Side-Channel Analysis of Integrate-and-Fire Neurons within Spiking Neural Networks

Matthias Probst*, Manuel Brosch*, Georg Sigl*†
*School of Computation, Information and Technology, Technical University of Munich, Munich, Germany
†Fraunhofer Institute for Applied and Integrated Security (AISEC), Munich, Germany
{matthias.probst, manuel.brosch, sigl}@tum.de

***Abstract*—Spiking neural networks gain increasing attention in constraint edge devices due to event-based low-power operation and little resource usage. Such edge devices often allow physical access, opening the door for Side-Channel Analysis.**

In this work, we introduce a novel robust attack strategy on the neuron level to retrieve the trained parameters of an implemented spiking neural network. Utilizing horizontal correlation power analysis, we demonstrate how to recover the weights and thresholds of a feed-forward spiking neural network implementation. We verify our methodology with real-world measurements of localized electromagnetic emanations of an FPGA design. Additionally, we propose countermeasures against the introduced novel attack approach. We evaluate shuffling and masking as countermeasures to protect the implementation against our proposed attack and demonstrate their effectiveness and limitations.

***Index Terms*—spiking neural networks, side-channel analysis, integrate-and-fire neuron**

I. Introduction

Implementations of Neural Networks (NNs) on edge devices become increasingly popular. Moving from cloud-Artificial Intelligence (AI) towards edge-AI is ongoing. Compression techniques enable the execution of NNs on resource constraint devices since they improve performance and power efficiency [2]. However, training networks is costly and often performed offline in powerful computing environments. Moreover, high-quality data is required for training. As a result, saving the effort for training is a lucrative target for an adversary.

Several publications [3]–[5] outline that Side-Channel Analysis (SCA) is putting the confidentiality of the Intellectual Property (IP) in the form of the parameters stored in an NN and its structure at risk. SCA provides a methodology to extract the network from a competitor to gain an advantage without the costly training process. For example, effective adversarial examples often require a white-box model [6], which can be obtained by the proposed SCA methods. Furthermore, trained models can also leak information about training data, especially when all network parameters are exposed. In medical records, training data must remain confidential, and leaking such sensitive information can result in a breach of privacy, for instance [7].

The paper is accepted to IEEE Transactions on Circuits and Systems I: Regular Papers [1] and is published under doi.org/10.1109/TCSI.2024.3470135. Digital Object Identifier 10.1109/TCSI.2024.3470135

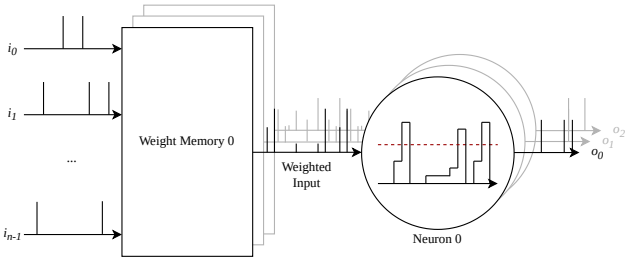


Fig. 1: Schematic of a simple SNN layer of three neurons reacting on n inputs.

As an alternative to classical NN implementations, Spiking Neural Networks (SNNs) offer lower power consumption and little resource usage. Human neurons interchange information via synapses in electric pulses, so-called spikes. This behavior is mimicked within SNNs and leads to an event-based processing, where information is encoded as spikes. A schematic of a layer of such a SNN can be seen in Fig. 1, where three neurons react on input spikes in parallel. Due to the simple and event-driven behavior of SNNs, they are more power efficient compared to classical NNs and, thus, ideal for usage on the edge [8].

Similar to implementations of classical NNs, the protection of IP stored in trained SNNs is crucial. Traditional NNs are already analyzed from a security perspective. However, little work exists in the domain of SNNs concerning hardware security analysis and countermeasures against such attacks.

A. Related Work

Recent publications focus on SCA to extract IP from classical NNs. Prominent works exploit the timing behavior [9] or power side-channel [10] to reverse engineer the architecture of an NN. In addition to the network architecture, the parameters are of particular interest. Batina et al. utilize Electromagnetic (EM) measurements to recover a complete NN from a software implementation [3]. They use a combination of the timing and EM side-channel to perform Simple Power Analysis (SPA), as well as Correlation Power Analysis (CPA). Takatoi et al. add to the work from Batina et al. by extracting the type of activation function from EM traces rather than timing behavior [11]. Considering hardware implementations of NNs, Hua et al. extract a Convolutional Neural Network (CNN), executed on

a hardware accelerator [5]. Dubey et al. extract the weights of a hardware implementation of a quantized NN by performing a CPA [4].

The collection of SCA on NNs indicates the need for effective countermeasures. Lui et al., for example, shuffle memory access patterns to hinder attacks focusing on this information [12]. Brosch et al. apply execution order shuffling to a microcontroller implementation of NNs to significantly increase the effort for an attacker to perform a CPA [13]. Dubey et al. also address the need for countermeasures and propose a masking scheme, which applies to different degrees of quantization in its latest iteration named ModuloNet [14]. Athanasiou et al. introduce another masking scheme, and its feasibility is demonstrated on a software implementation [15].

The related work we covered so far focuses merely on implementations of classical NNs but not on SNNs. Similar attacks on implementations of SNNs with the same scale as for classical NNs do not exist to the best of our knowledge. Nevertheless, Yang et al. propose an algorithm to effectively reconstruct the neural network topology from the firing activity of a biological neural network [16]. This can be transferred to an attack, if an attacker can access the communication between neurons. However, such an attack assumes a very skillful and well-equipped attacker. Garaffa et al. recover the weights of a SNN, which preprocess the associated input before it is fed to the first hidden layer, based on timing information and information depicted in power traces [17]. However, they only retrieve parameters of the input layer and are restricted to networks where dedicated spikes are exchanged and visible in the power trace. Goswami et al. use an activity monitoring approach to determine the number of active neurons in a SNN [18]. Again, they are limited to dedicated spiking patterns occurring in measurements. Nagarajan et al. utilize templates to retrieve parameters of an analogous spiking neural network consisting of two layers featuring three neurons each [19]. Their template attack is evaluated based on simulations and requires a profiling device in real-world. Their proposed countermeasure to introduce dummy neurons for additional noise indicates that the attack largely depends on high Signal-to-Noise Ratio (SNR).

The mentioned works [17], [18] have the drawback that they either need to resolve individual spikes or are limited to a specific spike exchange mechanism since individual spikes need to be visually identifiable in the traces. Moreover, they do not extract all parameters of the implemented SNN but only a subset. The work of Nagarajan et al. demonstrate their template-based attack only on a very small example and based on simulations. Moreover, a high SNR is required for the attack in [19]. Our work improves the state of the art because we do not rely on any assumptions about the spike exchange mechanism nor on the size of the SNN. Concretely, not even actual spikes need to be exchanged between the neurons. We target the neuron's implementation by utilizing a horizontal CPA approach where we discuss what part of a spiking neuron is targeted and how hypotheses are built for individual neurons in the SNN setting. Our horizontal CPA approach does not require a profiling phase, and our results show that we can attack neurons in any layer of a network of any size. Additionally, we can extract any parameter of the attacked SNN, i.e., hyperparameters such as layer sizes and neuron parameters as the weights. Furthermore, our proposed methodology allows for the extraction of parameters from a single trace, resulting in a high attack performance. These properties make our approach more robust than previous works, enable attacks on large SNNs. Widely used spike exchange protocols such as Address Event Representation (AER) or noise generation cannot mitigate our approach. We are also the first to analyze countermeasures in the context of SNNs on the neuron level in detail. Our proposed defenses provide resistance against our introduced CPA, reduce leakage in shuffling, and eliminate first-order leakage entirely in the case of masking.

B. Contribution

In this work, we address SCA of SNNs implemented on digital hardware. We employ horizontal CPA to extract parameters of synchronous feed-forward SNNs implementing Integrate-and-Fire (IF) neurons. In the context of SNNs, several neuron models and accelerator types exist, which this work cannot cover. Therefore, we focus on the IF neuron model. We target a Field Programmable Gate Array (FPGA) implementation similar to [20], [21] to acquire results from real-world measurements and extend our implementation with countermeasures. The main contributions are:

1) We introduce an attack methodology based on CPA to retrieve the parameters of a trained SNN from a hardware implementation. Due to the time-based processing of SNNs, a new attack approach is necessary compared to existing attacks on traditional NNs. We exploit the sequential spike processing in SNNs to craft a horizontal attack where a single trace is sufficient. The feasibility is analyzed based on local EM measurements of an FPGA, allowing us to retrieve parameters accurately with just one trace.
2) We introduce two countermeasures to hinder the attack: spatial shuffling and algorithmic masking on the neuron level. The results show a significant reduction of leakage when shuffling is used, while the introduced masking scheme prevents our attack completely.

To the best of our knowledge, countermeasures in the context of SNNs were not analyzed in any prior work. Our proposed methods are effective against the introduced CPA and SCA in general, which is confirmed by Test Vector Leakage Assessment (TVLA).

C. Organization

The remainder of this work is structured as follows: First, we cover the background behind SNNs and SCA in Sec. II. Then, Sec. III explains the fundamentals of our attack strategy and Sec. IV illustrates the results for local EM measurements. Our proposed countermeasures are covered and evaluated in Sec. V and Sec. VI. Finally, we conclude the paper in Sec. VII.

II. Background

This section presents the fundamentals to SNNs and the used hardware design. Moreover, the basics to SCA and side-channel leakage assessment are provided. Lastly, the threat model is defined.

A. Spiking Neural Networks

SNNs are inspired by the biological neural system and how information is transmitted and processed by the brain. The function of neurons within a SNN differs significantly from neurons in classical NNs. Like the classical neuron model, spiking neurons have weighted inputs to receive information from other preceding neurons or some external source. However, these inputs encode the information as so-called spikes [8]. While both multiplications and accumulations are required in classical NNs, accumulations are sufficient in SNNs. Moreover, SNNs are believed to reduce energy consumption significantly due to their event-driven sparsity [8]. The IF neuron model is the simplest possible for SNNs.

B. SNN Implementation

The realization of our IF neuron is depicted in Fig. 2. At the start, the neuron potential v is initialized with the negative threshold $-t$. An input i to the neuron provides the index for loading the respective weight w_i from weight memory. The weight w_i is added to the current membrane potential v and stored in the register when the reset logic is inactive. Once enough weights are accumulated, v becomes positive, triggering two events. First, a spike is emitted at the output o for one clock cycle in form of a logical 1 since the Most Significant Bit (MSB) is zero. Furthermore, the reset logic is now active and resets v to $-t$ via the bottom AND gate. The block MSB in Fig. 2 repeats the most significant bit's value in that process, so it is the same width as v or $-t$, while the spike output o remains a single bit value.

Multiple such neurons are implemented in parallel within a layer of a SNN. Each neuron in one layer reacts on the same input sequence. While neurons from the input layer react on the input provided to the SNN, subsequent layers react on the previous layer's output. As a network topology, we consider feed-forward SNNs similar to Multi-Layer Perceptrons (MLPs) from the classical NN domain. Weights w are $4\,\mathrm{bit}$ wide in our case, whereas the membrane potential register has a width of $8\,\mathrm{bit}$[1]. The threshold values are limited to the width of the potential register ($8\,\mathrm{bit}$), and each neuron can have a different threshold.

C. Side-Channel Analysis

SCA was introduced in 1996 by Kocher [22]. It originates from the cryptographic area and exploits information leakage about the secret key through physical quantities. Differential attacks, like CPA [23], employ statistical methods to utilize data dependencies within the power consumption for different input data to retrieve secret information.

[1]The bit-width of the weights and threshold are specified by our implementation. Other values will impact the computational effort of the CPA but not the success of it.

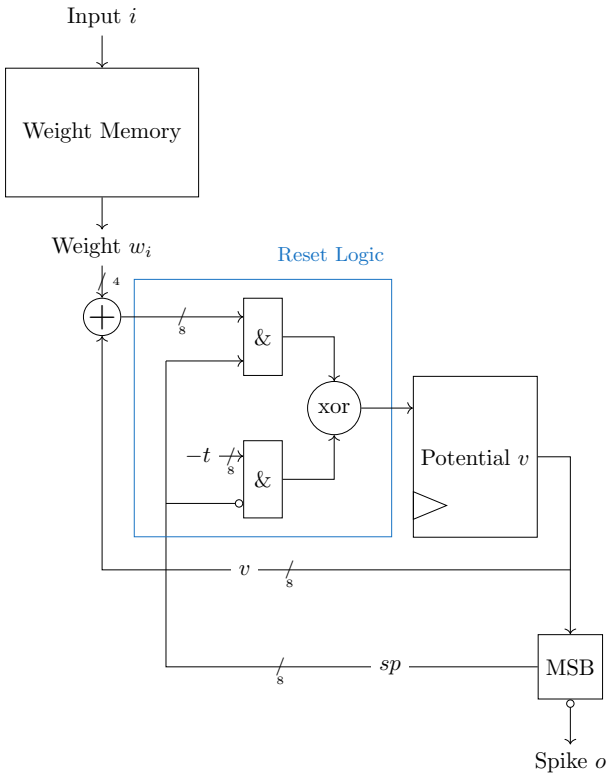


Fig. 2: Schematic of our IF neuron implementation.

CPA is used in the context of classical NNs to retrieve weight and bias values of neurons. Software [3] and hardware [24] implementations are demonstrated to be vulnerable to CPA. To mount a successful CPA, an operation must be chosen as a suitable point of attack. The target operation needs to combine known data with the secret data an adversary wants to retrieve. Typically, for NNs, the multiplication between weights and inputs is utilized since the input to the NN can be assumed to be known [3], [24]. Before performing an attack, the adversary needs to measure the device's power consumption or EM emanations while different input data is applied. The principle of the attack is to retrieve the weights of all neurons layer by layer, i.e., an incremental attack is performed where a layer can only be attacked if all preceding layers are known. Targeting the first layer, an attacker calculates intermediate results based on hypothetical values for a single weight of one neuron and the known corresponding input data. The intermediate results are calculated for all possible values a weight can have. Thereby, a matrix $\mathbf{V}_{\text{hyp}}$ is generated of dimension (number of traces $\times$ number of hypotheses), which includes all calculated intermediate results. Using a power model, each entry of $\mathbf{V}_{\text{hyp}}$ is transformed into a theoretical power consumption value, i.e., a new matrix $\mathbf{H}_P$ is generated. Every column of $\mathbf{H}_P$ is then correlated with

the measurements. When the device performs the targeted weight multiplication with the input data at a specific time, the correct hypothesis's correlation is significantly higher than the correlation of all other hypotheses. By repeating the described methodology, each weight of every neuron in the first layer can be extracted. As soon as all parameters of the first layer are known, the subsequent layer can be attacked because its input can be calculated. For more detailed information, we refer to [3], [24].

D. Test Vector Leakage Assessment

A generic methodology to evaluate an implementation to its leakage behavior is TVLA, which uses Welch's t-test to statistically assess the occurrence of data dependencies within side-channel measurements [25]. Commonly, the so-called fixed-vs.-random t-test is used to evaluate an implementation for side-channel leakages. While measuring the desired side-channel, the device processes fixed or random inputs randomly, respectively. During this process, the secret is kept fixed across all measurements. Then, the t-value for each sample point is calculated with the following formular:

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

where μ_0 and μ_1 are the sample means of the two datasets and the sample variances are represented by ${s_0}^2$ and ${s_1}^2$, respectively. Absolute scores above 4.5 indicate univariate first-order leakage that may be usable for an attack. The threshold of $|t| > 4.5$ is established to reject the null hypothesis that no leakage is present, with a confidence of more than 0.99999. The implementation can be seen as first-order secure if the threshold is not crossed for any sample point.

To detect leakages of higher orders, the test must be extended. For instance, the sample points must be mean-free squared to show univariate second-order leakage [26]. Several sample points in measurement must be combined to exhibit multivariate leakages since the actual leakage spreads over multiple sample points [26]. A mean-free multiplication of samples can indicate multivariate second-order leakage.

E. Threat Model

We assume an attacker with physical access to the device executing the target SNN. Moreover, the adversary knows where the neurons are located on the chip. The attacker is passive, i.e., can observe the device during its regular operation but cannot interfere with the computations by inducing faults, for example. We use a decapped chip for increased SNR. Consequently, we are in the semi-invasive scenario, where modifications to the hardware are allowed as long as they do not alter the device's functionality. Furthermore, no access to the memory or bus system is possible for the attacker, meaning monitoring memory access or retrieving data from memory is impossible. In addition, we assume that the network parameters are securely loaded to the neurons.

However, the attacker has complete control over the input and can observe side-channel information as physical access to the targeted device is assumed. An adversary is interested in extracting the IP stored within the SNN in the form of secret threshold and weight parameters since training a network successfully requires high computational effort and qualitative training data.

III. Attack Methodology

This section describes our approach to retrieving the neuron parameters with a horizontal CPA methodology. First, we explain how to extract the parameters of a neuron in the very first layer. Next, we transfer our approach to subsequent layers.

A. Neuron Parameter Retrieval from the First Layer

For a neuron located in the first layer, an attacker can control the inputs of this neuron according to our attacker model. Consequently, an attacker can apply a spike to one or many chosen inputs. Furthermore, she/he can repeat this process for multiple other inputs. To explain the attack, we assume that an adversary stimulates just one of all inputs and leaves the other inputs untouched, i.e., no spike is applied to them. When an adversary stimulates the input, she/he observes the EM emanations with a local EM-probe above the neuron. The adversary stimulates the input multiple times with some input sequence I_0, where she/he can choose a random input of the subset of currently stimulated inputs S, since this input is directly handed to the neurons of the very first layer $l = 0$. In case of attacking one input, S includes this one input. The adversary then continuously observes the EM radiation of the neuron while never applying a reset to the neuron[2]. While the input is applied, the neuron reacts based on its weight w and threshold t. The weight associated with this input is added to the negative threshold for the first stimulation. After applying more inputs, the neuron's potential increases steadily and eventually becomes positive at some point. When the neuron's potential v is positive, it resets it to the negative threshold, and the accumulation process repeats. An attack point needs to be determined to retrieve information about the weights of a neuron from the EM measurements. This target point needs to process the weights of a neuron together with data known by the attacker. As already mentioned, we are focusing at first on cases where an adversary stimulates only a single input. Consequently, only a single weight is added to the membrane potential v. For the proposed attack, an attacker calculates the value of the membrane potential at every point in time, where she/he applies a spike to the input, for each possible value of the weight and the threshold. Thereby, the membrane potential can have the following values for a point in time x:

$$v_x = \begin{cases} v_{x-1} + w_{\text{hyp}} - t_{\text{hyp}}, \text{ if } v_{x-1} - t_{\text{hyp}} < 0 \\ -\, t_{\text{hyp}}, \text{ if } v_{x-1} - t_{\text{hyp}} \geq 0 \end{cases} \tag{2}$$

where w_{hyp} and t_{hyp} are hypothetical assumptions for the weight and threshold of the neuron, respectively. An attacker

[2]Only at the beginning of this experiment, the neuron is reset, so the initial potential is the negative threshold.

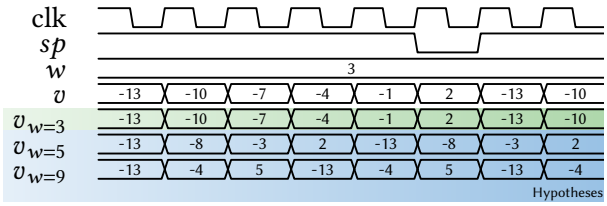


Fig. 3: Sequence of potential v_w changes based on some attackers hypotheses. The correct hypothesis is green, while incorrect examples are marked blue. The threshold for all hypotheses is set to the correct threshold 13.

calulates the value transitions in the membrane potential for each measured input sequence for all possible hypotheses for the attacked weight and threshold, using Eq. (2). Figure 3 shows the mentioned value transitions within the membrane potential for some weight hypotheses and the same assumed threshold hypothesis. All calculated hypothetical value transitions result in the hypotheses matrix $\mathbf{V}_{\text{hyp}}$. The values within $\mathbf{V}_{\text{hyp}}$ need to be transformed into theoretical power values by using the Hamming Distance (HD) power model, in our case. Each entry of the matrix is transformed, calculating

$$P_{\text{hyp}} = \text{HD}(v_{x-1}, v_x) = \text{HW}(v_{x-1} \oplus v_x). \tag{3}$$

Thereby, the power hypotheses matrix $\mathbf{H}_P$ is generated, that only includes theoretical power consumption values for each transition of the membrane potential.

To evaluate which hypothesis best fits the measured side-channel information, the adversary compares hypothetical power consumption values $\mathbf{H}_P$ with the traces $\mathbf{M}^*$. The set of traces, including points of signal transitions $\mathbf{M}^*$, is a sliced version of the overall measurement $\mathbf{M}$, as shown in Fig. 4. The interval size of $\mathbf{M}^*$ can be determined based on the device's sampling and operation frequency. We utilize Pearson's correlation coefficient to get a numerical value for the similarity of the actual power consumption and the hypothetical one. The highest absolute correlation will most likely belong to the correct hypothesis and recover the weight and threshold of the neuron. Wrong hypotheses will feature incorrect assumptions on the targeted neuron potential, as indicated in Fig. 3. Thus, the power consumption will differ drastically from the correct one in those cases, and the hypotheses will not match the measurements.

While keeping the local position of the probe constant at one point, the attacker will repeat the attack until all parameters of the neuron are retrieved. Next, the probe is moved to the next neuron to retrieve its parameters similarly. Since an adversary does not know the position of all neurons, she/he needs to move the probe over the chip with a reasonable step size. As just one trace per position is needed, the duration of this process is primarily influenced by moving the probe. We used half of the probe diameter in our experiments for sufficient resolution. If a position resolves two neurons simultaneously, the attack is still successful as indicated later in Sec. IV.

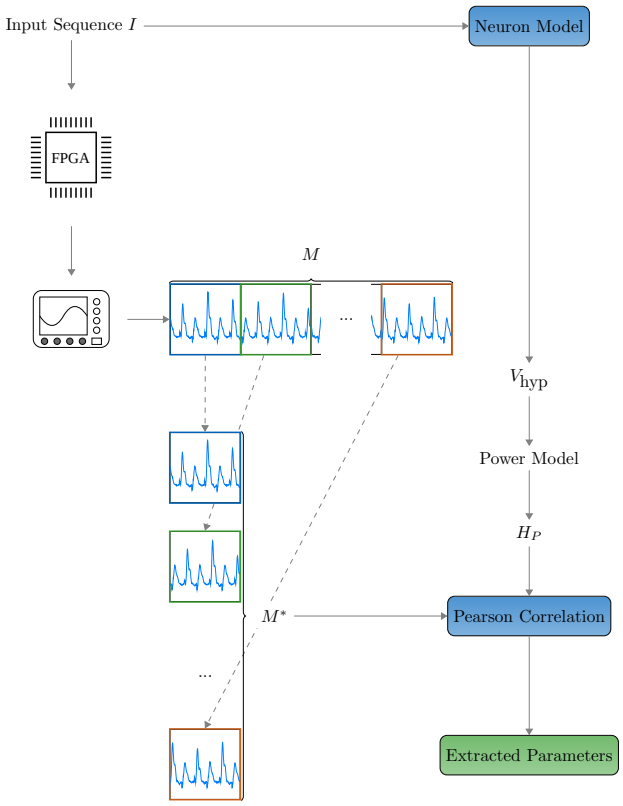


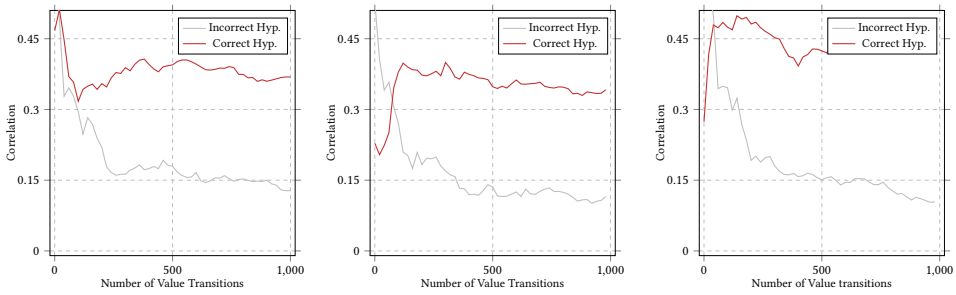
Fig. 4: Schematic of conducting our horizontal CPA

Moving the probe over the chip to attack neurons must be repeated for every layer.

The amount of weights considered in the hypotheses depends on the attacker's resources and can be chosen because the attacker can control the amount of active inputs. So far, we have described an adversary as only stimulating one input. This is the most essential form of our attack, and thus, one horizontal trace gives the attacker knowledge over the threshold and exactly one weight. The attacker will collect more traces for the remaining weight parameters and attack them similarly while considering the already extracted threshold parameter. Our implementation features an $8\,\text{bit}$ threshold and $4\,\text{bit}$ weights. The bit-width of both originates from state-of-the-art implementations such as Ju et al. [20] and Upegui et al. [21] which use $8\,\text{bit}$ and $9\,\text{bit}$ as their largest bit-width, respectively. Furthermore, the very well-studied field of classical NNs often employs aggressive quantization techniques. These methods reduce the bit-width of parameters to even less than $4\,\text{bit}$ while maintaining the original accuracy [27]–[29]. Thus, for extracting the threshold and one weight as a first step $2^{8+4} = 4096$ hypotheses are necessary. In contrast, for weight extraction alone $2^4 = 16$ hypotheses are enough[3].

An attacker with more available computational power can stimulate n_w inputs at once, allowing extracting n_w weights

[3]Note that the number of hypotheses changes for a different implementation



(a) Neuron in layer $l = 0$. (b) Neuron in layer $l = 1$. (c) Neuron in layer $l = 2$.

Fig. 5: CPA result for EM measurements for three neurons in layers $l = 0, 1, 2$.

with one horizontal attack. This also makes the attack more robust since the variance of potential values is higher. The stimulated input at each point in time will then be randomly chosen from the set of currently used inputs. The number of necessary hypotheses corresponds to $N_H = 2^{8+4 \cdot n_w}$ including the threshold and $N_H = 2^{4 \cdot n_w}$ if the threshold is already found, where n_w defines the number of weights that are used for the attack. Implementations with different parameter sizes can require more hypotheses. However, the same trend applies for any other CPA on NNs since the attack complexity is always proportional to the parameter sizes in any NN implementation.

B. Neuron Parameters of Subsequent Layers

With our approach, an attacker can retrieve all parameters of spiking neurons by controlling their input sequence and using CPA. Stimulating inputs of the input layer is trivial. However, the adversary cannot directly control the input for neurons in the subsequent layers. However, already extracted layers can be used to model the input of the next layer.

In Alg. 1, the methodology to get measurements for a neuron in a hidden layer with known inputs is described. In particular, once the input layer L_0 is known, it is possible to find the input I_1 to the second layer with $I_1 = \sigma(L_0, I_0)$, where I_0 equals the network input. The function σ allows the modeling of the sequence to some layer under the consideration of input sequences I_0 and known layer parameters L_0. In our experiments, we get the result of σ utilizing simulation with the already extracted network parameters. For further layers l the sequence I_l can be found with $\sigma(L_{0,..,l-1}, I_0)$. The sequence I_l may stimulate too many inputs of L_l at once, leading to a large hypotheses space. This is not desirable since, for extracting n_w at once, the adversary only wants to stimulate n_w inputs of the current layer under attack. We denote the set of inputs to be stimulated as S, which has a size of n_w. Thus, all elements of the inputs I_l of layer l must also be in the set S to be able to attack. The process for collecting measurements is explained in Alg. 1, independent of the currently targeted layer.

The attacker will randomly apply an input I_0 in the simulation model σ until a sequence I_l only stimulates the inputs of layer l included in the set S. This sequence needs to be sufficiently long to get enough points of interest so the CPA can determine the correct parameters. In Sec. IV, we show that about 200 signal transitions are sufficient here.

Data: Parameters of prev. Layers $L_{0,..,l-1}$
Result: Input Sequence I_l for conducting CPA
do
  $I_0 \leftarrow P_{0,\text{rand}}$; `// Apply random input to first layer`
  $I_l \leftarrow \sigma(L_{0,..,l-1}, I_0)$; `// Get input sequence` I `to layer` l
while $\{\exists x \in I_l | x \notin S\}$;

Algorithm 1: Finding input sequences I_l to layer l, which only include the subset of inputs S.

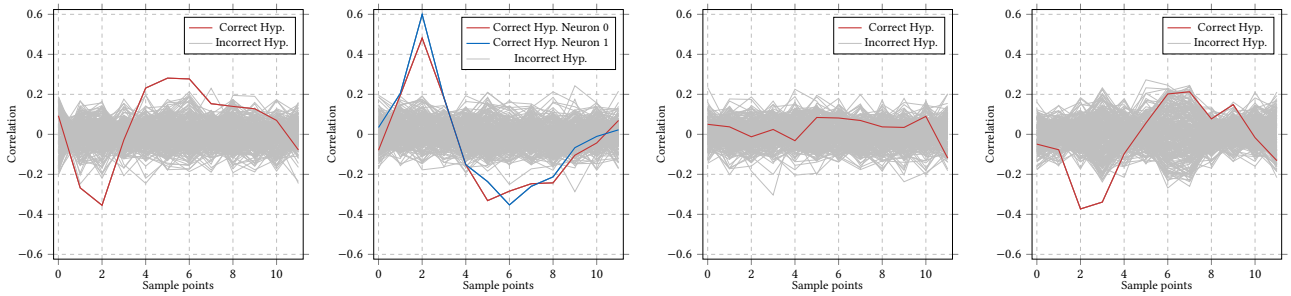
The adversary uses the input sequence I_l to build up the hypotheses. With the respective trace $\mathbf{M}_l$, the CPA is performed to retrieve the neuron's threshold and n_w parameters under test in layer l. Repeating the process of Alg. 1 in combination with the horizontal CPA gives the attacker all parameters of one neuron in L_l. Next, the probe position is changed to attack the next neuron.

Additionally, the attacker must determine if the targeted neuron belongs to the layer under attack. Since the network is attacked layer by layer, the attacker starts with layer L_0. By applying sequences I_0, hypotheses on parameters of L_0 are built. The CPA detects high correlation values since the modeled intermediate behavior suits the actual behavior of a L_0-neuron. If a neuron from a layer other than L_0 is measured, the modeled and actual behavior will not match, resulting in low overall correlation values with no distinct maximum. The low correlation indicates that the neuron does not belong to layer L_0.

IV. Attack Results

We utilize the NewAE Technology CW305 board featuring a Xilinx Artix-7 FPGA, which is decapped[4] to increase the SNR of EM radiation. The device runs at a frequency of 100 MHz. The measurements are collected by a PicoScope 6402D that samples with 1.25 GHz and a near field EM probe

[4]Decapping is not strictly necessary. A packaged chip results in a lower SNR, which may increase the number of traces for a successful attack.



(a) Neuron in layer $l = 0$.

(b) Two neurons in layer $l = 0$ measured at the same time.

(c) Neuron in layer $l = 1$ with hypotheses for $l = 0$

(d) Neuron in layer $l = 1$ with hypotheses for $l = 1$

Fig. 6: CPA result for EM measurements in the time domain.

(Langer ICR HH250-75), which is amplified with a Langer PA 303 by 30 dB. For the attack on EM traces, we implement a three-layer SNN with 20 neurons in layer $l = 0$ and 15 in each following layer ($l = 1, 2$). The 4-bit weights and the 8-bit thresholds are chosen randomly[5]. All results originate from measurements following Alg. 1.

For the horizontal CPA approach, we need to collect one trace $\mathbf{M}$ and crop it to $\mathbf{M}^*$. Alignment and the correct window size are crucial for horizontal CPA. In our case, a single trace $\mathbf{M}$ is collected, where we record about 200 signal changes of the potential value v. Directly after reset and, thus, before the first signal change of v, a trigger pin is set to indicate the start of the measurement. After that, the window size must be correct to keep the alignment constant. To determine the window size, we identify one clock cycle of the FPGA being 12.5 sample points. Furthermore, one value change in v is present every clock cycle. Thus, we chose our window size to be 12, while every 13th sample point is neglected. This results in a not completely perfect alignment since every second window has an offset of 0.5 sample points. The amount of windows in $\mathbf{M}^*$ is dependent on the length and, thus, on the included potential changes in $\mathbf{M}$.

Figure 5 shows the results of the CPA based on EM measurements. We target $n_w = 2$ weights at the same time as well as the threshold. We require at most 200 signal transitions of the target neuron potential in a single trace to retrieve the weight and threshold parameters in our experiments. Thus, an adversary can successfully recover the true parameters of each neuron. The retrieval is independent of the layer the neuron is located in since the difference in the results depicted in Figs. 5a to 5c is insignificant. Therefore, an attacker can retrieve all neurons by localizing them and applying our attack.

Figure 6 depicts four correlation results in the time domain, i.e., the correlation value when correlating $\mathbf{M}^*$ with $\mathbf{H}_P$ is plotted over time. The correct hypothesis/hypotheses are colored. Due to the mentioned small misalignment, the maximum correlations in Fig. 6 are found at sample points one to three. In Fig. 6a, we can see the correlation of a neuron in the first layer in the time domain. The result originates from the same measurement as the correlation plot from Fig. 5a when taking 200 signal transitions into account. One can see that the correct hypothesis exceeds all others at the aligned sample point two, where the neuron potential changes. In Fig. 6b, we can see the result if the local EM probe resolves two neurons at the same position. In that case, we can identify two significantly high hypotheses, giving the adversary the ability to extract both neurons' parameters at once, even with an inaccurate probe position. Each high hypothesis is the correct one for one neuron, respectively. Consequently, relevant correlation coefficients over 0.3 indicate parameters of actual neurons. If the correlations for all hypotheses are below the mentioned threshold, either there is no neuron present at the position, or the hypotheses were chosen incorrectly, as shown in Fig. 6c. In the case of Fig. 6c, the probe was placed over a neuron in the second layer even though the hypotheses for a first-layer attack were used. In both cases, the position can be neglected to attack the current layer.

Figure 6c shows results for attacking a second-layer neuron when using incorrect hypotheses assumptions. When correctly building hypotheses for the second layer, the correct parameters correlate again in Fig. 6d, giving us the result from Fig. 5b in the time domain. Thus, an adversary can determine to which layer the current neuron under attack belongs.

An adversary can precisely retrieve neuron parameters with the proposed attack and EM traces, resulting in an identical network clone. The setup used can record 20 traces per second. Hence, an NN with 600,600 parameters consisting of 600 neurons with 1,000 weights and one threshold value each, can be fully extracted within ten minutes. Moving the local EM probe will add about 50 minutes, giving the adversary all parameters in about an hour. Scaling this for larger prominent models, extracting all 25.6 million parameters of ResNet50 [30] takes less than 43 hours, for instance. To clone VGG16 [31] that consists of about 138 million parameters, our method requires about ten days. Furthermore, the setup can be scaled by placing multiple EM probes, enabling an attacker to retrieve as many neurons in parallel as probes on the chip.

[5]The attack is independent of the selected weight and threshold parameters, i.e., any weight and any threshold can be retrieved the same way. We choose random values to be independent of any specific network.

V. Countermeasures

In the following, we describe how shuffling and masking can protect the SNN parameters from being extracted. Both countermeasures can prevent our horizontal CPA and the classical vertical CPA approach. In addition, we discuss the overhead for the countermeasures.

A. Shuffling

Shuffling increases the attack effort by reducing the SNR within measurements but cannot counter a first-order attack completely. By executing operations in a random sequence, at a random place, or as a combination of both, the SNR decreases. Shuffling is used in the context of NNs to harden a microcontroller implementation against SCA in [13]. It is simple to implement but very effective against SCA, since even for a small network, the trace acquisition increases dramatically. By executing l operations in random order, the amount of traces for a successful attack increases according to the following formula

$$n_{traces} \approx \frac{28 \cdot l^s}{\rho^2_{ckt,proc}}, \tag{4}$$

where $\rho^2_{ckt,proc}$ is the correlation coefficient of the correct hypothesis and $s = 2$ for a CPA without preprocessing [32]. In the case of SNNs, shuffling the operation sequence is impossible due to their event-based processing.

Both examples [13], [32] are intended to randomize the execution sequence of the Multiply and Accumulate (MAC) operation sequence. This is impossible for SNNs since their event-based behavior is based on the execution sequence, and shuffling it will break the functionality. Since a localized attack is necessary, shuffling the position of the targeted operation is suited to increase attack effort. The horizontal CPA also dictates shuffling within one trace. In other words, shuffling the execution position for each new input is also insufficient to hamper the attack. Due to the attack properties, we need to randomly execute the potential computation at the same computation unit with a probability of l^{-1} when processing an input i at some point in time.

To achieve this in hardware, we separate the weight memory from the computation unit, which consists of the summation, the reset logic, the potential register, and the spike output. The computation units for shuffling five neurons are depicted in Fig. 7a. Nevertheless, the implementation can be adapted to feature more than five neurons. The computation unit is reused from the default neuron from Fig. 2 and extended by the multiplexer structures for input, loop, and output. The multiplexer is addressed with the corresponding spiking pattern $m_{\text{in/loop/out}}$. Hence, the computation indices match, i.e., w_0 is added upon v_0, and the spike o_0 is registered at the correct output. Consequently, the spiking pattern must be chosen to maintain the correct computation of the neurons. Additionally, the multiplexer structure ensures that the computation units are randomly shared between the five neurons while the input and output are identical to the unshuffled implementation of the network. Since the shuffling patterns are not pipelined in the design in Fig. 7a, the structure can process a new input every second clock cycle.

The shuffled computation unit group needs to be supplied with shuffling patterns to maintain the correctness of computations. To achieve this, we implement a shuffling pattern generator depicted in Fig. 7b. The generator is supplied with randomness dictating the shuffling pattern m. The amount of random bits required to shuffle n operations, i.e., n computation units, can be calculated by

$$\text{rand}_{\text{shuffle}} = \sum_{i=2}^{n} \lceil \log_2(i) \rceil. \tag{5}$$

According to Eq. (5), the unit above consisting of $n = 5$ computation units needs $8\,\text{bit}$ of randomness when processing a new input every second clock cycle. This randomness m is then influencing m_{in},m_{loop} and m_{out}. The bit vectors m_{in},m_{loop} and m_{out} are $15\,\text{bit}$ in size, meaning that every group of 3-bit is later the input signal to a multiplexer MUX. Concretely, the shuffling pattern generator sets the pattern for the next execution

$$\begin{aligned} m_{\text{in}} &\leftarrow S_m(m'_{\text{in}}) \\ m_{\text{loop}} &\leftarrow m \\ m_{\text{out}} &\leftarrow S^{-1}(m_{\text{in}}) \end{aligned} \tag{6}$$

where S_m denotes the shuffling function S with to pattern m The input pattern m_{in} is set to the previous input pattern shuffled by m. The output needs to invert m_{in}, which is achieved by the shuffling invert function S^{-1}. All of Eq. (6) can be found as their corresponding hardware units in Fig. 7b. The shuffling function S is realized with a multiplexer structure that changes the sequence of the five 3-bit groups within m_{in} according to m. The inverse shuffling function S^{-1} is indicated as a cloud, since the logic is rather complex to depict in Fig. 7b. It is realized with the procedural VHDL-code from listing 1, which results in combinatorial logic and a store of the result within a register[6].

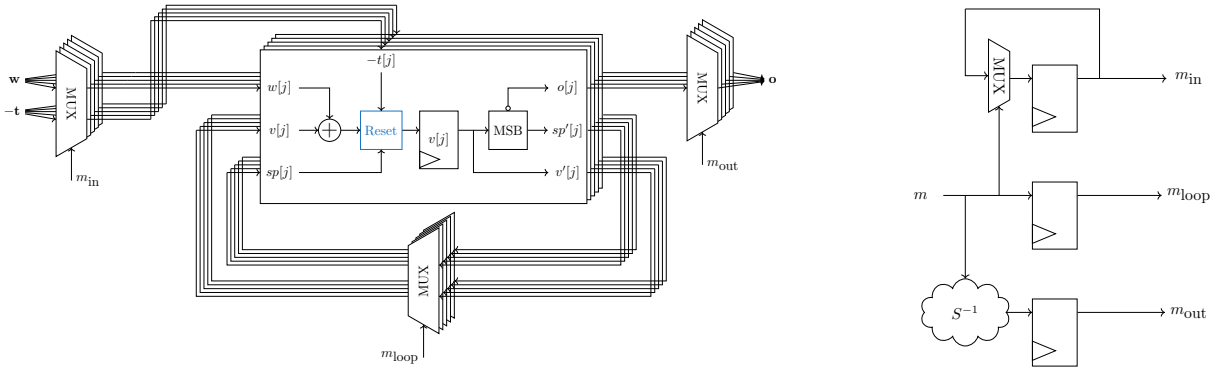
```
1  if rising_edge(clk) then
2      for i in 0 to 4 loop
3          if rst = '1' then
4              m_out(i) <= i;
5          else
6              for j in 0 to 4 loop
7                  if m_in(j) = i then
8                      m_out(i) <= j;
9                  end if;
10             end loop;
11         end if;
12     end loop;
13 end if;
```

Listing 1: VHDL for the inverse shuffling function S^{-1}

B. Masking

In contrast to shuffling, masking aims to make the power consumption independent of the processed data. To achieve

[6]Please note, that we omitted the casts of `i` and `j` to `std_logic_vector` for the sake of readability in listing 1.



(a) Shuffle computation unit for five neuron units

(b) Shuffling pattern generator

Fig. 7: The shuffled computation unit is depicted in (a) and the index j is used to highlight that internally the parameter indices are different from the index of the computational unit which they are processed in. The input and outputs are denoted as vectors indicated, where the index corresponds to the weight i-th neuron in unshuffled from. The shuffle pattern generator in (b) is necessary to generate multiplexer patterns. The multiplexer (MUX) in (b) realizes the shuffling function S within the vector m_{in}.

this, masking countermeasures split the algorithm's inputs and intermediate results into two or more shares by using random numbers [32].

Arithmetic and Boolean masking can be used to protect an implementation. Both techniques split some value $x \in \mathbb{Z}_L$ into $m+1$ shares. The first m shares x_0 to x_{m-1} are uniform random numbers, whereas the last share x_m is a combination of the unshared value x and the shares $x_0, x_1, \ldots, x_{m-1}$. Inputs and intermediate values remain shared during the algorithm computation. The output values can be recovered by recombining all shares. In Boolean masking, sharing and recombination are done by applying the *XOR* operation to all shares.

Masking countermeasures are typically proven to be secure in some assumed probing model. Concretely, we demonstrate the security of our masked neuron in the glitch-extended probing model. In the assumed model, an attacker can prob leakages by inserting glitch-extended probes at any wire within a component $\mathcal{G}$ of the implementation, often referred to as a gadget. Probed leakage includes all wires and combinatorial logic parts on which the probed wire depends. Since glitches cannot propagate across registers, they limit probing further information [33]. Formally, an adversary chose a set $\mathcal{P}$ of t glitch-extended probes in a gadget $\mathcal{G}$. The parameter t represents the attack order. Since we aim for first-order security, $t = 1$ in our case, i.e., an attacker can place a single probe. If an attacker cannot retrieve any information about secret data by probing any single wire, a gadget $\mathcal{G}$ is 1-probing glitch secure. To theoretically evaluate the security of an implementation, the notion of simulatability is relevant [34]. Simulatability means that the set of probes $\mathcal{P}$ is independent of the input values that are not provided to the simulator $\mathcal{S}$ [35]. For example, any single probe can be simulated with all input values since the distributions at arbitrary points within the gadget can be trivially determined. The notion of simulatability is the essential component to define Non-Interference (NI) and Strong Non-Interference (SNI). We are focusing on SNI since it allows a secure composition of gadgets without revealing any information, which could be the case if a gadget only satisfies NI.

Definition 1 (Glitch-Robust Strong Non-Interference [36]): A gadget $\mathcal{G}$ is glitch-robust t-SNI if for every set $\mathcal{I}$ of t_1 internal glitch-extended probes in $\mathcal{G}$ and for every set $\mathcal{O}$ of t_2 glitch-extended output probes, the joint set of probes $\mathcal{I} \cup \mathcal{O}$ can be simulated with at most t_1 shares of each input, for any t_1 and t_2, such that $t_1 + t_2 \leq t$.

To protect the IF-neuron by masking, the components must be implemented to operate on $m+1$ shares, where the original value x is shared by using m random values. In our case, Boolean masking is necessary since we need to compare the membrane potential against the threshold to determine whether a spike is generated[7]. Since we use Boolean masking, the m-th value is generated by applying the *XOR* operation on all random values and the original value x. In the following, shared representations are denoted as capital letters. At the same time, unshared original values are lowercase letters, i.e., the shared weight $W = (w_0, w_1, \ldots, w_m)$ can be recombined to the original value $w = \oplus_{i=0}^{m} w_i$.

The modified hardware implementation of a masked IF-neuron is depicted in Fig. 8. The potential V is initialized with the negative threshold value $-T$. When an input spike arrives, the selected weight W is added to the shared potential V with the secure adder gadget. After that, the reset logic is passed, which consists of two AND-gates implemented according to Domain-Oriented Masking (DOM) as shown in Fig. 9. The new potential V is stored in the membrane potential register

[7]Arithmetic masking does not allow to get the sign bit without converting the masks to the Boolean domain.

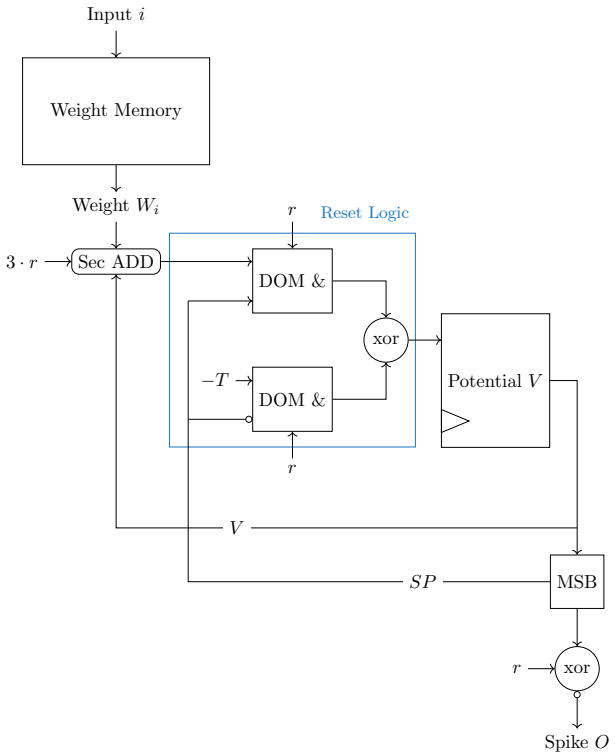


Fig. 8: Schematic of our Masked IF neuron implementation. Random values are denoted by r, where each r marks a new random value independent of other values already used.

after passing the reset logic. The MSB[8] of V is now checked, and whenever it indicates a positive value, a spike on the shared output O is emitted since the accumulation of weights has passed the threshold T. This also means that SP is now all zero in masked form, which forces the new potential to be $-T$ again due to the reset logic.

For the masked implementation, the weight memory size must be increased by $m+1$ since it features the shared weights W. To simplify Fig. 8, the weight memory is omitted in the schematic and would be located at the left, where the weight W is applied to the secure adder.

We implement secure AND-gates and utilize the DOM approach by Groß et al. [37]. To satisfy 1-SNI in the glitch-extended probing model, we extend DOM-gates by introducing further registers, which are highlighted in Fig. 9. During each gate operation, one random value is required for first-order security. The subsequent usage of the MSB of V is refreshed by adding a new random number. The required addition within the masked neuron is based on the secure addition module proposed by Coron et al. [38]. The first-order secure addition module allows the arithmetic sum of two Boolean masked values. It is further improved by

[8]MSB in Fig. 8 repeats the most significant bit's value, the same width as V or $-T$.

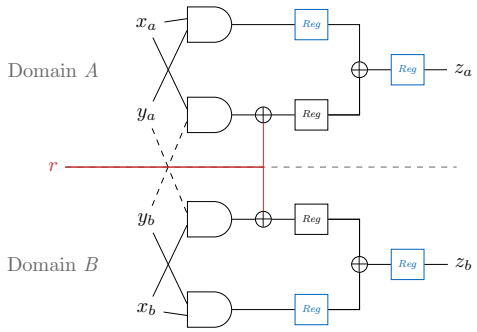


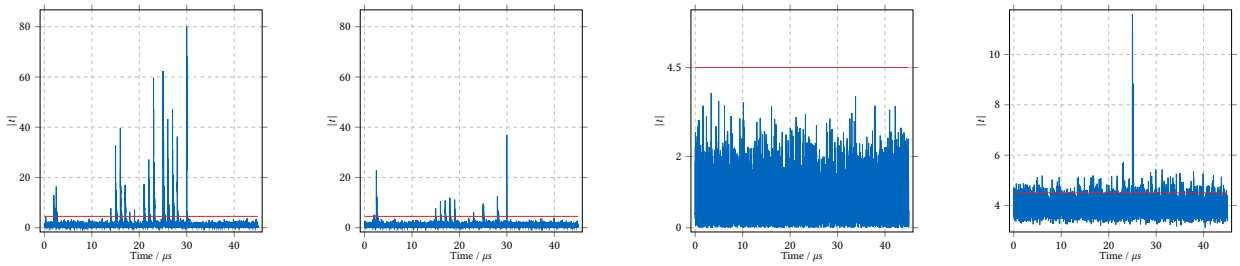
Fig. 9: 1-SNI DOM-gate under the glitch-extended probing model due to the introduction of highlighted registers.

Gigerl et al. [39] to achieve security in the presence of glitches, i.e., to fulfill Def. 1. The latency l of the secure addition equals $l = 2 \cdot n + 2$, where n is the bit-width of the inputs. Moreover, every summation requires $2 \cdot n + 3 \cdot (n-2)$ bits of randomness to remark the internal intermediate results. Both latency and randomness requirements indicate that the summation is the bottleneck of the masked neuron. All critical components, i.e., the secure adder and the DOM-AND-gate, can be implemented as higher-order secured schemes [37], [38]. Nevertheless, we only focus on first-order security in this work to give an impression of how effective masking can be in the context of SNNs.

In addition to designing the masked spiking neuron in order to fulfill Def. 1, we tested the obtained netlist with the formal verification tool CocoAlma [40]. CocoAlma confirms our implementation to be formally secure against first-order attacks when using the time-constraint probing model and considering glitches under CocoAlma's strict analysis.

C. Discussion of the Countermeasures

The required hardware resources for shuffling and masking are listed in Table I. Shuffling is listed with two numbers in each cell, where the first number corresponds to the resource usage of the shuffled computation unit and the second value to the shuffle generator. Since the computation unit shuffles five neurons, we must divide shuffling numbers by five to compare them with unprotected neurons. The number of LUTs increases by a factor of three per neuron from 18 to $308/5 = 61.6$ due to the multiplexer logic. The number of FFs increases from 19 to 85, which is roughly equal to the increase of neuron registers by a factor of five. The latency doubles and is defined by the shuffled computation unit. We intentionally omit the shuffle pattern generator, which is listed as second value in every column of shuffling in Table I, for the comparison above since it can be reused for all implementation computation unit groups of the complete SNN. It just needs to be implemented once for the entire network, and with the listed resource requirements, it is virtually negatable.



(a) 1st-order t-test on implementation with disabled masks.

(b) 1st-order t-test on shuffling implementation.

(c) 1st-order t-test on protected implementation.

(d) 2nd-order t-test on protected implementation.

Fig. 10: Results of t-test according to TVLA on the masked implementation based on one million EM-measurements. (a) shows leakage when masks are disabled in (b) masks are also disabled, but shuffling is active, which decreases leakage. The threshold of 4.5 is not crossed, and no leakage is indicated once masks are enabled in (c). However, second-order leakage is detected in (d) for the masked case. We used the protected neuron implementation for all figures, so the timeline is the same, and the leakage at each point in time is comparable between all experiments.

TABLE I: Resource requirements of the unprotected, shuffled and the masked IF-neuron. Numbers for the shuffled implementation are for shuffling groups of five neurons.

Implementation	LUTs	FFs	Randomness (bit)	Cycles
Unprotected	18	19	0	1
Shuffling	391 / 60	85 / 45	0 / 8	2 / 1
1st-order masked	349	389	48	13

First-order masking requires twenty times more resources and increases latency by a factor of 13. The size of the weight memory needs to be doubled to store both shares of the weights. Furthermore, the prevention of glitches necessitates a higher number of registers leading to the lager number of FFs. During each processing of an input event, ten random values of the same word size of the implementation are necessary. The first-order masked implementation's randomness, latency, and resource requirements can be tracked down to the secure addition gadget, which we adapted from Coron et al. [38]. This is a known bottleneck for converting between arithmetic and Boolean masks in secure post-quantum cipher implementations. Future improvements of less resource-hungry and more optimized secure adders can easily be implemented in our modular neuron implementation.

In our design, each neuron signals whenever it is ready to process a new input. Thus, the increased latency in the case of masking does not influence the design of the overall SNN since the same signal can be used. Nevertheless, classification latency will increase by 13 without optimization, such as pipelining in each neuron.

All parallel neurons may use the same randomness since a localized EM-attack is necessary. Therefore, the amount of random bytes is primarily defined by the amount of events during the SNN classification. Typically, SNNs feature a higher number of events compared to shuffling opportunities. Consequently, more randomness is expected to be required for masking than shuffling.

VI. Experimental Results for Countermeasures

We demonstrate the security of our countermeasures against SCA by implementing IF-neurons as depicted in Fig. 8, featuring 4-bit weights and 8-bits for the potential and threshold. Moreover, we show the increase in side-channel resistance of shuffling with five neurons of a small network.

A. Test Vector Leakage Assessment Results

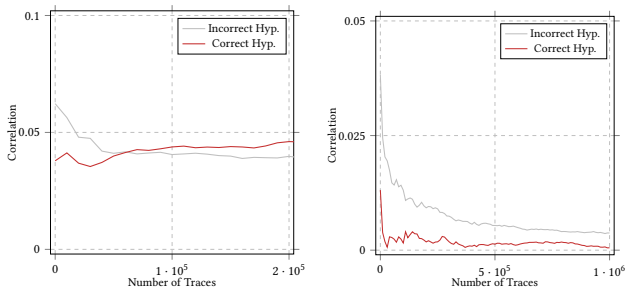
We perform a t-test, where the same input is applied in the fixed-set, i.e., the same weight w is added to the potential v. We use the masked implementation for all tests in Fig. 10 to have identical time steps and to allow comparison between all implementations. Please note that a real unprotected or shuffled implementation will only have 1/13 of the latency depicted here. A random input is applied in the random set, leading to a random weight being processed. Between every measurement, the reset signal is applied to the neurons.

In Fig. 10a, masks are disabled, i.e., no intermediate value is randomly shared. As expected, the resulting t-score exceeds the threshold at several sample points. Figure 10c shows the same implementation. However, now the masks are enabled. Since the threshold of 4.5 is not exceeded, no leakage is detectable, and the implementation is also demonstrated practical secure with up to one million traces. However, multivariate higher-order leakage remains, as depicted in Fig. 10d. The mentioned figure shows the maximum of t-values for the combination of a sample point at a time with all other sample points. Higher-order leakage is present, especially during reset and potential storing between $23\mu s$ and $28\mu s$.

Considering the shuffling countermeasure, a reduction in leakage in Fig. 10b compared to the unprotected implementation (Fig. 10a) is recognizable. Please note that we analyze the shuffled computation unit featuring the masked neuron design without any masks or randomness. This makes the time axis identical for all implementations in Fig. 10. The results are identical to an unmasked computation unit in terms of maximum t-value while the leakage is spread over merely two cycles instead of 13. In the case of five shuffled neurons

in Fig. 10b, the t-score is reduced by about 50% but not eliminated, which is expected for shuffling.

B. Resistance of Countermeasures against CPA



(a) Shuffling (b) Masking

Fig. 11: CPA results on the protected implementation. Shuffling between five neurons is activated in (a) and significantly increases the amount of required traces compared to an unprotected implementation. Enabling masks in (b) outlines a low correlation coefficient for any hypothesis, making a first-order attack impossible.

Compared to the general TVLA approach, we also verify the resistance of the countermeasures against the proposed CPA attack methodology. Figure 11 illustrates the CPA using EM measurements of the protected neuron. The attack is successful on an unprotected implementation with about 200 signal transitions, as Sec. IV demonstrates. In the case of shuffling, we randomly chose the placement of five neurons, i.e., the shuffling factor equals $l = 5$. Theoretically, shuffling requires an attacker to record $l^2 = 25$ more traces than the unprotected implementation. However, the shuffling countermeasure increases the required traces from 200 to $100,000$ traces, which approximately matches a factor of 480. This is much higher than the expected factor of 25, which might be due to the intrinsic noise of SNNs. When shuffling is applied, an adversary will not purely record traces of one neuron at one measurement location but instead of a few neurons. This can effectively decrease the SNR not just as Mangard et al. [32] suggest but by a higher factor.

Attacking the masking countermeasure by CPA is not possible with one million measurements, as Fig. 11b indicates. Neither the correct hypothesis nor any incorrect hypothesis achieves a significantly high value. This behavior validates the TVLA results since masking does not yield any leakage. The results demonstrate that both countermeasures are effective against SCA and applicable to SNNs.

Nevertheless, a designer must effectively tailor a protected implementation with a beneficial trade-off between implementation overhead and security. A hybrid countermeasure of masking and shuffling can be preferable. As our attack is a feed-forward approach, where layers must be attacked after each other, the first layer requires the highest protection. Subsequent layers can only be attacked with the first layer and, thus, do not need such a high level of protection.

C. Discussion

Both proposed countermeasures can protect the implementation against SCA. The implications of the countermeasures on the implementation are described in Sec. V-C. Based on the findings, we summarize the properties of our countermeasures as follows.

Shuffling can offer a simple-to-implement countermeasure that requires less randomness than masking. However, due to the spatial shuffling approach, an implementation needs to be tailored to support the random placement strategy efficiently. Conversely, masking eliminates leakage but is more costly regarding required randomness and hardware resources.

Although not evidenced by experimental results, a joint countermeasure of masking and shuffling can offer a better trade-off between implementation size, randomness, and security. Figure 10b indicates that shuffling can reduce leakage, while masking can eliminate first-order leakage. However, an attacker can possibly perform a second-order attack to retrieve information. Combining both countermeasures can reduce second-order leakage, increasing the number of traces an attacker needs to mount a second-order attack. Consequently, the effort required for a second-order attack is further increased when shuffling is enabled.

Overall, a trade-off between the cost of implementing a countermeasure and the value of the network must be made. Depending on the use case, one or a joint countermeasure can be used to harden a SNN against SCA and protect the IP stored within.

VII. Conclusion

In this work, we outlined the feasibility of horizontal CPA to recover all parameters of feed-forward SNNs utilizing IF neurons by localized EM measurements. Our strategy allows the extraction of about half a million parameters of a SNN within less than an hour. To protect SNNs, we introduced shuffling and masking as countermeasures. Shuffling requires a shuffled computation unit and a shuffling pattern generator but less randomness and resources than masking. In contrast, masking requires a factor of ten more in hardware resources for each neuron and additional randomness during every event processing. The resistance of both countermeasures against CPA is demonstrated with EM measurements.

Acknowledgements

We want to thank the anonymous reviewers for their valuable comments and suggestions on the paper, which helped us improve it. This work was partly funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16KIS1663 (SIPSENSIN). The responsibility for the content of this publication lies with the authors.

References

[1] M. Probst, M. Brosch, and G. Sigl, "Side-channel analysis of integrate-and-fire neurons within spiking neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[2] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, 2020.

[3] L. Batina, S. Bhasin, D. Jap, and S. Picek, "Csi nn: Reverse engineering of neural network architectures through electromagnetic side channel," in *USENIX Security Symposium (USENIX Security)*, pp. 515–532, 2019.

[4] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 197–208, IEEE, 2020.

[5] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[6] A. Bagheri, O. Simeone, and B. Rajendran, "Adversarial training for probabilistic spiking neural networks," in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5, IEEE, 2018.

[7] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing," in *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 17–32, 2014.

[8] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 2, pp. 1–35, 2019.

[9] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[10] Y. Xiang, Z. Chen, Z. Chen, Z. Fang, H. Hao, J. Chen, Y. Liu, Z. Wu, Q. Xuan, and X. Yang, "Open dnn box by power side-channel attack," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2020.

[11] G. Takatoi, T. Sugawara, K. Sakiyama, and Y. Li, "Simple electromagnetic analysis against activation functions of deep neural networks," tech. rep., 2020.

[12] Y. Liu, D. Dachman-Soled, and A. Srivastava, "Mitigating reverse engineering attacks on deep neural networks," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 657–662, IEEE, 2019.

[13] M. Brosch, M. Probst, and G. Sigl, "Counteract side-channel analysis of neural networks by shuffling," in *2022 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, IEEE, Mar 2022.

[14] A. Dubey, A. Ahmad, M. A. Pasha, R. Cammarota, and A. Aysu, "Modulonet: Neural networks meet modular arithmetic for efficient hardware masking," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 506–556, 2022.

[15] K. Athanasiou, T. Wahl, A. A. Ding, and Y. Fei, "Masking feedforward neural networks against power analysis attacks," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 1, pp. 501–521, 2022.

[16] X. Yang, J. Wang, and C. Liu, "Reconstructing neural network topology from firing activity," in *2020 39th Chinese Control Conference (CCC)*, pp. 7106–7111, IEEE, 2020.

[17] L. C. Garaffa, A. Aljuffri, C. Reinbrecht, S. Hamdioui, M. Taouil, and J. Sepulveda, "Revealing the secrets of spiking neural networks: The case of izhikevich neuron," in *2021 24th Euromicro Conference on Digital System Design (DSD)*, pp. 514–518, IEEE, 2021.

[18] B. Goswami, T. Das, and M. Suri, "Experimental investigation of side channel attacks on neuromorphic spiking neural networks," *IEEE Embedded Systems Letters*, 2023.

[19] K. Nagarajan, R. Roy, R. O. Topaloglu, S. Kannan, and S. Ghosh, "Scann: Side channel analysis of spiking neural networks," *Cryptography*, vol. 7, no. 2, p. 17, 2023.

[20] X. Ju, B. Fang, R. Yan, X. Xu, and H. Tang, "An fpga implementation of deep spiking neural networks for low-power and fast classification," *Neural Computation*, vol. 32, no. 1, pp. 182–204, 2020.

[21] A. Upegui, C. A. Peña-Reyes, and E. Sánchez, "A hardware implementation of a network of functional spiking neurons with hebbian learning," in *Biologically Inspired Approaches to Advanced Information Technology: First International Workshop, BioADIT 2004, Lausanne, Switzerland, January 29-30, 2004, Revised Selected Papers 1*, pp. 233–243, Springer, 2004.

[22] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology — CRYPTO '96* (N. Koblitz, ed.), (Berlin, Heidelberg), pp. 104–113, Springer Berlin Heidelberg, 1996.

[23] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems*, pp. 16–29, Springer, 2004.

[24] K. Yoshida, T. Kubota, S. Okura, M. Shiozaki, and T. Fujino, "Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2020.

[25] T. Schneider and A. Moradi, "Leakage assessment methodology," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 495–513, Springer, 2015.

[26] B. Richter, *Advanced Side-Channel Measurement and Testing*. phdthesis, May 2020.

[27] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[28] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

[29] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[32] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*, vol. 31. Springer Science & Business Media, 2008.

[33] S. Faust, V. Grosso, S. M. D. Pozo, C. Paglialonga, and F.-X. Standaert, "Composable masking schemes in the presence of physical defaults & the robust probing model," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 89–120, aug 2018.

[34] S. Belaïd, F. Benhamouda, A. Passelègue, E. Prouff, A. Thillard, and D. Vergnaud, "Randomness complexity of private circuits for multiplication," in *Advances in Cryptology – EUROCRYPT 2016*, pp. 616–648, Springer Berlin Heidelberg, 2016.

[35] G. Cassiers and F.-X. Standaert, "Trivially and efficiently composing masked gadgets with probe isolating non-interference," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2542–2555, 2020.

[36] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, P.-Y. Strub, and R. Zucchini, "Strong non-interference and type-directed higher-order masking," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, oct 2016.

[37] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," *Cryptology ePrint Archive*, 2016.

[38] J.-S. Coron, J. Großschädl, and P. K. Vadnala, "Secure conversion between boolean and arithmetic masking of any order," in *Advanced Information Systems Engineering*, pp. 188–205, Springer Berlin Heidelberg, 2014.

[39] B. Gigerl, R. Primas, and S. Mangard, "Formal verification of arithmetic masking in hardware and software." Cryptology ePrint Archive, Paper 2022/849, June 2022. https://eprint.iacr.org/2022/849.

[40] V. Hadžić and R. Bloem, "Cocoalma: A versatile masking verifier," in *Conference on Formal Methods in Computer-aided Design – FMCAD 2021*, 2021.