The softmax function is a type of sigmoid function able to map values into multiple outputs (e.g., classes). The softmax function is ideally used in the output layer of the classifier in order to obtain the probabilities defining a class for each input [5]. To denote a vector, we represent it in bold style.

$$f(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}},\ for\ j = 1,\ldots,K. \tag{4}$$

The Rectified Linear Unit (ReLU) is a nonlinear function that is differing from the previous two activation functions as it does not activate all the neurons at the same time [35]. By activating only a subset of neurons at any time, we make the network sparse and easier to compute [2]. Consequently, such properties make ReLU probably the most widely used activation function in ANNs today.

$$f(x) = max(0, x). \tag{5}$$

2.2 Side-channel Analysis

Side-channel Analysis (SCA) exploits weaknesses on the implementation level [33]. More specifically, all computations running on a certain platform result in unintentional physical leakages as a sort of physical signatures from the reaction time, power consumption, and Electromagnetic (EM) emanations released while the device is manipulating data. SCA exploits those physical signatures aiming at the key (secret data) recovery. In its basic form, SCA was proposed to perform key recovery attacks on the implementation of cryptography [23, 22]. One advantage of SCA over traditional cryptanalysis is that SCA can apply a divide-and-conquer approach. This means that SCA is typically recovering small parts of the key (sub-keys) one by one, which is reducing the attack complexity.

Based on the analysis technique used, different variants of SCA are known. In the following, we recall a few techniques used later in the paper. Although the original terms suggest power consumption as the source of leakage, the techniques apply to other side channels as well. In particular, in this work, we are using the EM side channel and the corresponding terms are adapted to reflect this.

Simple Power (or Electromagnetic) Analysis (SPA or SEMA). Simple power (or EM) analysis, as the name suggests, is the most basic form of SCA [22]. It targets information from the sensitive computation that can be recovered from a single or a few traces. As a common example, SPA can be used against a straightforward implementation of the RSA algorithm to distinguish square from multiply operation, leading to the key recovery. In this work, we apply SPA, or actually SEMA to reverse engineer the architecture of the neural network.

Differential Power (or Electromagnetic) Analysis (DPA or DEMA). DPA or DEMA is an advanced form of SCA, which applies statistical techniques to recover secret information from physical signatures. The attack normally tests for dependencies between actual physical signature (or measurements) and hypothetical physical signature, i.e., predictions on intermediate data. The hypothetical signature is based on a leakage model and key hypothesis. Small parts of the secret key (e.g., one byte) can be tested independently. The knowledge of the leakage model comes from the adversary's intuition and expertise. Some commonly used leakage models for representative devices are the Hamming weight for microcontrollers and the Hamming distance in FPGA, ASIC, and GPU [4, 31] platforms. As the measurements can be noisy, the adversary often needs many measurements, sometimes millions. Next, statistical tests like correlation [6] are applied to distinguish the correct key hypothesis from other wrong guesses. In the following, DPA (DEMA) is used to recover secret weights from a pre-trained network.

3 Side-channel Based Reverse Engineering of Neural Networks

In this section, we discuss the threat model we use, the experimental setup and reverse engineering of various elements of neural networks.

3.1 Threat Model

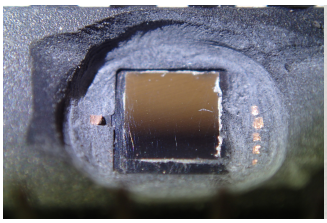
The main goal of this work is to recover the neural network architecture using only side-channel information.

Scenario. We select to work with MLP and CNNs since: 1) they are commonly used machine learning algorithms in modern applications, see e.g., [16, 11, 36, 48, 25, 21]; 2) they consist of different types of layers that are also occurring in other architectures like recurrent neural networks; and 3) in the case of MLP, the layers are all identical, which makes it more difficult for SCA and could be consequently considered as the worst-case scenario.

We choose our attack to be as generic as possible. For instance, we have no assumption on the type of inputs or its source, as we work with real numbers. If the inputs are in the form of integers (like the MNIST database), the attack becomes easier, since we would not need to recover mantissa bytes and deal with precision. We also assume that the implementation of the machine learning algorithm does not include any side-channel countermeasures.

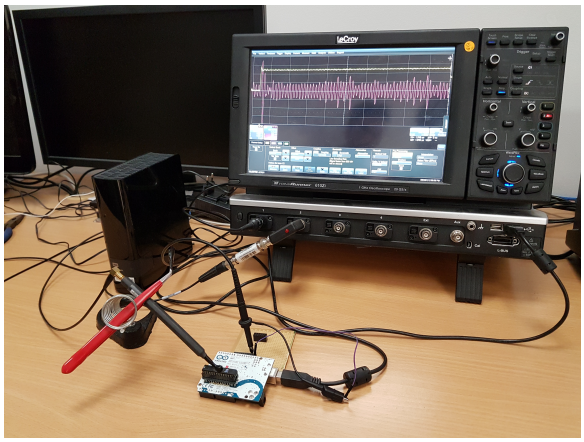
Attacker's capability. The attacker in consideration is a passive one. This implies him/her acquiring measurements of the device while operating "normally" and not interfering with its internal operations by evoking faulty computations and behavior by e.g., glitching the device, etc. More in details, we consider the following setting:

1. **Attacker does not know the architecture of the used network but can feed random (and hence known) inputs to the architecture. We note that the attacks and analysis presented in our work do not rely on any assumptions on the distributions of the inputs, although a common assumption in SCA is that they are chosen uniformly at random.** Basically, we assume that the attacker has physical access to the device (can be remote, via EM signals) and he/she knows that the device runs some neural net. The attacker only controls the execution of it through selecting the inputs, but



(a) Target 8-bit microcontroller mechanically decapsulated

(b) Langer RF-U 5-2 Near Field Electromagnetic passive Probe



(c) The complete measurement setup

Figure 2: Experimental Setup

he/she can observe the outputs and side-channel information (but not individual intermediate values). The attack scenario is often referred to as *known-plaintext attack*. An adequate use case would be when the attacker legally acquires a copy of the network with API access to it and aims at recovering its internal details e.g. for IP theft.

2. **Attacker is capable of measuring side-channel information leaked from the implementation of the targeted architecture.** The attacker can collect multiple side-channel measurements while processing the data and use different side-channel techniques for her analysis. In this work, we focus on timing and EM side channels.

3.2 Experimental Setup

Here we describe the attack methodology, which is first validated on Atmel ATmega328P. Later, we also demonstrate the proposed methodology on ARM Cortex-M3.

The side-channel activity is captured using the Lecroy WaveRunner 610zi oscilloscope. For each known input, the attacker gets one measurement (or trace) from the oscilloscope. In the following, nr. of inputs or nr. of traces are used interchangeably. Each measurement is composed of many samples (or points). The number of samples (or length of the trace) depends on sampling frequency and execution time. As shown later, depending on the target, nr. of samples can vary from thousands (for multiplication) to millions (for a whole CNN network). The measurements are synchronized with the operations by common handshaking signals like start and stop of computation. To further improve the quality of measurements, we opened the chip package mechanically (see Figure 2a). An RF-U 5-2 near-field electromagnetic (EM) probe from Langer is used to collect the EM measurements (see Figure 2b). The setup is depicted in Figure 2c. We use the probe as an antenna for spying on the EM side-channel leakage from the underlying processor running ML. Note that EM measurements also allow to observe the timing of all the operations and thus the setup allows for timing side-channels analysis as well. Our choice of the target platform is motivated by the following considerations:

- Atmel ATmega328P: This processor typically allows for high quality measurements. We are able to achieve a high signal-to-noise ratio (SNR) measurements, making this a perfect tuning phase to develop the methodology of our attacks.
- ARM Cortex-M3: This is a modern 32-bit microcontroller architecture featuring multiple stages of the pipeline, on-chip co-processors, low SNR measurements, and wide application. We show that the developed methodology is indeed versatile across targets with a relevant update of measurement capability.

In addition, real-world use cases also justify our platforms of choice. Similar micro-controllers are often used in wearables like Fitbit (ARM Cortex-M4), several hardware crypto wallets, smart home devices, etc. Additionally, SCA on a GPU or an FPGA platform is practically demonstrated in several instances, thus our methodology can be directly adapted for those cases as well. For different platforms, the leakage model could change, but this would not limit our approach and methodology. In fact, adequate leakage models are known for platforms like FPGA [4] and GPU [31]. Moreover, as for ARM Cortex-M3, low SNR of the measurement might force the adversary to increase the number of measurements and apply signal pre-processing techniques, but the main principles behind the analysis remain valid.

As already stated above, the exploited leakage model of the target device is the Hamming weight (HW) model. A microcontroller loads sensitive data to a data bus to perform indicated instructions. This data bus is pre-charged to all '0's' before every instruction. Note that data bus being pre-charged is a natural behavior of microcontrollers and not a vulnerability introduced by the attacker. Thus, the power consumption (or EM radiation) assigned to the value of the data being loaded is modeled as the number of bits equal to '1'. In other words, the power consumption of loading data

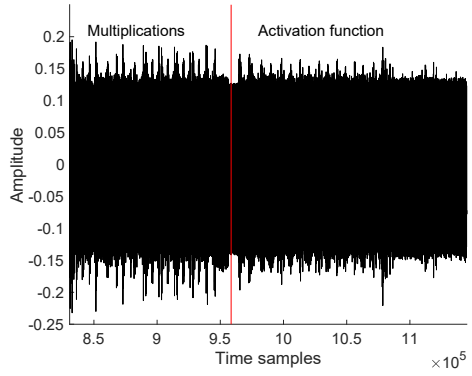


Figure 3: Observing pattern and timing of multiplication and activation function

x is:

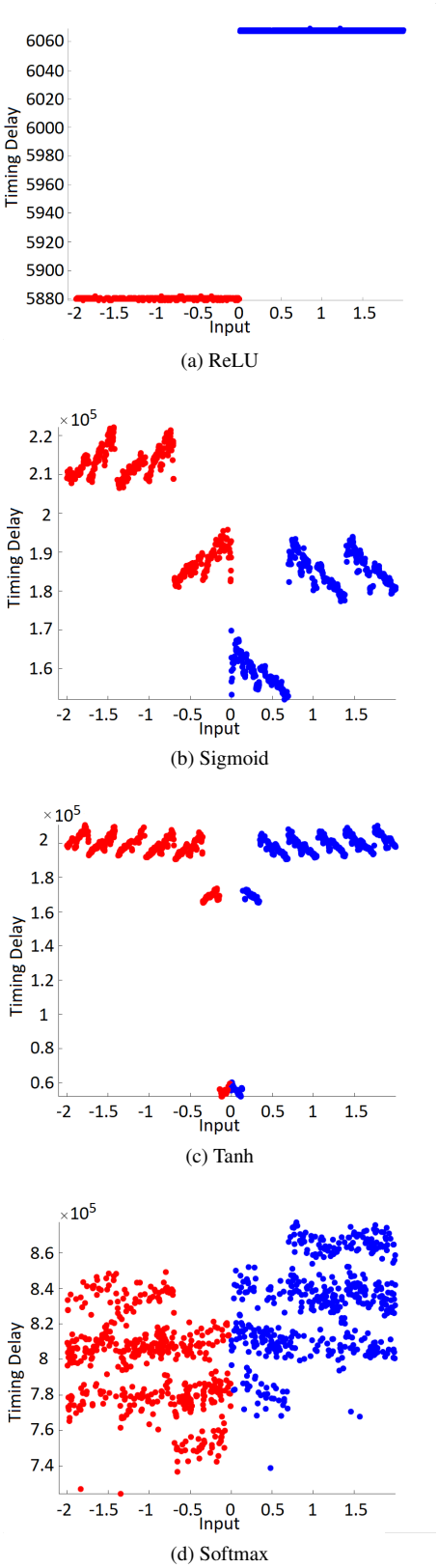
$$HW(x) = \sum_{i=1}^{n} x_i \ , \tag{6}$$

where x_i represents the i^{th} bit of x. In our case, it is the secret pre-trained weight which is regularly loaded from memory for processing and results in the HW leakage. To conduct the side-channel analysis, we perform the divide-and-conquer approach, where we target each operation separately. The full recovery process is described in Section 3.6.

Several pre-trained networks are implemented on the board. The training phase is conducted offline, and the trained network is then implemented in C language and compiled on the microcontroller. In these experiments, we consider multilayer perceptron architectures consisting of a different number of layers and nodes in those layers. Note that, with our approach, there is no limit in the number of layers or nodes we can attack, as the attack scales linearly with the size of the network. The methodology is developed to demonstrate that the key parameters of the network, namely the weights and activation functions can be reverse engineered. Further experiments are conducted on deep neural networks with three hidden layers but the method remains valid for larger networks as well.

3.3 Reverse Engineering the Activation Function

We remind the reader that nonlinear activation functions are necessary in order to represent nonlinear functions with a small number of nodes in a network. As such, they are elements used in virtually any neural network architecture today [25, 15]. If the attacker is able to deduce the information on the type of used activation functions, he/she can use that knowledge together with information about input values to deduce the behavior of the whole network.



(a) ReLU

(b) Sigmoid

(c) Tanh

(d) Softmax

Figure 4: Timing behavior for different activation functions

Table 1: Minimum, Maximum, and Mean computation time (in *ns*) for different activation functions

Activation Function	Minimum	Maximum	Mean
ReLU	5 879	6 069	5 975
Sigmoid	152 155	222 102	189 144
Tanh	51 909	210 663	184 864
Softmax	724 366	877 194	813 712

We analyze the side-channel leakage from different activation functions. We consider the most commonly used activation functions, namely ReLU, sigmoid, tanh, and softmax [14, 35]. The timing behavior can be observed directly on the EM trace. For instance, as shown later in Figure 8a, a multiplication is followed by activation with individual signatures. For a similar architecture, we test different variants with each activation function. We collect EM traces and measure the timing of the activation function computation from the measurements. The measurements are taken when the network is processing random inputs in the range, i.e., $x \in \{-2,2\}$. A total of 2000 EM measurements are captured for each activation function. As shown in Figure 3, the timing behavior of the four tested activation functions have distinct signatures allowing easy characterization.

Different inputs result in different processing times. Moreover, the timing behavior for the same inputs largely varies depending on the activation function. For example, we can observe that ReLU will require the shortest amount of time, due to its simplicity (see Figure 4a). On the other hand, tanh and sigmoid might have similar timing delays, but with different pattern considering the input (see Figure 4b and Figure 4b), where tanh is more symmetric in pattern compared to sigmoid, for both positive and negative inputs. We can observe that softmax function will require most of the processing time, since it requires the exponentiation operation which also depends on the number of neurons in the output layer. As neural network algorithms are often optimized for performance, the presence of such timing side-channels is often ignored. A function such as tanh or sigmoid requires computation of e^x and division and it is known that such functions are difficult to implement in constant time. In addition, constant time implementations might lead to substantial performance degradation. Other activation functions can be characterized similarly. Table 1 presents the minimum, maximum, and mean computation time for each activation function over 2000 captured measurements. While ReLU is the fastest one, the timing difference for other functions stands out sufficiently, to allow for a straightforward recovery. To distinguish them, one can also do some pattern matching to determine which type of function is used, if necessary. Note, although Sigmoid and Tanh have similar Maximum and mean values, the Minimum value differs significantly. Moreover, the attacker can sometimes pre-characterize (or profile) the timing behavior of the target activation function independently for better precision, especially when common libraries are used for standard functions like multiplication, activation function, etc.

3.4 Reverse Engineering the Multiplication Operation

A well-trained network can be of significant value. Main distinguishing factors for a well trained network against a poorly trained one, for a given architecture, are the weights. With fine-tuned weights, we can improve the accuracy of the network. In the following, we demonstrate a way to recover those weights by using SCA.

For the recovery of the weights, we use the Correlation Power Analysis (CPA) i.e., a variant of DPA using the Pearson's correlation as a statistical test.[1] CPA targets the multiplication $m = x \cdot w$ of a known input x with a secret weight w. Using the HW model, the adversary correlates the activity of the predicted output m for all hypothesis of the weight. Thus, the attack computes $\rho(t,w)$, for all hypothesis of the weight w, where ρ is the Pearson correlation coefficient and t is the side-channel measurement. The correct value of the weight w will result in a higher correlation standing out from all other wrong hypotheses w^*, given enough measurements. Although the attack concept is the same as when attacking cryptographic algorithms, the actual attack used here is quite different. Namely, while cryptographic operations are always performed on fixed length integers, in ANN we are dealing with real numbers.

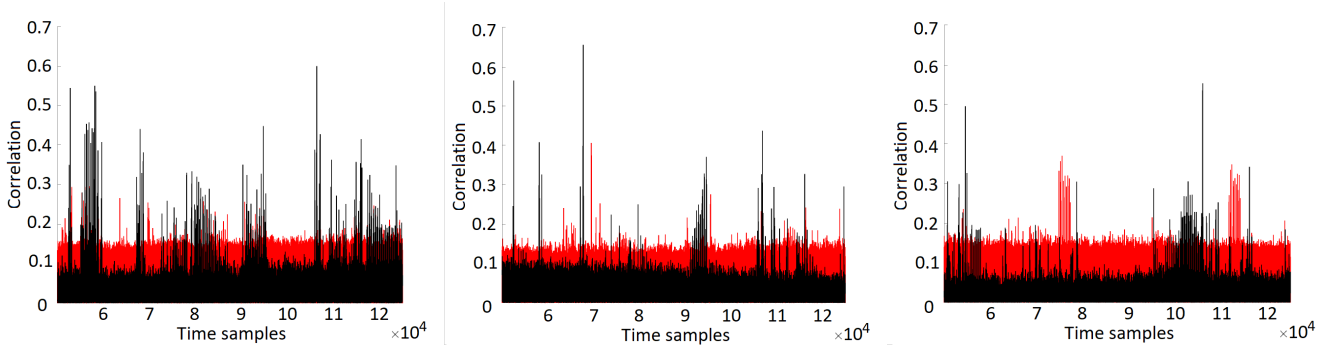
We start by analyzing the way the compiler is handling floating-point operations for our target. The generated assembly is shown in Table 2, which confirms the usage of IEEE 754 compatible representation as stated above. The knowledge of the representation allows one to better estimate the leakage behavior. Since the target device is an 8-bit microcontroller, the representation follows a 32-bit pattern $(b_{31}...b_0)$, being stored in 4 registers. The 32-bit consist of: 1 sign bit (b_{31}), 8 biased exponent bits $(b_{30}...b_{23})$ and 23 mantissa (fractional) bits $(b_{22}...b_0)$. It can be formulated as:

$$(-1)^{b_{31}} \times 2^{(b_{30}...b_{23})_2 - 127} \times (1.b_{22}...b_0)_2.$$

For example, the value 2.43 can be expressed as $(-1)^0 \times 2^{(1000000)_2 - 127} \times (1.00110111000010100011111)_2$. The measurement t is considered when the computed result m is stored back to the memory, leaking in the HW model i.e., $HW(m)$. Since 32-bit m is split into individual 8-bits, each byte of m is recovered individually. Hence, by recovering this representation, it is enough to recover the estimation of the real number value.

To implement the attack two different approaches can be considered. The first approach is to build the hypothesis on

[1] It is called CEMA in case of EM side channel.



(a) First byte mantissa for weight = 2.43 (b) Second byte mantissa for weight = 2.43 (c) Third byte mantissa for weight = 2.43

Figure 5: Correlation of different weights candidate on multiplication operation

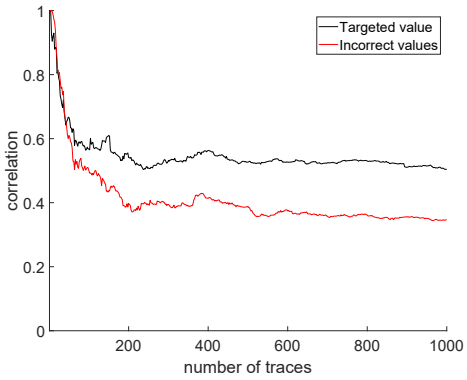
Table 2: Code snippet of the returned assembly for multiplication: $x = x \cdot w (= 2.36$ or 0x3D0A1740 in IEEE 754 representation). The multiplication itself is not shown here, but from the registers assignment, our leakage model assumption holds.

#	Instruction	Comment
11a	ldd r22, Y+1	0x01
11c	ldd r23, Y+2	0x02
11e	ldd r24, Y+3	0x03
120	ldd r25, Y+4	0x04
122	ldi r18, 0x3D	61
124	ldi r19, 0x0A	10
126	ldi r20, 0x17	23
128	ldi r21, 0x40	64
12a	call 0xa0a	multiplication
12e	std Y+1, r22	0x01
130	std Y+2, r23	0x02
132	std Y+3, r24	0x03
134	std Y+4, r25	0x04

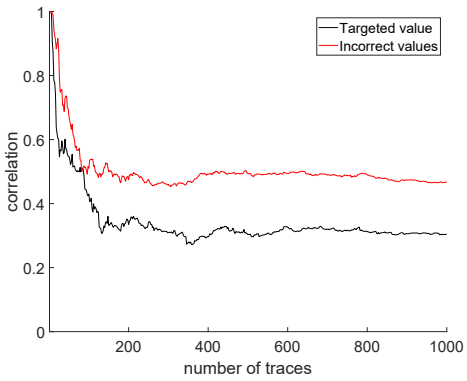
the weight directly. For this experiment, we target the result of the multiplication m of known input values x and unknown weight w. For every input, we assume different possibilities for weight values. We then perform the multiplication and estimate the IEEE 754 binary representation of the output. To deal with the growing number of possible candidates for the unknown weight w, we assume that the weight will be bounded in a range $[-N,N]$, where N is a parameter chosen by the adversary, and the size of possible candidates is denoted as $s = 2N/p$, where p is the precision when dealing with floating-point numbers.

Then, we perform the recovery of the 23-bit mantissa of the weight. The sign and exponent could be recovered separately. Thus, we are observing the leakage of 3 registers, and based on the best CPA results for each register, we can reconstruct the mantissa. Note that the recovered mantissa does not directly relate to the weight, but with the recovery of the sign and exponent, we could obtain the unique weight value. The traces are measured when the microcontroller performs secret weight multiplication with uniformly random values between -1 and 1 ($x \in \{-1,1\}$) to emulate normalized input values. We set $N = 5$ and to reduce the number of possible candidates, we assume that each floating-point value will have a precision of 2 decimal points, $p = 0.01$. Since we are dealing with mantissa only, we can then only check the weight candidates in the range $[0,N]$, thus reducing the number of possible candidates. We note here that this range $[-5,5]$ is based on the previous experiments with MLP. Although, in the later phase of the experiment, we targeted the floating point and fixed-point representation (2^{32} in the worst case scenario on a 32-bit microcontroller, but could be less if the value is for example normalized), instead of the real value, which could in principle cover all possible floating values.

In Figure 5, we show the result of the correlation for each byte with the measured traces. The horizontal axis shows the time of execution and vertical axis correlation. The experiments were conducted on 1 000 traces for each case. In the figure, the black plot denotes the correlation of the "correct" mantissa weight ($|m(\hat{w}) - m(w)| < 0.01$), whereas the red plots are from all other weight candidates in the range described earlier. Since we are only attacking mantissa in this phase, several weight candidates might have similar correlation peaks. After the recovery of the mantissa, the sign bit and exponent can be recovered similarly, which narrows down the list candidate to a unique weight. Another observation is that the correlation value is not very high and scattered across different clock cycles. This is due to the reason that the measurements are noisy and since the operation is not constant-time, the interesting time samples are distributed across multiple clock cycles. Nevertheless, it is shown that the side-channel leakage can be exploited to recover the weight up to certain precision. Multivariate side channel analysis [42] can be considered if distributed samples hinder recovery.

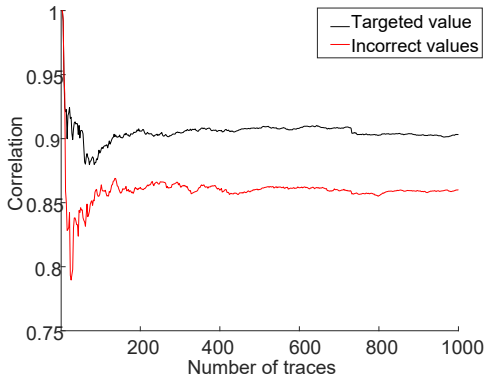


(a) weight = 1.635

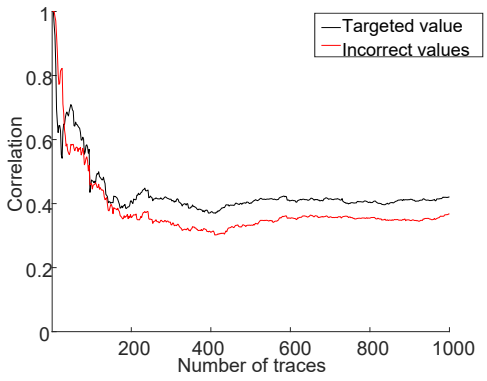


(b) weight = 0.890

Figure 6: Correlation comparison between the correct and incorrect mantissa of the weights. (a) Correct mantissa can be recovered (correct values/black line has a higher value compared to max incorrect values/red line). (b) A special case where the incorrect value of mantissa has a higher correlation, recovering 0.896025 (1100100000..00) instead of 0.89 (1100011110...10), still within precision error limits resulting in attack success



(a) First byte recovery (sign and 7-bit exponent)



(b) Second byte recovery (lsb exponent and mantissa)

Figure 7: Recovery of the weight

We emphasize that attacking real numbers as in the case of weights of ANN can be easier than attacking cryptographic implementations. This is because cryptography typically works on fixed-length integers and exact values must be recovered. When attacking real numbers, small precision errors due to rounding off the intermediate values still result in useful information.

To deal with more precise values, we can target the mantissa multiplication operation directly. In this case, the search space can either be $[0, 2^{23} - 1]$ to cover all possible values for the mantissa (hence, more computational resources will be required) or we can focus only on the most significant bits of the mantissa (lesser candidates but also with lesser precision). Since the 7 most significant bits of the mantissa are processed in the same register, we can aim to target only those bits, assigning the rest to 0. Thus, our search space is now $[0, 2^7 - 1]$. The mantissa multiplication can be performed as $1.mantissa_x \times 1.mantissa_w$, then taking the 23 most significant bits after the leading 1, and normalization (updating the exponent if the result overflows) if necessary.

In Figure 6, we show the result of the correlation between the HW of the first 7-bit mantissa of the weight with the traces. Except for Figure 6b, the other results show that the correct mantissa can be recovered. Although the correlation is not increasing, it is important that the difference becomes stable after a sufficient amount of traces is used and eventually distinguishing correct weight from wrong weight hypotheses. The most interesting result is shown in Figure 6b, which at first glance looks like a failure of the attack. Here, the target value of the mantissa is **1100011**110...10, while the attack recovers **1100100**000..00. Considering the sign and exponents, the attack recovers *0.890625* instead of *0.89*, i.e., a precision error at 4^{th} place after decimal point. Thus, in both cases, we have shown that we can recover the weights from the SCA leakage.

In Figure 7, we show the composite recovery of 2 bytes of the weight representation i.e., a low precision setting where

we recover sign, exponent, and most significant part of mantissa. Again, the targeted (correct) weight can be easily distinguished from the other candidates. Hence, once all the necessary information has been recovered, the weight can be reconstructed accordingly.

3.5 Reverse Engineering the Number of Neurons and Layers

After the recovery of the weights and the activation functions, now we use SCA to determine the structure of the network. Mainly, we are interested to see if we can recover the number of hidden layers and the number of neurons for each layer. To perform the reverse engineering of the network structure, we first use SPA (SEMA). SPA is the simplest form of SCA which allows information recovery in a single (or a few) traces with methods as simple as a visual inspection. The analysis is performed on three networks with different layouts.

The first analyzed network is an MLP with one hidden layer with 6 neurons. The EM trace corresponding to the processing of a randomly chosen input is shown in Figure 8a. By looking at the EM trace, the number of neurons can be easily counted. The observability arises from the fact that multiplication operation and the activation function (in this case, it is the Sigmoid function) have completely different leakage signatures. Similarly, the structures of deeper networks are also shown in Figure 8b and Figure 8c. The recovery of output layer then provides information on the number of output classes. However, distinguishing different layers might be difficult, since the leakage pattern is only dependent on multiplication and activation function, which are usually present in most of the layers. We observe minor features allowing identification of layer boundaries but only with low confidence. Hence, we develop a different approach based on CPA to identify layer boundaries.

The experiments follow a similar methodology as in the previous experiments. To determine if the targeted neuron is in the same layer as previously attacked neurons, or in the next layer, we perform a weight recovery using two sets of data.

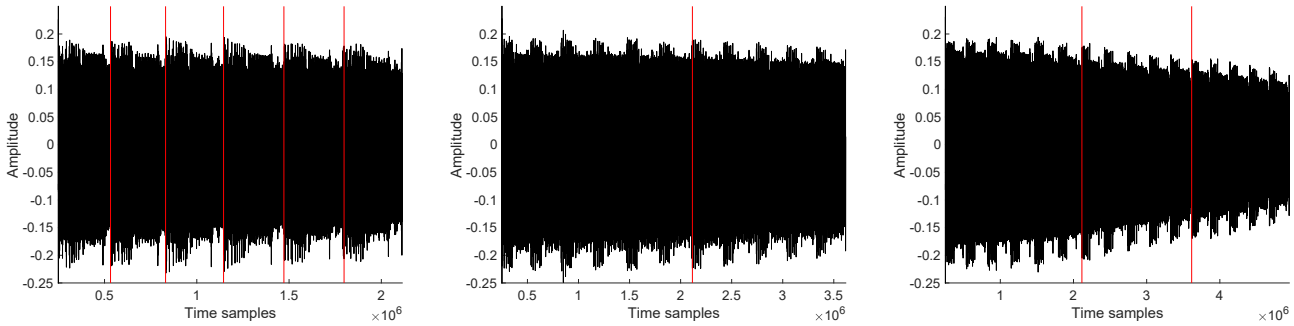
Let us assume that we are targeting the first hidden layer (the same approach can be done on different layers as well). Assume that the input is a vector of length N_0, so the input x can be represented $x = \{x_1, ..., x_{N_0}\}$. For the targeted neuron y_n in the hidden layer, perform the weight recovery on 2 different hypotheses. For the first hypothesis, assume that the y_n is in the first hidden layer. Perform weight recovery individually using x_i, for $1 \leq i \leq N_0$. For the second hypothesis, assume that y_n is in the next hidden layer (the second hidden layer). Perform weight recovery individually using y_i, for $1 \leq i \leq (n-i)$. For each hypothesis, record the maximum (absolute) correlation value, and compare both. Since the correlation depends on both inputs to the multiplication operation, the incorrect hypothesis will result in a lower correlation value. Thus, this can be used to identify layer boundaries.

3.6 Recovery of the Full Network Layout

The combination of previously developed individual techniques can thereafter result in full reverse engineering of the network. The full network recovery is performed layer by layer, and for each layer, the weights for each neuron have to be recovered one at a time. Let us consider a network consisting of N layers, $L_0, L_1, ..., L_{N-1}$, with L_0 being the input layer and L_{N-1} being the output layer. Reverse engineering is performed with the following steps:

1. The first step is to recover the weight w_{L_0} of each connection from the input layer (L_0) and the first hidden layer (L_1). Since the dimension of the input layer is known, the CPA/CEMA can be performed n_{L_0} times (the size of L_0). The correlation is computed for 2^d hypotheses (d is the number of bits in IEEE 754 representation, normally it is 32 bits, but to simplify, 16 bits can be used with lesser precision for the mantissa). After the weights have been recovered, the output of the sum of multiplication can be calculated. This information provides us with input to the activation function.
2. In order to determine the output of the sum of the multiplications, the number of neurons in the layer must be known. This can be recovered by the combination of SPA/SEMA and DPA/DEMA technique described in the previous subsection (2 times CPA for each weight candidate w, so in total $2n_{L_0}2^d$ CPA required), in parallel with the weight recovery. When all the weights of the first hidden layer are recovered, the following steps are executed.
3. Using the same set of traces, timing patterns for different inputs to the activation function can be built, similar to Figure 4. Timing patterns or average timing can then be compared with the profile of each function to determine the activation function (a comparison can be based on simple statistical tools like correlation, distance metric, etc). Afterward, the output of the activation function can be computed, which provides the input to the next layer.
4. The same steps are repeated in the subsequent layers ($L_1, ..., L_{N-1}$, so in total at most $2Nn_L2^d$, where n_L is $max(n_{L_0}, ..., n_{L_{N-1}})$) until the structure of the full network is recovered.

The whole procedure is depicted in Figure 9. In general, it can be seen that the attack scales linearly with the size of the network. Moreover, the same set of traces can be reused for various steps of the attack and attacking different layers, thus reducing measurement effort.



(a) One hidden layer with 6 neurons (b) 2 hidden layers (6 and 5 neurons each) (c) 3 hidden layers (6,5,5 neurons each)

Figure 8: SEMA on hidden layers

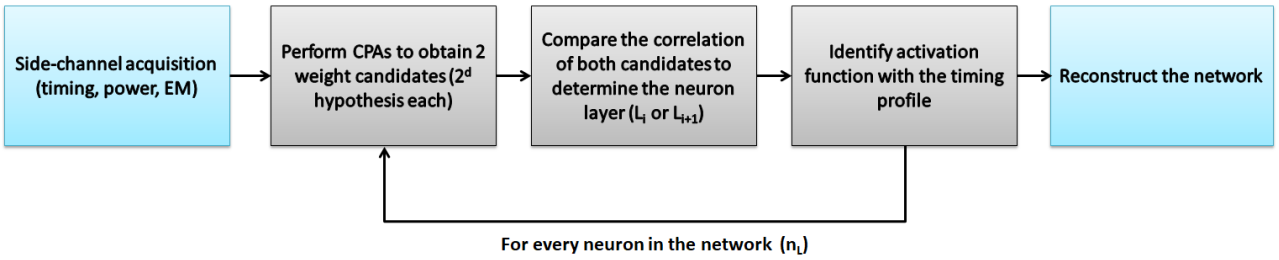


Figure 9: Methodology to reverse engineer the target neural network

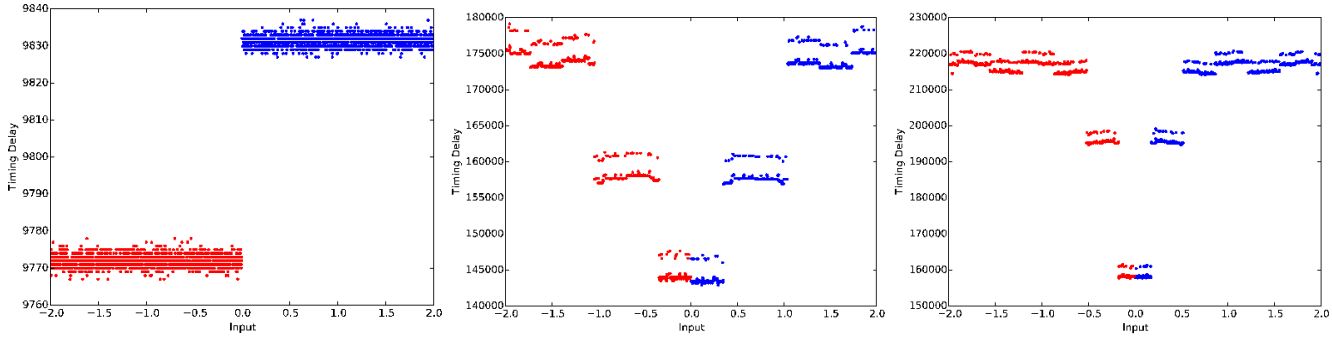
4 Experiments with ARM Cortex-M3

In the previous section, we propose a methodology to reverse engineer sensitive parameters of a neural network, which we practically validated on an 8-bit AVR (Atmel ATmega328P). In this section, we extend the presented attack on a 32-bit ARM microcontroller. ARM microcontrollers form a fair share of the current market with huge dominance in mobile applications, but also seeing rapid adoption in markets like IoT, automotive, virtual and augmented reality, etc. Our target platform is the widely available *Arduino due* development board which contains an *Atmel SAM3X8E ARM Cortex-M3* CPU with a 3-stage pipeline, operating at 84 MHz. The measurement setup is similar to previous experiments (Lecroy WaveRunner 610zi oscilloscope and RF-U 5-2 near-field EM probe from Langer). The point of measurements was determined by a benchmarking code running AES encryption. After capturing the measurements for the target neural network, one can perform reverse engineering. Note that ARM Cortex-M3 (as well as M4 and M7) have support for deep learning in the form of CMSIS-NN implementation [27].

The timing behavior of various activation functions is shown in Figure 10. The results, though different from previous experiments on AVR, have unique timing signatures, allowing identification of each activation function. Here, sigmoid and tanh activation functions have similar minimal computation time but the average and maximum values are higher for tanh function. To distinguish, one can obtain multiple inputs to the function, build patterns and do pattern matching to determine which type of function is used. The activity of a single neuron is shown in Figure 11a, which uses sigmoid as an activation function (the multiplication operation is shown separated by a vertical red line).

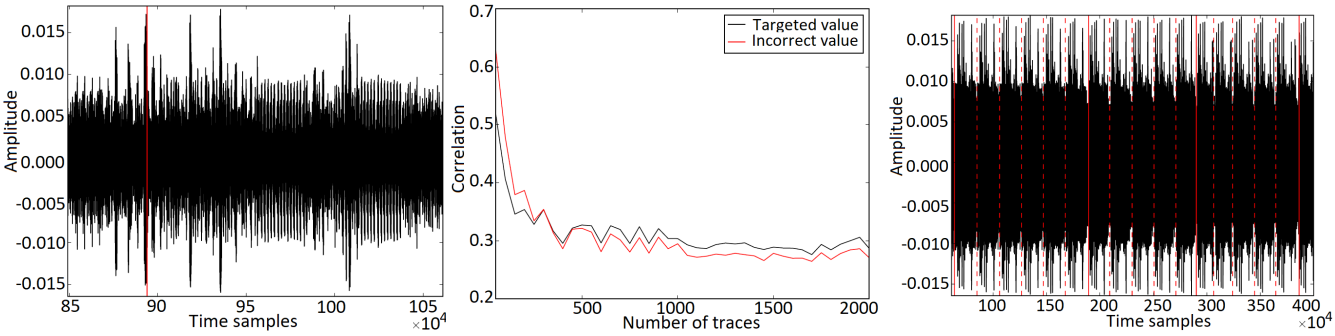
A known input attack is mounted on the multiplication to recover the secret weight. One practical consideration in attacking multiplication is that different compilers will compile it differently for different targets. Modern microcontrollers also have dedicated floating point units for handling operations like multiplication of real numbers. To avoid the discrepancy of the difference of multiplication operation, we target the output of multiplication. In other words, we target the point when multiplication operation with secret weight is completed and the resultant product is updated in general purpose registers or memory. Figure 11b shows the success of attack recovering secret weight of 2.453, with a known input. As stated before, side-channel measurements on modern 32-bit ARM Cortex-M3 may have lower SNR thus making attack slightly harder. Still, the attack is shown to be practical even on ARM with 2× more measurements. In our setup, getting 200 extra measurements takes less than a minute. Similarly, the setup and number of measurements can be updated for other targets like FPGA, GPU, etc.

Finally, the full network layout is recovered. The activity



(a) ReLU (b) Sigmoid (c) Tanh

Figure 10: Timing behavior for different activation functions



(a) Observing pattern and timing of multiplication and activation function (b) Correlation comparison between correct and incorrect mantissa for weight=2.453 (c) SEMA on hidden layers with 3 hidden layers (6,5,5 neurons each)

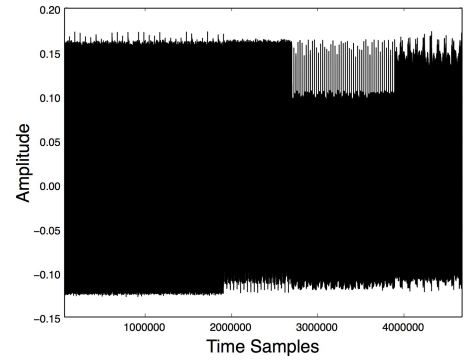
Figure 11: Analysis of an (6,5,5,) neural network

of a full network with 3 hidden layers composed of 6, 5, and 5 neurons each is shown in Figure 11c. All the neurons are observable by visual inspection. The determination of layer boundaries (shown by a solid red line) can be determined by attacking the multiplication operation and following the approach discussed in Section 3.6.

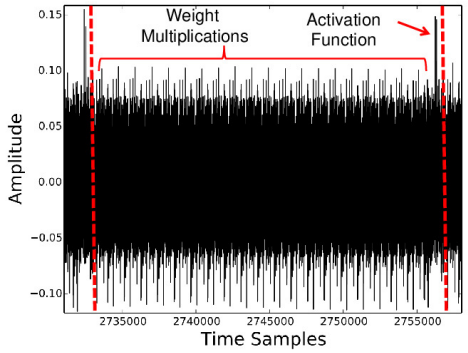
4.1 Reverse Engineering MLP

The migration of our testbed to ARM Cortex-M3 allowed us to test bigger networks, which are used in some relevant case-studies. First, we consider an MLP that is used in profiling side-channel analysis [41]. Our network of choice comes from the domain of side-channel analysis which has seen the adoption of deep learning methods in the past. With this network, a state-of-the-art profiled SCA was conducted when considering several datasets where some even contain implemented countermeasures. Since the certification labs use machine learning to evaluate the resilience of cryptographic implementations to profiled attacks, an attacker being able to reverse engineer that machine learning would be able to use it to attack implementations on his own. The MLP we investigate has 4 hidden layers with dimensions $(50, 30, 20, 50)$, it uses ReLU activation function and has Softmax at the output. The whole measurement trace is shown in Figure 12(a) with a zoom on 1 neurons in the third layer in Figure 12(b). When measuring at 500 $MSamples/s$, each trace had ~ 4.6 million samples. The dataset is DPAcontest v4 with 50 samples and 75 000 measurements [46]. The first 50 000 measurements are used for training and the rest for testing. We experiment with the Hamming weight model (meaning there are 9 output classes). The original accuracy equals 60.9% and the accuracy of the reverse engineered network is 60.87%. While the previously developed techniques are directly available, there are a few practical issues.

- As the average run time is $9.8ms$, each measurement would take long considering the measurement and data saving time. To boost up the SNR, averaging is recommended. We could use the oscilloscope in-built feature for averaging. Overall, the measurement time per trace was slightly over one second after averaging 10 times.
- The measurement period was too big to measure the whole period easily at a reasonable resolution. This was resolved by measuring two consecutive layers at a time



(a)



(b)

Figure 12: (a) Full EM trace of the MLP network from [41], (b) zoom on one neuron in the third hidden layer showing 20 multiplications, followed by a ReLU activation function. 50 such patterns can be seen in (a) identifying third layer in (50,30,20,50) MLP

in independent measurements. It is important to always measure two consecutive layers and not individual layer to determine layer boundaries. This issue otherwise can be solved with a high-end oscilloscope.

- We had to resynchronize traces each time according to the target neuron which is a standard pre-processing in side-channel attacks.

Next, we experiment with an MLP consisting of 4 hidden layers, where each layer has 200 nodes. We use the MNIST database as input to the MLP [29]. The MNIST database contains 60 000 training images and 10 000 testing images where each image has 28×28 pixel size. The number of classes equals 10. The accuracy of the original network is equal to 98.16% while the accuracy of the reverse engineered network equals 98.15%, with an average weight error converging to 0.0025.

We emphasize that both attacks (on DPAcontest v4 and MNIST) were performed following exactly the same procedure as in previous sections leading to a successful recovery of the network parameters. Finally, in accordance with the conclusions that our attack scales linearly with the size of the network, we did not experience additional difficulties when compared to attacking smaller networks.

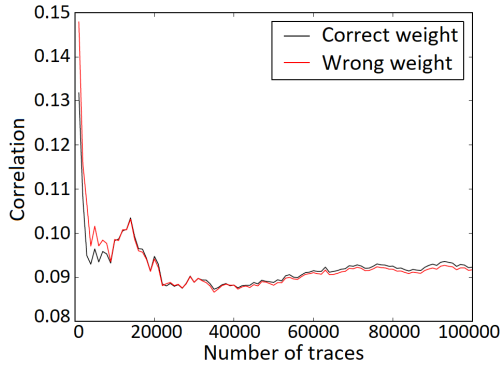
4.2 Reverse Engineering CNN

When considering CNN, the target is the CMSIS-NN implementation [27] on ARM Cortex-M3 with measurement setup same as in previous experiments. Here, as input, we target the CIFAR-10 dataset [24]. This dataset consists of 60 000 32×32 color images in 10 classes. Each class has 6 000 images and there are in total 50 000 training images and 10 000 test images. The CNN we investigate is the same as in [27] and it consists of 3 convolutional layers, 3 max pooling layers, and one fully-connected layer (in total 7 layers).

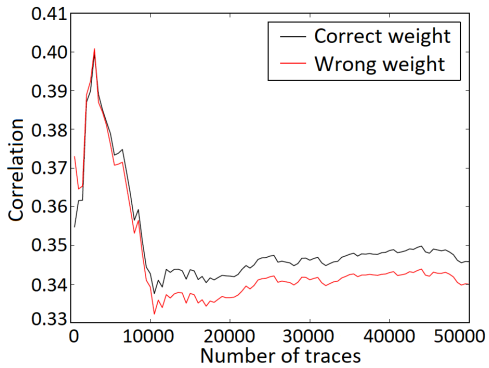
We choose as target the multiplication operation from the input with the weight, similar as in previous experiments. Differing from previous experiments, the operations on real values are here performed using fixed-point arithmetic. Nevertheless, the idea of the attack remains the same. In this example, numbers are stored using 8-bit data type – `int8` (q7). The resulting multiplication is stored in temporary `int` variable. This can also be easily extended to `int16` or `int32` for more precision. Since we are working with integer values, we use the Hamming weight model of the hypothetical outputs (since the Hamming weight model is more straightforward in this case).

If the storing of temporary variable is targeted, as can be seen from Figure 13(a), around 50 000 traces will be required before the correct weight can be distinguished from the wrong weights. This is based on 0.01 precision (the absolute difference from the actual weight in floating number). However, in this case, it can be observed that the correlation value is quite low (~ 0.1). In the case that the conversion to `int8` is performed after the multiplication, this can be also targeted. In Figure 13(b), it can be seen that after 10 000 traces, the correct weight candidate can be distinguished, and the correlation is slightly higher (~ 0.34).

Next, for pooling layer, once the weights in the convolution part are recovered, the output can be calculated. Most CNNs use max pooling layers, which makes it also possible to simply guess the pooling layer type. Still, because the max pooling layer is based on the following conditional instruction, $conditional(if(a > max)max = a)$, it is straightforward to differentiate it from the average pooling that has summation and division operations. This technique is then repeated to reverse engineer any number of convolutional and pooling layers. Finally, the CNN considered here uses ReLU activation function and has one fully-connected layer, which are reverse engineered as discussed in previous sections. In our experiment, the original accuracy of the CNN equals 78.47% and the accuracy of the recovered CNN is 78.11%. As it can be seen, by using sufficient measurements (e.g., $\sim$ 50 000), we are able to reverse engineer CNN architecture as well.



(a) `int` scenario



(b) `int8` scenario

Figure 13: The correlation of correct and wrong weight hypotheses on different number of traces targeting the result of multiplication operation stored as different variable type: (a) `int`, (b) `int8`

5 Mitigation

As demonstrated, various side-channel attacks can be applied to reverse engineer certain components of a pre-trained network. To mitigate such a recovery, several countermeasures can be deployed:

1. Hidden layers of an MLP must be executed in sequence but the multiplication operation in individual neurons within a layer can be executed independently. An example is shuffling [50] as a well-studied side-channel countermeasure. It involves shuffling/permuting the order of execution of independent sub-operations. For example, given N sub-operations $(1,\ldots,N)$ and a random permutation σ, the order of execution becomes $(\sigma(1),\ldots,\sigma(N))$ instead. We propose to shuffle the order of multiplications of individual neurons within a hidden layer during every classification step. Shuffling modifies the time window of operations from one execution to another, mitigating a classical DPA/DEMA attack.
2. Weight recovery can benefit from the application of masking countermeasures [8, 42]. Masking is another widely studied side-channel countermeasure that is even accompanied by a formal proof of security. It involves assuring that sensitive computations are with random values to remove the dependencies between actual data and side-channel signatures, thus preventing the attack. Every computation of $f(x,w)$ is transformed into $f_m(x\oplus m_1, w\oplus m_2) = f(x,w)\oplus m$, where m_1, m_2 are uniformly drawn random masks, and f_m is the masked function which applies mask m at the output of f, given masked inputs $x\oplus m_1$ and $w\oplus m_2$. If each neuron is individually masked with an independently drawn uniformly random mask for every iteration and every neuron, the proposed attacks can be prevented. However, this might result in a substantial performance penalty.
3. The proposed attack on activation functions is possible due to the non-constant timing behavior. Mostly considered activation functions perform exponentiation operation. Implementation of constant time exponentiation has been widely studied in the domain of public key cryptography [13]. Such ideas can be adjusted to implement constant time activation function processing.

Note, the techniques we discuss here represent well-explored methods of protecting against side-channel attacks. As such, they are generic and can be applied to any implementation. Unfortunately, all those countermeasures also come with an area and performance cost. Shuffling and masking require a true random number generator that is typically very expensive in terms of area and performance. Constant time implementations of exponentiation [1] also come at performance efficiency degradation. Thus, the optimal choice of protection mechanism should be done after a systematic resource and performance evaluation study.

6 Further Discussions and Conclusions

Neural networks are widely used machine learning family of algorithms due to its versatility across domains. Their effectiveness depends on the chosen architecture and fine-tuned parameters along with the trained weights, which can be proprietary information. In this work, we practically demonstrate reverse engineering of neural networks using side-channel analysis techniques. Concrete attacks are performed on measured data corresponding to implementations of chosen networks. To make our setting even more general, we do not assume any specific form of the input data (except that inputs are real values).

We conclude that using an appropriate combination of SEMA and DEMA techniques, all sensitive parameters of the network can be recovered. The proposed methodology is demonstrated on two different modern controllers, a classic 8-bit AVR and a 32-bit ARM Cortex-M3 microcontroller. As also shown in this work, the attacks on modern devices are

typically somewhat harder to mount, due to lower SNR for side-channel attacks, but remain practical. In the presented experiments, the attack took twice as many measurements, requiring roughly 20 seconds extra time. Overall, the attack methodology scales linearly with the size of the network. The attack might be easier in some setting where a new network is derived from well known network like VGG-16, Alexnet, etc. by tuning hyper-parameters or transfer learning. In such cases, the side-channel based approach can reveal the remaining secrets. However, analysis of such partial cases is currently out of scope.

The proposed attacks are both generic in nature and more powerful than the previous works in this direction. Finally, suggestions on countermeasures are provided to help the designer mitigate such threats. The proposed countermeasures are borrowed mainly from side-channel literature and can incur huge overheads. Still, we believe that they could motivate further research on optimized and effective countermeasures for neural networks. Besides continuing working on countermeasures, as the main future research goal, we plan to look into more complex CNNs. Naturally, this will require stepping aside from low power ARM devices and using for instance, FPGAs. Additionally, in this work, we considered only feed-forward networks. It would be interesting to extend our work to other types of networks like recurrent neural networks. Since such architectures have many same elements like MLP and CNNs, we believe our attack should be (relatively) easily extendable to such neural networks.

References

[1] AL HASIB, A., AND HAQUE, A. A. M. M. A comparative study of the performance and security issues of AES and RSA cryptography. In *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on* (2008), vol. 2, IEEE, pp. 505–510.

[2] ALBERICIO, J., JUDD, P., HETHERINGTON, T., AAMODT, T., JERGER, N. E., AND MOSHOVOS, A. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)* (June 2016), pp. 1–13.

[3] ATENIESE, G., MANCINI, L. V., SPOGNARDI, A., VILLANI, A., VITALI, D., AND FELICI, G. Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers. *Int. J. Secur. Netw. 10*, 3 (Sept. 2015), 137–150.

[4] BHASIN, S., GUILLEY, S., HEUSER, A., AND DANGER, J.-L. From cryptography to hardware: analyzing and protecting embedded Xilinx BRAM for cryptographic applications. *Journal of Cryptographic Engineering 3*, 4 (2013), 213–225.

[5] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[6] BRIER, E., CLAVIER, C., AND OLIVIER, F. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2004), Springer, pp. 16–29.

[7] COLLOBERT, R., AND BENGIO, S. Links Between Perceptrons, MLPs and SVMs. In *Proceedings of the Twenty-first International Conference on Machine Learning* (New York, NY, USA, 2004), ICML '04, ACM, pp. 23–.

[8] CORON, J.-S., AND GOUBIN, L. On boolean and arithmetic masking against differential power analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2000), Springer, pp. 231–237.

[9] DOWLIN, N., GILAD-BACHRACH, R., LAINE, K., LAUTER, K., NAEHRIG, M., AND WERNSING, J. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (2016), ICML'16, JMLR.org, pp. 201–210.

[10] FREDRIKSON, M., LANTZ, E., JHA, S., LIN, S., PAGE, D., AND RISTENPART., T. Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing. In *USENIX Security* (2014), pp. 17–32.

[11] GILMORE, R., HANLEY, N., AND O'NEILL, M. Neural network based attack on a masked implementation of AES. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (May 2015), pp. 106–111.

[12] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] HACHEZ, G., AND QUISQUATER, J.-J. Montgomery exponentiation with no final subtractions: Improved results. In *International Workshop on Cryptographic Hardware and Embedded Systems* (2000), Springer, pp. 293–301.

[14] HAYKIN, S. *Neural Networks: A Comprehensive Foundation*, 2nd ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[15] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep Residual Learning for Image Recognition. *CoRR abs/1512.03385* (2015).

[16] HEUSER, A., PICEK, S., GUILLEY, S., AND MENTENS, N. Lightweight Ciphers and their Side-channel Resilience. *IEEE Transactions on Computers* (2017), 1–1.

[17] HUA, W., ZHANG, Z., AND SUH, G. E. Reverse Engineering Convolutional Neural Networks Through Side-channel Information Leaks. In *Proceedings of the 55th Annual Design Automation Conference* (New York, NY, USA, 2018), DAC '18, ACM, pp. 4:1–4:6.

[18] ILYAS, A., ENGSTROM, L., ATHALYE, A., AND LIN, J. Black-box Adversarial Attacks with Limited Queries and Information. *CoRR abs/1804.08598* (2018).

[19] JAP, D., STÖTTINGER, M., AND BHASIN, S. Support vector regression: exploiting machine learning techniques for leakage modeling. In *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy* (2015), ACM, p. 2.

[20] KHAN, A., GOODHUE, G., SHRIVASTAVA, P., VAN DER VEER, B., VARNEY, R., AND NAGARAJ, P. Embedded memory protection, Nov. 22 2011. US Patent 8,065,512.

[21] KOBER, J., AND PETERS, J. *Reinforcement Learning in Robotics: A Survey*, vol. 12. Springer, Berlin, Germany, 2012, pp. 579–610.

[22] KOCHER, P., JAFFE, J., AND JUN, B. Differential power analysis. In *Annual International Cryptology Conference* (1999), Springer, pp. 388–397.

[23] KOCHER, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference* (1996), Springer, pp. 104–113.

[24] KRIZHEVSKY, A., NAIR, V., AND HINTON, G. CIFAR-10 (Canadian Institute for Advanced Research).

[25] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1* (USA, 2012), NIPS'12, Curran Associates Inc., pp. 1097–1105.

[26] KUČERA, M., TSANKOV, P., GEHR, T., GUARNIERI, M., AND VECHEV, M. Synthesis of Probabilistic Privacy Enforcement. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2017), CCS '17, ACM, pp. 391–408.

[27] LAI, L., SUDA, N., AND CHANDRA, V. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *CoRR abs/1801.06601* (2018).

[28] LECUN, Y., BENGIO, Y., ET AL. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks 3361*, 10 (1995).

[29] LECUN, Y., AND CORTES, C. MNIST handwritten digit database.

[30] LERMAN, L., POUSSIER, R., BONTEMPI, G., MARKOWITCH, O., AND STANDAERT, F.-X. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *International Workshop on Constructive Side-Channel Analysis and Secure Design* (2015), Springer, pp. 20–33.

[31] LUO, C., FEI, Y., LUO, P., MUKHERJEE, S., AND KAELI, D. Side-channel power analysis of a GPU AES implementation. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on* (2015), IEEE, pp. 281–288.

[32] MAGHREBI, H., PORTIGLIATTI, T., AND PROUFF, E. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering* (2016), Springer, pp. 3–26.

[33] MANGARD, S., OSWALD, E., AND POPP, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006. ISBN 0-387-30857-1, `http://www.dpabook.org/`.

[34] MITCHELL, T. M. *Machine Learning*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 1997.

[35] NAIR, V., AND HINTON, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (USA, 2010), ICML'10, Omnipress, pp. 807–814.

[36] NARAEI, P., ABHARI, A., AND SADEGHIAN, A. Application of multilayer perceptron neural networks and support vector machines in classification of healthcare data. In *2016 Future Technologies Conference (FTC)* (Dec 2016), pp. 848–852.

[37] OHRIMENKO, O., COSTA, M., FOURNET, C., GKANTSIDIS, C., KOHLWEISS, M., AND SHARMA, D. Observing and Preventing Leakage in MapReduce. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, ACM, pp. 1570–1581.

[38] OHRIMENKO, O., SCHUSTER, F., FOURNET, C., MEHTA, A., NOWOZIN, S., VASWANI, K., AND COSTA, M. Oblivious Multi-party Machine Learning on Trusted Processors. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Berkeley, CA, USA, 2016), SEC'16, USENIX Association, pp. 619–636.

[39] PAPERNOT, N., MCDANIEL, P., GOODFELLOW, I., JHA, S., CELIK, Z. B., AND SWAMI, A. Practical Black-Box Attacks Against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (New York, NY, USA, 2017), ASIA CCS '17, ACM, pp. 506–519.

[40] PARASHAR, A., RHU, M., MUKKARA, A., PUGLIELLI, A., VENKATESAN, R., KHAILANY, B., EMER, J., KECKLER, S. W., AND DALLY, W. J. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (June 2017), pp. 27–40.

[41] PICEK, S., HEUSER, A., JOVIC, A., BHASIN, S., AND REGAZZONI, F. The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems 2019*, 1 (Nov. 2018), 209–237.

[42] PROUFF, E., AND RIVAIN, M. Masking against side-channel attacks: A formal security proof. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2013), Springer, pp. 142–159.

[43] RISCURE. https://www.riscure.com/blog/automated-neural-network-construction-genetic-algorithm/, 2018.

[44] SHOKRI, R., STRONATI, M., SONG, C., AND SHMATIKOV, V. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)* (May 2017), pp. 3–18.

[45] SONG, C., RISTENPART, T., AND SHMATIKOV, V. Machine Learning Models That Remember Too Much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2017), CCS '17, ACM, pp. 587–601.

[46] TELECOM PARISTECH SEN RESEARCH GROUP. DPA Contest (4th edition), 2013–2014. `http://www.DPAcontest.org/v4/`.

[47] TEUFL, P., PAYER, U., AND LACKNER, G. From NLP (Natural Language Processing) to MLP (Machine Language Processing). In *Computer Network Security* (Berlin, Heidelberg, 2010), I. Kotenko and V. Skormin, Eds., Springer Berlin Heidelberg, pp. 256–269.

[48] THOMAS, P., AND SUHNER, M.-C. A New Multilayer Perceptron Pruning Algorithm for Classification and Regression Applications. *Neural Processing Letters 42*, 2 (Oct 2015), 437–458.

[49] TRAMÈR, F., ZHANG, F., JUELS, A., REITER, M. K., AND RISTENPART, T. Stealing Machine Learning Models via Prediction APIs. *CoRR abs/1609.02943* (2016).

[50] VEYRAT-CHARVILLON, N., MEDWED, M., KERCKHOF, S., AND STANDAERT, F.-X. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *International Conference on the Theory and Application of Cryptology and Information Security* (2012), Springer, pp. 740–757.

[51] WANG, B., AND GONG, N. Z. Stealing Hyperparameters in Machine Learning. *CoRR abs/1802.05351* (2018).

[52] WEI, L., LIU, Y., LUO, B., LI, Y., AND XU, Q. I Know What You See: Power Side-Channel Attack on Convolutional Neural Network Accelerators. *CoRR abs/1803.05847* (2018).

[53] XU, X., LIU, C., FENG, Q., YIN, H., SONG, L., AND SONG, D. Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2017), CCS '17, ACM, pp. 363–376.

[54] XU, Y., CUI, W., AND PEINADO, M. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2015), SP '15, IEEE Computer Society, pp. 640–656.