# Guarding Machine Learning Hardware Against Physical Side-channel Attacks

ANUJ DUBEY, North Carolina State University, USA
ROSARIO CAMMAROTA, Emerging Security Lab, Intel Corporation, USA
VIKRAM SURESH, Circuit Research Lab, Intel Corporation, USA
AYDIN AYSU, North Carolina State University, USA

Machine learning (ML) models can be trade secrets due to their development cost. Hence, they need protection against malicious forms of reverse engineering (e.g., in IP piracy). With a growing shift of ML to the edge devices, in part for performance and in part for privacy benefits, the models have become susceptible to the so-called physical side-channel attacks.

ML being a relatively new target compared to cryptography poses the problem of side-channel analysis in a context that lacks published literature. The gap between the burgeoning edge-based ML devices and the research on adequate defenses to provide side-channel security for them thus motivates our study. Our work develops and combines different flavors of side-channel defenses for ML models in the hardware blocks. We propose and optimize the *first defense based on Boolean masking*. We first implement all the masked hardware blocks. We then present an adder optimization to reduce the area and latency overheads. Finally, we couple it with a shuffle-based defense.

We quantify that the area-delay overhead of masking ranges from 5.4× to 4.7× depending on the adder topology used and demonstrate a first-order side-channel security of millions of power traces. Additionally, the shuffle countermeasure impedes a straightforward second-order attack on our first-order masked implementation.

CCS Concepts: • **Security and privacy → Side-channel analysis and countermeasures;**

Additional Key Words and Phrases: Side-channel attack, neural networks, masking

**ACM Reference format:**
Anuj Dubey, Rosario Cammarota, Vikram Suresh, and Aydin Aysu. 2022. Guarding Machine Learning Hardware Against Physical Side-channel Attacks. *J. Emerg. Technol. Comput. Syst.* 18, 3, Article 56 (April 2022), 31 pages.
https://doi.org/10.1145/3465377



## 1 INTRODUCTION

**Machine learning (ML)** has become ubiquitous in various domains, such as healthcare [24], automotive [26], and cybersecurity [54], among others. ML has also made significant advances in

This project is supported in part by NSF under Grants No. 1943245 and No. SRC GRC Task 2908.001.
Authors' addresses: A. Dubey and A. Aysu, North Carolina State University, 891, Raleigh, North Carolina, USA, 27606; emails: {aanujdu, aaysu}@ncsu.edu; R. Cammarota, Emerging Security Lab, Intel Corporation, San Diego, California, USA, 92127; V. Suresh, Circuit Research Lab, Intel Corporation, Hillsboro, Oregon, USA, 97124.

1550-4832/2022/04-ART56 $15.00
https://doi.org/10.1145/3465377

terms of performance albeit with increased development costs—e.g., training a recent ML model is estimated to cost over $4.6M [48]. The increasing demand for ML models along with their costly development has created a favorable market for selling ML models directly as a service to the customers [74]. The model provider can either host the trained model on the cloud or deploy it directly on an edge device like a surveillance camera [80]. The recent trend is indeed to directly deploy the ML model on an edge platform for better performance and privacy [103]. These valuable ML model IPs should be kept confidential. High fidelity model extraction attacks have already been proposed in prior literature that use mathematical and/or cryptanalytic approaches to reverse engineer the internals of the model [13, 41]. The shift from cloud-based servers to edge devices has additionally exposed the ML models to the easily applicable and extremely potent physical side-channel attacks [4, 22, 70, 83, 93, 95, 99]. Furthermore, the number of queries for success in side-channels attacks is orders of magnitude lower than that of the mathematical model extraction attacks. ML models may also be more vulnerable to such attacks when compared with cryptographic implementations, because the latter usually has a built-in key-refresh mechanism unlike ML models, which places an upper-bound on the number of traces that can be captured by an adversary [94].

Physical side-channel attacks exploit the information leakage of the secret through physical properties of the device like power-consumption and **electromagnetic emanations (EM)**. The **Differential Power Analysis (DPA)**, for instance, exploits the inherent correlations between the secret-key dependent data being processed and the power consumption of the device [44]. Many cryptographic implementations have been shown to be vulnerable to such attacks since then [15, 51, 67]. Researchers have accordingly proposed effective ways to mitigate such attacks [29, 40, 62, 68, 87]. However, the side-channel analysis research focused primarily on protecting cryptographic implementations, because it was the only domain that required confidentiality. But lately, ML models have also become lucrative targets for side-channel attacks. Stealing the model internals also assists in adversarial attacks that aim to create misclassifications in the model for malicious purposes [100].

Physical side-channel attacks are easily applicable if the adversary has physical access to the device, which is why edge-based ML accelerators are unsurprisingly susceptible to such attacks [4, 22, 93, 99]. With the recent developments in remote physical side-channel attacks, it is also possible to extend reverse engineering of ML models to a multi-tenant cloud-based FPGA setting [57, 77, 101]. However, the research on developing adequate countermeasures is still quite immature. With the latest market research predicting a tremendous growth in the sales of edge-based ML hardware in the coming years [37], there is an urgent need to develop efficient and robust side-channel defenses for ML applications.

The existing works on building countermeasures against power-based side-channel attacks on ML accelerators extend the ones developed for the cryptographic applications. The only two available techniques from our earlier works either utilize a combination of masking and hiding for side-channel resilience or a full Boolean masking approach [21, 22]. Hiding aims to equalize power consumption throughout the execution typically using a precharge and differential logic-based circuit. Masking splits (or encodes) the original secret into multiple statistically independent shares to break the correlation of the secret data with the power consumption. The former approach is cost-conscious but challenging to implement effectively due to the precise control required in back-end flow for the hiding countermeasure. The latter approach of using Boolean masking is relatively easy to implement effectively, because it is an algorithm-level defense and does not require a precise back-end control. However, it incurs higher performance and area overheads when implemented trivially.

In this article, we extend our previous work on Boolean masking that was published at the IEEE/ACM International Conference on Computer Aided Design 2020 [21]. The earlier work

presents the first fully masked ML accelerator design that provides protection against power side-channel attacks. The extensions over the previous work include the following.

- Our previous work used a pipelined ripple-carry architecture for the masked adder design, which had a latency of 100 cycles. In the extension, we design and implement a masked Kogge-Stone architecture for the adder that reduces its latency from 100 cycles to 31 cycles. We quantify that such a change improves the area-delay overheads of the whole design from 5.4× to 4.7× without changing the side-channel security. We quantify the area reduction mainly in terms of the number of **look-up tables (LUTs)**, because LUTs are the limiting factor in FPGAs; flip flops are available in abundance.[1] Section 4.3 discusses the details of the design, Section 6.2.3 illustrates the side-channel resilience, and Section 6.3 presents the implementation results.
- We propose a shuffle countermeasure to improve the side-channel resilience and make it more difficult to conduct a second-order side-channel attack on our first-order masked design. Specifically, the scheme randomize the sequence in which neurons are computed for each layer by adopting the **random start index (RSI)** method originally developed for protecting AES implementations [55, 92]. Shuffling introduces noise in the temporal domain and increases the number of measurements for a successful attack by $N^2$, where $N$ is the number of hidden layer nodes. We discuss the details of the implementation in Section 5. We also demonstrate that shuffling helps to increase even the second-order side-channel security of the design up to at least 3M traces in a low noise setup.
- We also explore an alternative implementation, where we switch our earlier proposed adder using the pipelined Trichina's AND gate [90] with a masked LUT having five inputs. We show that such a design further reduces the area-delay product by 2× but does leak information against a first-order side-channel attack. The leakage, however, is relatively smaller compared to an unprotected design with leakages becoming statistically significant only after capturing 270k measurements. We discuss the details of the implementation and quantify the side-channel leakage in Section 7.1.1.

The rest of this manuscript is organized as follows: Section 2 discusses our assumptions on the adversary and the victim; Section 3 briefly discusses related work on ML model extraction, existing side-channel defenses in cryptography, neural networks and describes our baseline hardware design; Section 4 describes in detail the hardware design of our proposed masked neural network components; Section 5 discusses the implementation of the shuffle countermeasure; Section 6 presents our results on hardware implementation results and the side-channel evaluation; Section 7 discusses potential ways to reduce the costs further or provide provable security; finally, we conclude our article in Section 8.

## 2 THREAT MODEL

We adopt the standard DPA threat model in which an adversary has direct physical access to the target device running inference [4, 22, 45], or can obtain power measurements remotely when the device executes neural network computations [101]. The adversary can control the inputs and observe the corresponding outputs from the device as in chosen-plaintext attacks.

Figure 1 shows our threat model where the training phase is trusted but the trained model is deployed to an inference engine operating in an untrusted environment. The adversary is after the trained model parameters (e.g., weights and biases)—input data privacy is out of scope [93].

[1]The target FPGA is Spartan-6 in which each slice consists of four LUTs but eight flip-flops.

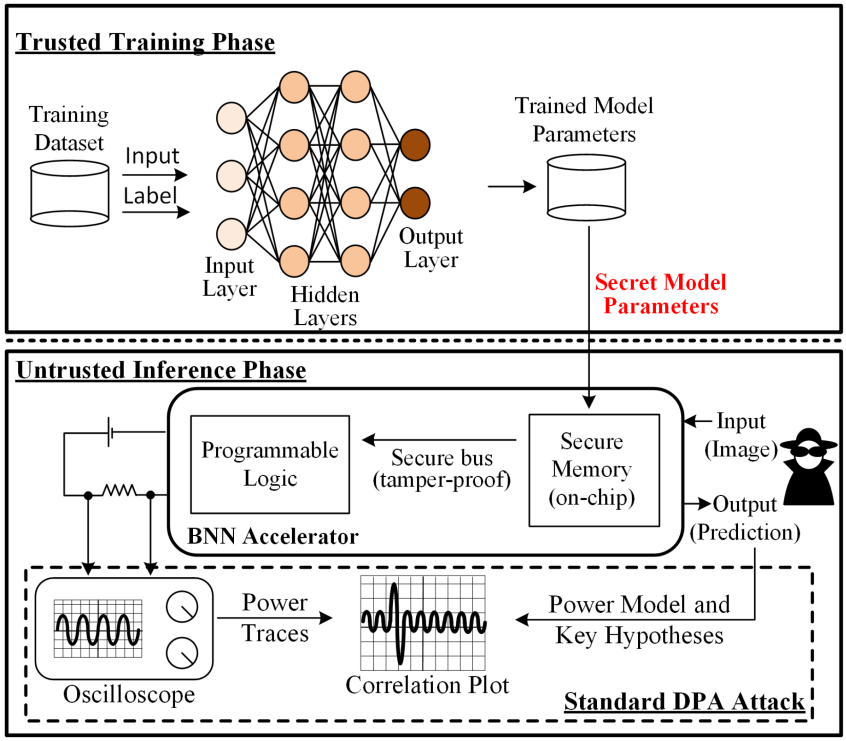


Fig. 1. Standard DPA threat model applied to ML model stealing where the trained neural network is deployed on an edge device running in an untrusted environment.

We assume that the trained ML model is stored in a protected memory and the standard techniques are used to securely transfer it (i.e., bus snooping attacks are out of scope) [9]. The adversary, therefore, has gray-box access to the device, i.e., it knows all the design details up to the level of each logic gate but does not know the trained ML model. We restrict the secret variables to just the parameters and not the hyperparameters such as the number of neurons, following earlier work [22, 42, 89]. In fact, an adversary will still not be able to clone the model with just the hyperparameters if it does not possess the required compute power or training dataset. This is analogous to the scenario in cryptography where an adversary, even after knowing the implementation of a cipher, cannot break it without the correct key.

We target a hardware implementation of the neural network, not software. The design fully fits on the FPGA. Therefore, it does not involve any off-chip memory access and executes with constant-flow in constant time. These attributes make the design resilient to any type of digital (memory, timing, access-pattern, etc.) side-channel attack. However, the physical side-channels like power and EM emanations still exist; we address the power-based side-channel leakages in our work. Other implementation attacks on neural networks such as the fault attacks are out of scope [7, 8].

## 3 BACKGROUND AND RELATED WORK

This section presents related work on the privacy of ML applications, the current state of side-channel defenses, preliminaries on BNNs, and our BNN hardware design.

### 3.1 ML Model Extraction

Recent developments in the field of ML point to several motivating scenarios that demand asset confidentiality. First, training is a computationally intensive process and hence requires the model provider to invest money in high-performance compute resources (e.g., a GPU cluster). The model provider might also need to invest money to buy a labeled dataset for training or label an unstructured dataset. Therefore, knowledge about either the parameters or hyperparameters can provide an unfair business advantage to the user of the model, which is why the ML model should be private.

Theoretical model extraction analyzes the query-response pair obtained by repeatedly querying an unknown ML model to steal the parameters [13, 41, 64, 71]. This type of attack is similar to the class of theoretical cryptanalysis in the cryptography literature. Digital side-channels, by contrast, exploit the leakage of secret-data dependent *intermediate computations* like access-patterns or timing in the neural network computations to steal the parameters [20, 23, 33, 34, 96], which can usually be mitigated by making the secret computations constant-flow and constant-time. Physical side-channels target the leak in the physical properties like CMOS power-draw or electromagnetic emanations that will still exist in a constant-flow/constant-time algorithm's implementation [4, 22, 83, 93, 95, 99]. Mitigating physical side-channels are thus harder than digital side-channels in hardware accelerator design and has been extensively studied in the cryptography community.

### 3.2 Side-channel Defenses

The researchers have proposed numerous countermeasures against DPA. These countermeasures can be broadly classified as either *hiding-based* or *masking-based*. The former aims to make the power-consumption constant throughout the computation by using power-balancing techniques [61, 87, 98]. The latter splits the sensitive variable into multiple statistically independent shares to ensure that the power consumption is independent of the sensitive variable throughout the computation [2, 6, 28, 40, 65, 90].

The security provided by hiding-based schemes hinges upon the precision of the back-end design tools to create a near-perfect power-equalized circuit by balancing the load capacitances and synchronizing their activity across the leakage prone paths. This is not a trivial task and prior literature shows how a well-balanced dual-rail-based defense is still vulnerable to localized EM attacks [36]. By contrast, masking transforms the algorithm itself to work in a secure way by never evaluating the secret variables directly. This keeps the security largely independent of the back-end design and makes masking a favorable choice over hiding.

### 3.3 Neural Network Classifiers

Neural network algorithms learn how to perform a certain task. In the learning phase, the user sends a set of inputs and expected outputs to the machine (a.k.a., training), which helps it to approximate (or learn) the function mapping the input-output pairs. The learned function can then be used by the machine to generate outputs for unknown inputs (a.k.a., inference).

A neural network consists of units called neurons (or nodes) and these neurons are usually grouped into layers. The neurons in each layer can be connected to the neurons in the previous and next layers. Each connection has a weight associated with it, which is computed during the training. The neurons work in a feed-forward fashion passing information from one layer to the next.

The weights and biases can be initialized to be random values or a carefully chosen set before training [66]. These weights and biases are the *critical parameters* that our countermeasure aims to

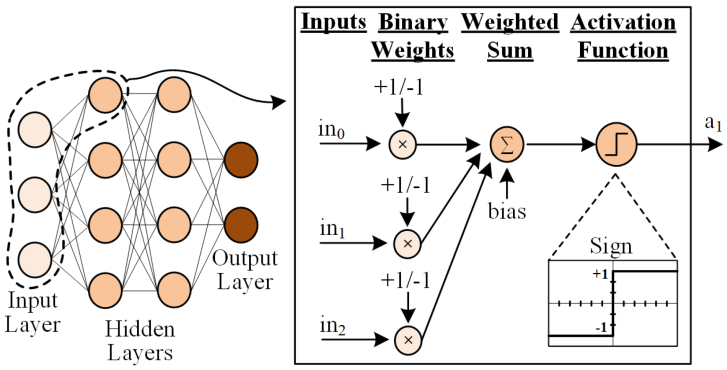


Fig. 2. A typical Binarized Neural Network where the neuron performs weighted summations of the input pixels with binarized weights, and the activation function is a sign function.

protect. During training, a set of inputs along with the corresponding labels are fed to the network. The network computes the error between the actual outputs and the labels and tunes the weights and biases to reduce it, converging to a state where the accuracy is acceptable.

### 3.4 Binarized Neural Networks

Figure 2 depicts the neuron computation in a fully connected BNN. The weights and biases of a neural network are typically floating-point numbers. However, high area, storage costs, and power demands of floating-point hardware do not fare well with the requirements of the resource-constrained edge devices. **Binarized Neural Networks (BNNs)** [35], with their low hardware cost and power needs fit very well in this use-case while providing a reasonable accuracy. BNNs restrict the weights and activations to binary values (+1 and −1), which can easily be represented in hardware by a single bit. This reduces the storage costs for the weights from floating-point values to binary values. The XNOR-POPCOUNT operation implemented using XNOR gates replaces the large floating-point multipliers resulting in a huge area and performance gain [69]. In Figure 2, the neuron in the first hidden layer multiplies the input values with their respective binarized weights. The generated products are added to the bias, and the result is fed to the activation function, which is a sign function that binarizes the non-negative and negative inputs to +1 to −1, respectively. Hence, the activations in the subsequent layer are also binarized.

Prior works have demonstrated the computational efficiency of XNOR-POPCOUNT-based arithmetic operations in DNNs. A prior proposed GPU kernel exploiting the XNOR-POPCOUNT operations achieved 23× faster matrix multiplication compared to a naive baseline implementation [17]. The designed kernel is 3.4× faster than cuBLAS and the MLP runs 7× faster using the XNOR-POPCOUNT kernel compared to the baseline. Well known semiconductor companies like Xilinx, Intel, and Apple are showing interest in BNNs due to these advantages [43, 84, 91]. Owing to their low memory footprint and lightweight nature of operations, BNNs are considered attractive for edge applications such as FPGA-accelerators [25], cryptographic neural network inference systems [50], and for designing low-bitwidth ConvNets [102], among many other applications. Despite their low-bitwidth operations, the accuracy obtained by BNNs is comparable to that obtained by full-precision neural networks. For instance, the accuracy loss of a BNN-ConvNet with 1-bit weights and activations was found to be less than 0.5% compared to a full-precision ConvNet with seven convolutional layers and one dense layer [102] when evaluated on the Google **Street View House Number (SVHN)** dataset. Similarly, another work [49] achieved less than 5% accuracy loss

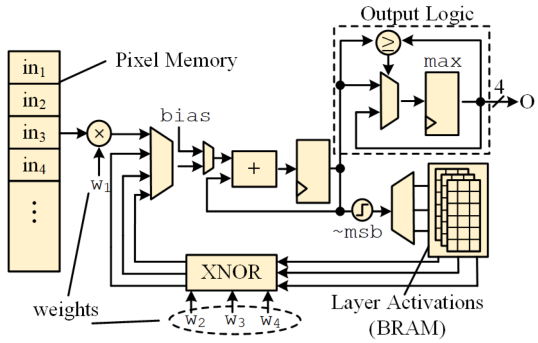


Fig. 3. The hardware design of baseline unmasked BNN. The hardware performs weighted summations using a single adder in the input layer and using XNOR-POPCOUNT logic in the hidden layers. The sign function is implemented by simply inverting the MSB of the weighted summation. The output logic finds the maximum weighted summation out of the 10 output layer nodes sequentially.

on the ImageNet dataset [19], a challenging dataset notorious for its complexity in the computer vision community, using a ResNet-like network built entirely using binary convolution blocks, compared to a full precision network.

### 3.5 Our Baseline BNN Hardware Design

We consider a BNN having an input layer of 784 nodes, three hidden layers of 1,010[2] nodes each, and an output layer of 10 nodes. The 784 input nodes denote the 784 pixel values in the $28 \times 28$ grayscale images of the **Modified National Institute of Standards and Technology (MNIST)** database and 10 output nodes represent the 10 output classes of the handwritten numerical digit.

*3.5.1 Weighted Summations.* We choose to use a single adder in the design and sequentialize all the additions in the algorithm to reduce the area costs. Figure 3 shows our baseline BNN design. The computation starts from the input layer pixel values stored in the Pixel Memory. For each node of the first hidden layer, the hardware multiplies 784 input pixel values one by one and accumulates the sum of these products. The final summation is added with the bias reusing the adder with a multiplexed input and fed to the activation function. The hardware uses XNOR and POPCOUNT[3] operations to perform weighted summations in the hidden layers. The final layer summations are sent to the output logic.

In the input layer computations, the hardware multiplies an 8-bit unsigned input pixel value with its corresponding weight. The weight values are binarized to either 0 or 1 (representing a $-1$ or $+1$, respectively). Figure 4 shows the realization of this multiplication with a multiplexer that takes in the pixel value ($a$) and its 2's complement ($-a$) as the data inputs and weight ($\pm 1$) as the select line. The 8-bit unsigned pixel value, when multiplied by $\pm 1$, needs to be sign-extended to 9-bits, resulting in a 9-bit $2 \times 1$ multiplexer.

*3.5.2 Activation Function.* The activation function binarizes the non-negative and negative inputs to $+1$ and $-1$, respectively, for each node of the hidden layer. In hardware, this is implemented using a simple NOT gate that takes the MSB of the summations as its input.

[2]The number of hidden layer nodes can change based on the type of adder used, which is explained later in the manuscript.
[3]The POPCOUNT operation also involves an additional step of subtracting the number of nodes (1,010) from the final sum, which can be done as part of bias addition step.

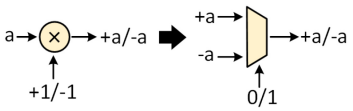


Fig. 4. The multiplication of input pixel $a$ with the weight (+1/−1) is implemented as a 9-bit $2 \times 1$ multiplexer in the baseline BNN design.

*3.5.3 Output Layer.* The summations in the output layer represent the confidence score of each output class for the provided image. Therefore, the final classification result is the class having the maximum confidence score. Figure 3 shows the hardware for computing the classification result. As the adder generates output layer summations, they are sent to the output logic block that performs a rolling update of the max register (*max*) if the newly received sum is greater than the previously computed max. In parallel, the hardware also stores the index of the current max node. The index stored after the final update is sent out as the final output of the neural network. The hardware takes 2.8M cycles to finish one inference.

*Exemplary DPA Attack on Baseline Implementation.* A DPA adversary can target the 20-bit accumulator register at the output of the adder in Figure 3. The register accumulates the sum of the products of image pixels and weights in every cycle. Thus, the power consumption of any of those cycles can be modeled by using a hamming distance (HD) model. For example, if the adversary targets the fourth cycle, the power model $M$ is given as follows:

$$M = HD((w_0 \times p_0 + w_1 \times p_1 + w_2 \times p_2), (w_0 \times p_0 + w_1 \times p_1 + w_2 \times p_2 + w_3 \times p_3)),$$

where $p_i$ and $w_i$ denote the $i$th image pixel and weight, respectively. The number of hypotheses in this case is 16 ($2^4$), since the adversary attacks four weights $w_0$, $w_1$, $w_2$, and $w_3$. Since the hardware adds the bias in the 785th cycle, the adversary needs to extract the 784 weights first and then construct the power model for the sum with bias to attack it.

## 4 FULLY MASKING THE NEURAL NETWORK

This section discusses the hardware design and implementation of all components in the masked neural network. Prior work on masking of neural networks shows that arithmetic masking alone cannot mask integer addition due to a leakage in the sign-bit [22]. Hence, we apply gate-level *Boolean* masking to perform integer addition in a secure fashion. We express the entire computation of the neural network as a sequence of AND and XOR operations and apply gate-level masking on the resulting expression. XORs, being linear, do not require any additional masking, and AND gates are replaced with secure, Trichina style AND gates [90]. Furthermore, we design specialized circuits for BNN's unique components like Masked Multiplexer and Masked Output Layer.

We first explain the notations in equations and figures. Any variable without a subscript or superscript represents an N-bit number. We use the subscript to refer to a single bit of the N-bit number. For example, $a_7$ refers to the 8th bit of $a$. The superscript in masking refers to the different secret shares of a variable. To refer to a particular share of a particular bit of an N-bit number, we use both the subscript and the superscript. For example, $a_4^1$ refers to the second Boolean share of the 5th bit of $a$. If a variable only has the superscript (say $i$), then we are referring to its full N-bit $i$th Boolean share; N can also be equal to 1, in which case $a$ is simply a bit. r (or $r_i$) denotes a fresh, random bit. The operation $\oplus$ represents a bitwise XOR of operands.

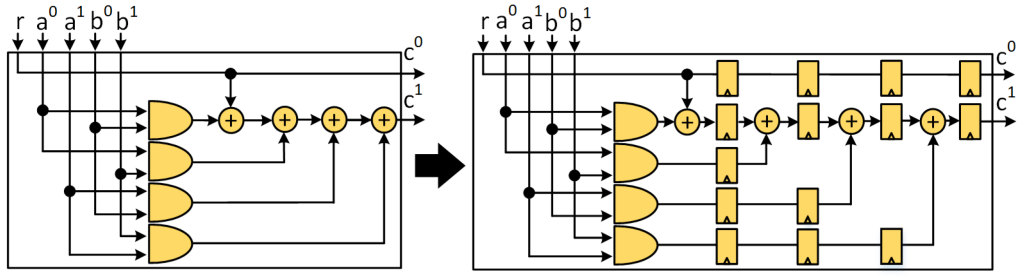


Fig. 5. Trichina's AND Gate implementation: glitch-prone (left) and glitch-resistant (right). Flip-flops synchronize arrival of asynchronous gate outputs at XOR gates' inputs to reduce glitches.

### 4.1 A Glitch-Resilient Trichina's AND Gate

Among the closely related masking styles [72], we chose to implement Trichina's method due to its simplicity and implementation efficiency. Figure 5 (left) shows the basic structure and functionality of the Trichina's gate, which implements a 2-bit, masked, AND operation of $c = a \cdot b$. Each input ($a$ and $b$) is split into two shares ($a^0$ and $a^1$ s.t. $a = a^0 \oplus a^1$, $b^0$ and $b^1$ s.t. $b = b^0 \oplus b^1$). These shares are sequentially processed with a chain of AND gates initiated with a fresh random bit ($r$). A single AND operation thus uses three random bits. The technique ensures that output is the Boolean masked output of the original AND function, i.e., $c = c^0 \oplus c^1$, while all the intermediate computations are randomized.

Unfortunately, the straightforward adoption of Trichina's AND gate can lead to information leakage due to glitches [62]. For instance, in Figure 5 (left) if the products $a_0 \cdot b_0$ and $a_0 \cdot b_1$ reach the input of second XOR gate before random mask $r$ reaches the input of first XOR gate, the output at the XOR gate will evaluate (glitch) to $(a_0 \cdot b_0) \oplus (a_0 \cdot b_1) = a_0 \cdot (b_0 \oplus b_1)$ temporarily, which leads to secret value $b$ being unmasked. Therefore, we opted for an extension of the Trichina's AND gate by adding flip-flops to synchronise the arrival of inputs at the XOR gates (see Figure 5 right). The only XOR gate not having a flip-flop at its input is the leftmost XOR gate in the path of $c_1$, which is not a problem, because a glitching output at this gate does not combine two shares of the same variable. Similar techniques have been used in past [3]. Masking styles like the Threshold gates [52, 53, 86] may be considered for even stronger security guarantees, but they will add further area-performance-randomness overhead.

### 4.2 Masked Ripple Carry Adder

We adopt the ripple-carry style of implementation for the adder first. It is composed of N 1-bit full adders where the carry-out from each adder is the carry-in for the next adder in the chain, starting from LSB. Therefore, ripple-carry configuration eases parameterization and modular design of the Boolean masked adders.

*4.2.1 Design of a Masked Full Adder.* A 1-bit full adder takes as input two operands and a carry-in and generates the sum and the carry, which are a function of the two operands and the carry-in. If the input operand bits are denoted by $a$ and $b$ and carry-in bit by $c$, then the Boolean equation of the sum $S$ and the carry $C$ can be described as follows:

$$S = a \oplus b \oplus c, \tag{1}$$

$$C = a \cdot b \mid b \cdot c \mid c \cdot a. \tag{2}$$

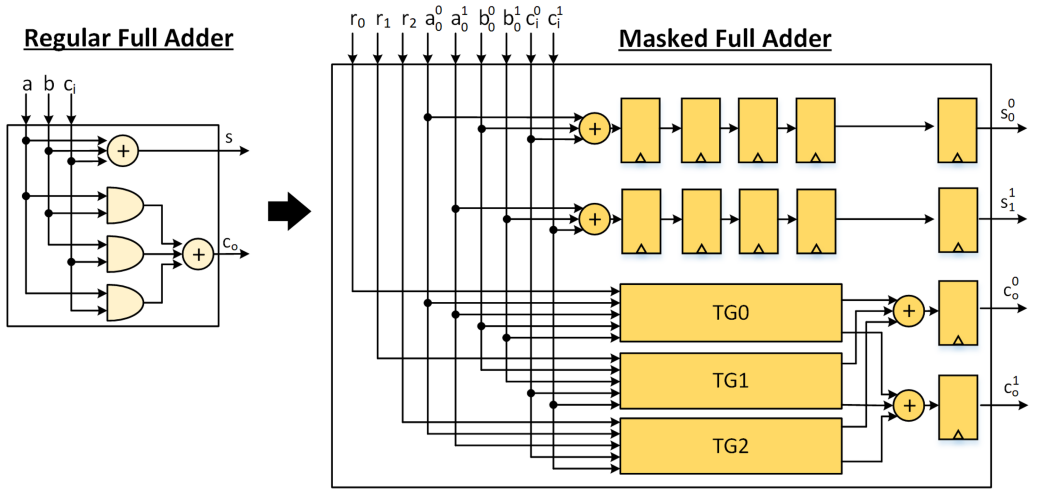


Fig. 6. An unmasked Full Adder design (left) and its gate-level masking using Trichina AND Gates (right) that has a latency of 5 cycles.

However, the non-linear OR operation in the carry function is usually replaced with the linear XOR operator to simplify the masking of carry:

$$C = a \cdot b \oplus b \cdot c \oplus c \cdot a. \tag{3}$$

Figure 6 shows the regular, 1-bit full adder (on the left), and the resulting masked adder with Trichina's AND gates (on the right). In the rest of the subsection, we will discuss the derivation of the masked full adder equations.

First step is to split the secret variables ($a$, $b$, and $c$) into Boolean shares. The hardware samples a fresh, random mask from a uniform distribution and performs XOR with the original variable. If we represent the random masks as $a^0$, $b^0$, and $c^0$, then the masked values $a^1$, $b^1$, and $c^1$ can be generated as follows:

$$a^1 = a \oplus a^0,\ b^1 = b \oplus b^0,\ c^1 = c \oplus c^0. \tag{4}$$

A masking scheme always works on the two shares independently without combining them at any point in the operation, because that will reconstruct the secret and create a side-channel leak.

The function of sum-generation is linear, making it easy to directly and independently compute the Boolean shares of $S$:

$$S = S^0 \oplus S^1,$$

where

$$S^0 = a^0 \oplus b^0 \oplus c^0,\ S^1 = a^1 \oplus b^1 \oplus c^1.$$

Unlike the sum-generation, carry-generation is a non-linear operation due to the presence of an AND operator. Hence, the hardware cannot directly and independently compute the Boolean shares $C^0$ and $C^1$ of $C$. We use the Trichina's construction explained in Section 4.1 to mask carry-generation.

The hardware uses three Trichina's AND gates to mask the three AND operations in Equation (3) using three random masks. This generates two Boolean shares from each Trichina AND operation. At this point, the expression is linear again, and therefore, the hardware can redistribute the terms, similar to the masking of sum operation.

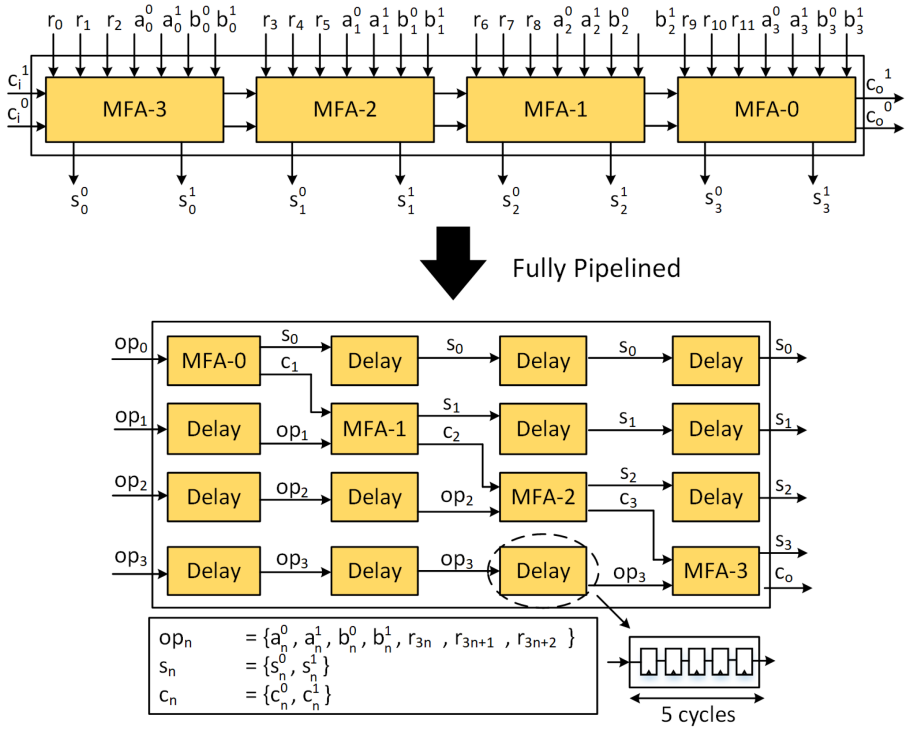


Fig. 7. Modular design of a masked 4-bit adder using Masked Full Adders (top) and its pipelined version (bottom). Pipelining ensures a throughput of 1 summation per cycle.

In the following equations, we use $TG(x, y, r)$ to represent the product $x \cdot y$ implemented via Trichina's AND Gate as illustrated in the following equation:

$$x \cdot y = TG(x, y, r) = m^0 \oplus m^1,$$

where $m^0$ and $m^1$ are the two Boolean shares of the product. Replacing each AND operation in Equation (3) with TG, we can write

$$TG(a, b, r_0) = d^0 \oplus d^1, \tag{5}$$

$$TG(b, c, r_1) = e^0 \oplus e^1, \tag{6}$$

$$TG(c, a, r_2) = f^0 \oplus f^1, \tag{7}$$

where $d^0$, $d^1$, $e^0$, $e^1$, $f^0$, and $f^1$ are the output shares from each Trichina Gate. From Equations (3), (5), (6), and (7), we get

$$carryout = TG(a, b, r_0) \oplus TG(b, c, r_1) \oplus TG(c, a, r_2).$$

Replacing the TGs from Equations (5), (6), and (7) and rearranging the terms, we get

$$carryout = (d^0 \oplus e^0 \oplus f^0) \oplus (d^1 \oplus e^1 \oplus f^1),$$

which can also be written as a combination of two Boolean shares $C^0$ and $C^1$, where

$$C^0 = d^0 \oplus e^0 \oplus f^0, \; C^1 = d^1 \oplus e^1 \oplus f^1.$$

Therefore, we create a masked full adder that takes in the Boolean shares of the two bits to be added along with a carry-in and gives out the Boolean shares of the sum and carry-out.

*4.2.2 The Modular Design of Pipelined N-bit Full Adder.* The masked full adders can be chained together to create an N-bit masked adder that can add two masked N-bit numbers. Figure 7 (top) shows how to construct a 4-bit masked adder as an example. We pipeline the N-bit adder to yield a throughput of one by adding registers between the full-adders corresponding to each bit (see Figure 7 (bottom)).

### 4.3 Masked Kogge Stone Adder

We also propose and implement the masked design for a baseline Kogge Stone adder (KSA) architecture. KSA is a type of parallel prefix adder that has a (logarithmically) lower latency compared to that of the ripple carry adder. KSA builds on the concept of carry look ahead. It starts by computing the *generate* ($g_i$) and *propagate* ($p_i$) bits for each position given by the following equations:

$$g_i = a_i \cdot b_i, \tag{8}$$

$$p_i = a_i \oplus b_i, \tag{9}$$

where $a_i$ and $b_i$ are the $i$th bits of the operands. The generate bit being 1 implies that the carry will definitely be asserted at that position. The propagate bit being 1 implies that the carry will only be asserted if there is an incoming carry from the previous step. Therefore, the carry can only be asserted if the generate bit is 1 or the propagate bit is 1 with an incoming carry from the previous position.

This concept of generates and propagates for a single position can be extended to compute the so-called *group generate* and *group propagate* bits that denote if a group of bits do generate or propagate a carry. For example, the following two equations illustrate how to compute the group generate ($G_{1:0}$) and propagate ($P_{1:0}$) for the group of least significant two bits from the individual generates and propagates $g_0, p_0, g_1$, and $p_1$:

$$G_{1:0} = g_1 \oplus p_1 \cdot g_0,$$

$$P_{1:0} = p_1 \cdot p_0.$$

Larger groups can be created from smaller groups by combining them in a similar fashion. Let $G_{i:k}$, $P_{i:k}$, $G_{k:j}$, and $P_{k:j}$ represent the group generate and propagate bits for the group of bits from $i^{th}$ to $k^{th}$ position, and $k$th to $j$th position, respectively. The generic equation to combine these quantities and get a group generate and propagate for the combined group from $i$th to $j$th bit is given below:

$$G_{i:j} = G_{i:k} \oplus P_{i:k} \cdot G_{k:j}, \tag{10}$$

$$P_{i:j} = P_{i:k} \cdot P_{k:j}. \tag{11}$$

Figure 8 shows the baseline KSA schematic for 8-bit operands.[4] First, the KSA computes the sum bits without carry by element-wise XORing of the operands. Next, it keeps combining the generates and propagates in parallel until all the group generates and propagates for each position represents the group from that position until the least significant bit, effectively creating a direct dependence of each carry to the carry-in of the adder. Finally, the adder computes the carry at each position directly from the final group generate, group propagate, and the carry-in and XORs it with the sum without carry to calculate the result.

[4]We use a 20-bit KSA in the actual design, but illustrate an 8-bit KSA's schematic here for simplicity.

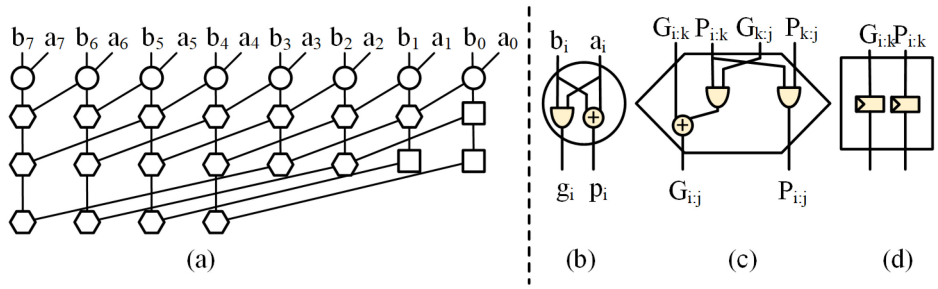


Fig. 8. Panel (a) shows the topology of an 8-bit KSA and the location of the three different constituents in the design using circles, hexagons, and squares. Panels (b), (c), and (d) describe the internals of these three constituents, respectively.

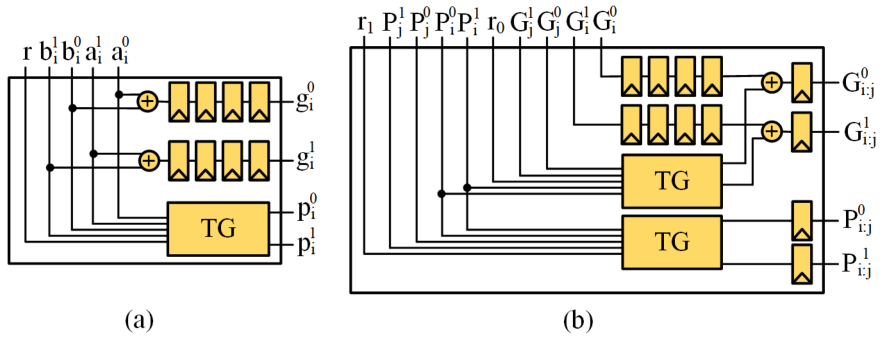


Fig. 9. Part (a) describes the masking of the individual generate and propagate blocks. Part (b) shows the masked hardware for the group generate and propagate blocks.

We have designed a masked version of the KSA. Figure 9 shows the schematic of this design. The adder in the masked design receives two Boolean shares of each operand. The function to compute the sum without carry is an XOR operation, which is linear. Therefore, the adder directly and independently computes the Boolean shares of the sum without carry. Similarly it also computes the Boolean shares of the individual propagate bits, because they only involve an XOR operation (Equation (9)). The computation of the individual generate bits, however, involve the non-linear AND operation. The adder thus replaces the regular AND gates with the synchronised Trichina's AND gate discussed in Section 4.1 to produce the Boolean shares of the generate bits. The produced generate and propagate bits are propagated down the adder tree and combined in each stage using the combination function from Equations (10) and (11). The combination function uses 1 XOR and 2 AND gates. The XOR, again, does not require any additional masking, whereas two AND gates are replaced by two synchronised Trichina's AND gates. In the final stage, the adder XORs the respective shares of the sum without carry and the generate bits to calculate the Boolean shares of the actual sum.

In the masked KSA architecture the adder's latency is reduced from 100 cycles of masked RCA to 30 cycles. The reduced latency directly helps to reduce the area of the throughput-optimization circuitry that is discussed in Section 4.6. The adder latency decides the number of buffers needed to store the concurrent partial summations and the data width of the corresponding demultiplexer and multiplexer blocks. For instance, in the RCA-based design the hardware uses 101 buffers to

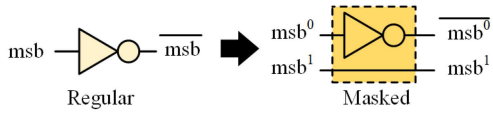


Fig. 10. The unmasked activation function (left) is implemented as a NOT gate and the masked version (right) receives two Boolean shares of MSB from masked adder and inverts one of them.

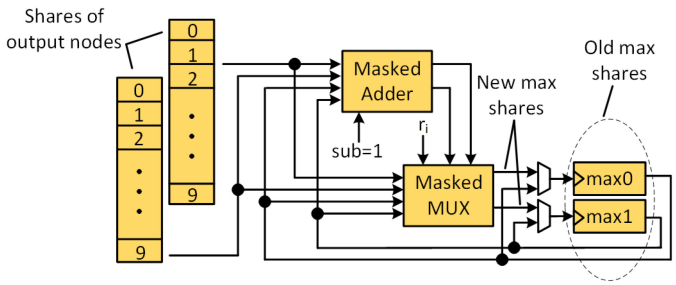


Fig. 11. Masking of the Output Layer that uses a masked subtractor and a masked multiplexer to find the node with the maximum confidence score among the 10 output nodes.

compute the summations for 101 nodes concurrently, because the adder only produces 1 output in 101 cycles.

### 4.4 Masking of Activation Function

The baseline hardware implements the activation function as an inverter as discussed in 3.5.2. In the masked version, the MSB output from the adder is a pair of Boolean shares. To perform NOT operation in a masked way, the hardware simply inverts one of the Boolean shares as Figure 10 shows.

### 4.5 Masking the Output Layer

The hardware stores the 10 output layer summations in the form of Boolean shares. To determine the classification result, it needs to find the maximum value among the 10 masked output nodes. Specifically, it needs to *compare* two signed values expressed as Boolean shares. We transform the problem of *masked comparison* to *masked subtraction*.

Figure 11 shows the hardware design of the masked output layer. The hardware subtracts each output node value from the current maximum and swaps the current maximum (old max shares) with the node value (new max shares) if the MSB is 1 using a masked multiplexer. An MSB of 1 signifies that the difference is negative and hence the new sum is greater than the latest max. Instead of building a new masked subtractor, we reuse the existing masked adder to also function as a subtractor through a *sub* flag, which is set while computing max. In parallel, the hardware uses one more masked multiplexer-based update-circuit to update the Boolean shares of the index corresponding to the current max node (not shown in the figure). This is to prevent known-ciphertext attacks, ciphertext being the classification result in our case. Finally, the Masked Output Logic computes the classification result in the form of (Boolean) shares of the node's index having the maximum confidence score.

Subtraction is essentially adding a number with the 2's complement of another number. 2's complement is computed by taking bitwise 1's complement and adding 1 to it. A bitwise inverse is implemented as an XOR operation with 1 and the addition of 1 is implemented by setting the

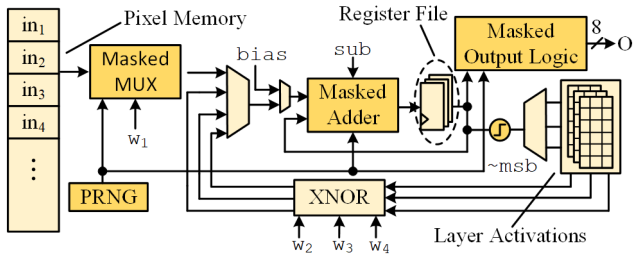


Fig. 12. Hardware Design of the Fully Masked Neural Network. The components related to masking are shown in dark yellow. The register file helps in throughput optimization.

initial carry-in to 1. The additional XOR, being a linear operator, changes nothing with respect to the masking of the new adder-subtractor circuit.

### 4.6 Scheduling of Operations

We optimize the scheduling in such a way that the hardware maintains a throughput of 1 addition per cycle. The latency of the masked 20-bit RCA adder is 100 cycles. Therefore, the result from the adder will only be available after 101 cycles[5] from the time it samples the inputs. The hardware cannot feed the next input in the sequence until the previous sum is available because of the data dependency between the accumulated sum and the next accumulated sum. This incurs a stall for 101 cycles leading to a total of $784 * 101 = 79{,}184$ cycles for each node computation. That is a 784× performance drop over the unmasked implementation with a regular adder.

We solve the problem by finding useful work for the adder that is independent of the summation in-flight, during the stalls. We observe that computing the weighted summation of one node is completely independent of the next node's computation. The hardware utilizes this independence to improve the throughput by starting the next node computation while the result for the first node arrives. Similarly, all the nodes up till 101 can be computed upon concurrently using the same adder and achieve the exact same throughput as the baseline design. This comes at the expense of additional registers (see Figure 12[6]) for storing 101 summations[7] plus some control logic but a throughput gain of 784× (or 1,010× in hidden layers) is worthwhile. The optimization only works if the number of next-layer nodes is greater than, and a multiple of 101. This restricts optimizing the output layer (of 10 nodes) and contributes to the 3.5% increase in the latency of the masked design.

The masked KSA design extension that we propose has a latency of only 31 cycles. This directly impacts the size of the throughput optimization circuit. Since the adder produces a result in every 32[8] cycles, the hardware requires a register file of only 32 entries to store the parallel partial summations and accordingly a multiplexer and demultiplexer of width 32. Hence, we reduce the area cost of the throughput optimization circuitry by 3×, which is coherent with the synthesis results shown in Table 2.

[5]Need an additional cycle for the accumulator register as well.
[6]The register file also has a demultiplexing and multiplexing logic to update and consume the correct accumulated sum in sequence, which is not shown for simplicity.
[7]This is why we use 1010 neurons, which is a multiple of 101, in the hidden layers.
[8]Need an additional cycle for the accumulator register.

## 5 SHUFFLING THE NEURAL NETWORK COMPUTATIONS

Shuffling is a commonly used technique to defend against side-channel attacks [92]. The premise of shuffling is to introduce noise in the temporal domain by either randomizing the order of independent operations and/or by adding random delays within the algorithm. In DPA attacks, the adversary tries to correlate its hypotheses with the side-channel measurements at a fixed point of interest over multiple measurements assuming that the targeted operation happens *at the same time in each trace*. Since in a shuffled implementation the targeted operation will happen at the same time instance in a small set of traces, the number of traces needed for a successful attack increases. Prior work by Mangard et al. demonstrates that the number of traces required for a successful attack is $p^2$ if the probability of occurrence of the operation of attack is $1/p$ [51]. Therefore, shuffling would increase the number of measurements needed to run a successful side-channel attack by $p^2$. Shuffling improves the side-channel resilience in general for all orders due to the temporal perturbation introduced. However, we note that our proposed masking scheme already provides first-order side-channel security and thus, the adversary will naturally try a second-order attack as the next step. Therefore, shuffling makes it more difficult to conduct a second-order side-channel attack on our first-order masked design.

Earlier studies have adopted shuffling as a countermeasure against physical side-channel attacks on both software and hardware implementations of cryptographic primitives [92]. Depending on the performance and area budget, the shuffle is implemented either as **random permutation (RP)** or as a **random starting index (RSI)** [55]. RP-based shuffling typically uses the Fisher-Yates shuffle algorithm[9] that generates unbiased permutations of a given sequence with $n$ elements leading to $n!$ possible permutations. In RSI, the original sequence is cyclically rotated by a random chosen amount, which is the starting index, leading to $n$ possible permutations. We identify that the computation of the activation value of each node in a layer is independent of other nodes. This allows the hardware to shuffle the order of computation of activation values, which is by default done sequentially in the unshuffled implementation. The RSI-based shuffle is almost free in hardware, because only the starting read address of the memory storing the parameters needs to be initialized with a random index. This makes RSI a very good candidate for a low area-budget design. Given the area overheads and constant-time enforcement challenges of Fisher-Yates shuffle, we have opted for applying the RSI in this work.

RSI-based shuffling enables the number of possible permutations to be equal to the number of nodes in a hidden layer, i.e., 1,024. Based on the analysis by Mangard et al. [51], the number of traces for a successful attack increases by $1,024^2\times$, i.e., about six orders-of-magnitude. The large number of independent operations in neural networks helps to provide a significantly higher side-channel security using RSI, compared to that in block ciphers, which is typically in two orders-of-magnitude due to fewer independent operations, for instance, the number of SBox computations. Implementing RP-based shuffling offers a better security but has significant area costs due to the following reasons. Every step of the Fisher-Yates shuffle requires generating a uniform random number within a dynamically changing range. However, PRNGs typically generate random numbers only within ranges that are powers of 2. One way to confine the generated random number within the required range is to perform a modulo but that leads to biased permutations [47]. This is a well-studied problem and one way to solve it is by discarding certain random numbers that create the bias (a.k.a. rejection sampling) that makes the algorithm variable-time, and possibly introduce timing-based side-channel vulnerabilities. The rejection sampler and modulo operation with non-powers of 2 incur significant area overheads when implemented on hardware.

[9]Also known as the Knuth algorithm.

In the shuffled neural network implementation, the hardware generates a random index to start the node computations in a layer. The hardware then evaluates all the nodes sequentially from the randomly chosen node until the last node and then the remaining nodes from the first node until the starting node. The hardware can further shuffle the reads of the input and hidden layer nodes during weighted summation, because addition is commutative. However, we do not chose to implement that shuffling in this work, because the number of input layer nodes is 784, which is not a power of 2 and hence requires a variable time random number generator for an unbiased generation of numbers between 0 and 783 [47].

## 6 RESULTS

In this section, we describe the hardware setup used to implement the neural network and capture power measurements, the leakage assessment methodology that we follow to evaluate the security of the proposed design, and the hardware implementation results.

### 6.1 Hardware Setup

We use Verilog for the front-end design and Xilinx ISE 14.7 for the back-end. We use the DONT_TOUCH synthesis attribute and disable the tool options like LUT combining, register re-ordering, and so on, to prevent any optimization in the masked components. These are standard practices in implementing masking schemes on FPGAs.

Our side-channel evaluation platform is the SAKURA-G FPGA board [31]. It hosts Xilinx Spartan-6 (XC6SLX75-2CSG484C) as the main FPGA that executes the neural network inference. An on-board amplifier amplifies the voltage drop across a 1Ω shunt resistor on the power supply line. We use Picoscope 3206D [85] as the oscilloscope to capture the measurements from the board. We conduct different experiments with varying design and sampling frequencies to shorten the time of acquisition and evaluation and to also validate a sound measurement setup.

(1) We use the design and sampling frequencies of 24 and 125 MHz, respectively, for the experiments presented in the conference version. Since we evaluate the complete inference time window for `Variant-I`, we had to increase the design frequency to shorten the inference latency and accelerate the validation process.
(2) We use the design and sampling frequencies of 1.5 and 500 MHz, respectively, to evaluate the synchronized Trichina's AND gate, `Variant-II`, and `Variant-III`. Synchronized Trichina's AND gate is a small hardware primitive with a latency of only four clock cycles, and we only capture small time windows to evaluate variants `II` and `III`. Therefore, we can decrease the design frequency without any significant time overheads in the leakage evaluation process.

We discuss more on how the high latency of the design poses challenges in evaluation in Section 7.2.

We use Riscure's Inspector SCA [75] software to communicate with the board and initiate a capture on the oscilloscope. By default, the Inspector software does not support SAKURA-G board communication. Hence, we develop our own custom software modules to automate the setup. The modules implement the communication protocol and perform the register reads/writes on the FPGA to start the inference and capture the result.

### 6.2 Leakage Evaluation

We perform the leakage assessment of the proposed design using the non-specific fixed versus random t-tests, which is a common and generic way to assess the side-channel vulnerability in a given implementation [5]. Each input is an image consisting of 784 8-bit unsigned pixels. Since we run a non-specific TVLA test, our setup first selects an image to be used as the fixed image. Then,

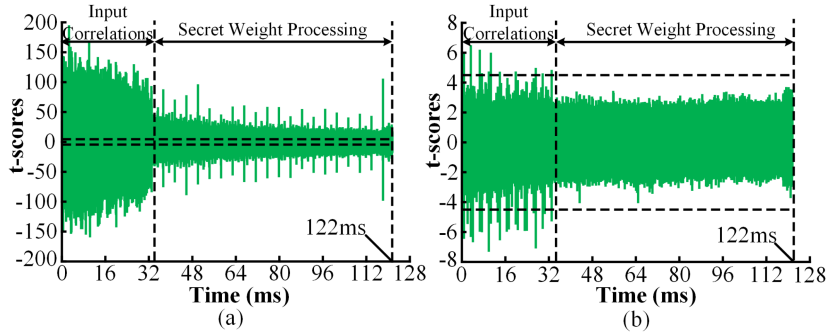


Fig. 13. TVLA results of the unmasked (a) and masked (b) RCA-based implementation. The results clearly show that the unmasked design is insecure, whereas the masked design is secure with 99.99% confidence (t-scores always below ±4.5).

it sends fixed and random images to the device in a randomly interspersed fashion as prescribed by Schneider et al. in a prior work [78]. A t-score within the threshold range of ±4.5 implies that the power traces do not leak any information about the data being processed with up to 99.99% confidence. The measurement and evaluation is quite tedious, and we refer the reader to Section 7.2 for further details. We demonstrate the security up to 2M traces, which is much greater than the first-order security of the currently best-known defense [22].

**Pseudorandom Number Generators (PRNG)** produce the fresh, random masks required for masking. We choose TRIVIUM [11] as the PRNG, which is a hardware implementation friendly PRNG specified as an International Standard under ISO/IEC 29192-3, but any cryptographically secure PRNG can be employed. TRIVIUM generates $2^{64}$ bits of output from an 80-bit key; hence, the PRNG has to be re-seeded before the output space is exhausted. Note that this randomness generation is not a unique problem for masking NN and do exist in masking crypto circuits; hence, we adopt their approach.

Next, we present the security evaluation for three design variants. `Variant-I` is the design that uses only the masked RCA, which is taken from the shorter, conference version of this article [21]. `Variant-II` is the masked KSA-based design that we propose as part of our extensions. `Variant-III` incorporates the shuffling of node computations as well along with the masking of the KSA, which is also a part of the extensions.

*6.2.1 First-order Tests on* `Variant-I`. First, we conduct a first-order t-test on the design with PRNGs disabled, which is equivalent to an unmasked (baseline/unprotected) design. Figure 13(a) shows the result of this experiment, where we clearly observe significant leakages throughout the execution, since the t-scores are greater than the standard threshold of ±4.5. Next, we conduct the same test, but with PRNGs enabled, which is the masked design. Figure 13(b) shows the results for this case, where we observe that the t-scores never cross the threshold of ±4.5 except a small portion in the start of the plot.

The leakages in the initial portion of the plot are due to input correlations during the input layer computations. The hardware loads the input pixel after every 101 cycles and feeds it to the masked multiplexer. However, the secret variable is the *weight* and not the input pixel, which is never exposed, because the masked multiplexer randomises the output using a fresh, random mask.

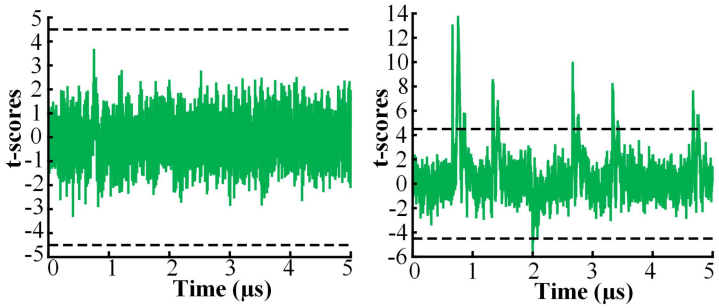


Fig. 14. First-order (left) and second-order (right) t-tests on Trichina's AND gate at a low design frequency of 1.5 MHz and sampling frequency of 500 MHz.

*6.2.2 High Precision First and Second-order Tests on* `Variant-I`. We performed univariate second-order t-test on the fully masked design [78] but 1M traces were not sufficient to reveal the leakages. Due to the extremely lengthy measurement and evaluation times it was infeasible to continue the test for more traces. Therefore, we perform first and second-order evaluation on the isolated synchronized Trichina's AND gate, which is one of the main building blocks of the design. We reduce the design frequency to 1.5 MHz to increase the accuracy of the measurement and prevent any clock cycle aliasing. The **signal-to-noise ratio (SNR)** for a single gate was not sufficient to see leakage even at 10M traces; hence, we amplify the SNR by instantiating 32 independent instances of the Trichina's AND gate in the design, driven by the same inputs. We present the results for this experiment in Figure 14 that shows no leakage in the first-order t-test but significant leakages in the second-order t-tests for 500k traces. Thus, the successful second-order t-test validates the correctness of our measurement setup and the first-order masking implementation.

*6.2.3 First-order Tests on* `Variant-II` *and* `Variant-III`. The tests involve (i) increasing the sampling frequency of the oscilloscope to capture potential glitch-caused leakages more effectively and (ii) evaluating the canonical layers' computation rather than the entire scheme to achieve a practical validation. Note that this is a standard practice in side-channel analysis—e.g., only the first few exponentiations are analyzed in a 1,024-bit RSA scheme [12]. We present our evaluations for two short time windows out of the overall inference, which involve unique computations, for `Variant-II`. The setup captures the first time window when hardware transitions from the input layer computations to the first hidden layer computations (Figure 15). The second time window is the complete evaluation of the output layer, which involves the computation of maximum confidence using the newly proposed masked KSA (Figure 16). Figure 15(a) shows that the first-order t-test with PRNGs disabled clearly leaks during the computations of both the input and hidden layer. However, in the masked design, i.e., with PRNGs switched on, the t-scores never cross the threshold of ±4.5 during the hidden layer computations. The results for `Variant-III` are similar with an additional decrease in the t-scores because of increased temporal noise due to shuffling. Figure 15(c) shows the TVLA results for `Variant-III`.

Figure 16 shows the TVLA results for the entire output layer computation with PRNGs off and PRNGs on. The results again demonstrate that the unmasked design leaks significantly, whereas the masked design's leakage is within the standard threshold of ±4.5 throughout the computation of the masked classification result.

*6.2.4 Evaluation of Frequency Traces.* Converting the power traces to frequency domain is one potential way to nullify the effect of shuffling, because shuffling only introduces noise in the time

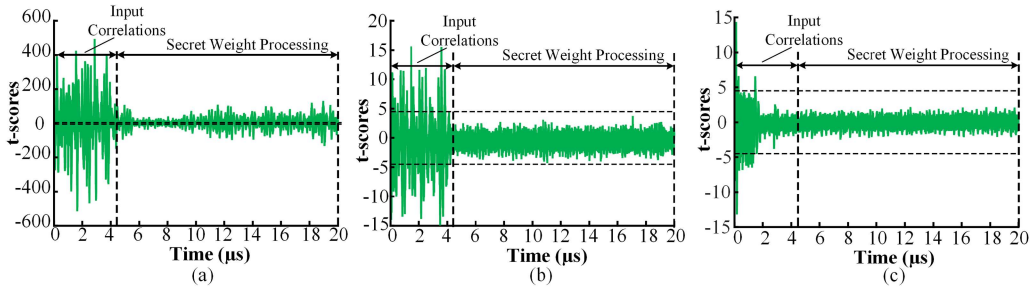


Fig. 15. First-order t-test results of the KSA-based design with PRNGs off (a), PRNGs on (b), and PRNGs on coupled with RSI shuffling of the node computations (c). Panel (a) shows clear leakages throughout the execution, panel (b) only shows leakages during the input layer computations, and panel (c) shows a significantly lower leakage during the input layer computations as well due to the shuffle.

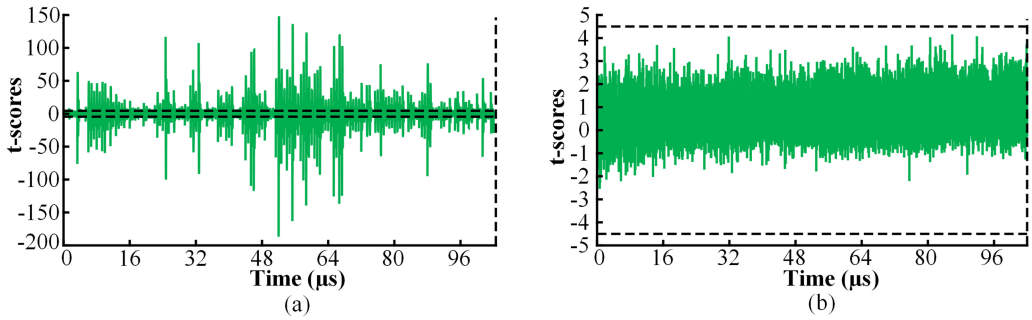


Fig. 16. First-order t-test results of the output layer computations in the KSA-based design with PRNGs off (a), PRNGs on (b). Panel (a) shows clear leakages throughout the execution, whereas panel (b) shows that the t-scores stay confined to the threshold of ±4.5.

domain. Thus, the adversary may conduct a side-channel attack on the frequency domain traces to extract the secret information. However, masking the design reduces the leakages in the frequency domain as well and resists a first-order DPA on the frequency traces. We conduct a TVLA test on the frequency domain traces of `Variant-III` to quantify the side-channel information in the frequency domain. Prior works have also used frequency domain TVLA analysis to quantify leakages in the frequency domain [46, 82]. Figure 17 shows the average frequency trace and the first-order TVLA results with 2M traces. We observe that the t-scores are within the threshold for the frequency traces implying that the design does not leak information in the frequency domain as well.

### 6.3 Masking Overheads

Table 1 summarizes our implementation results for the baseline unmasked design and all the proposed variants. We observe that the `Variant-I` incurs an area overhead of 5.4× and 6.8× in the LUTs and FFs compared to the baseline. `Variant-II`, however, reduces these overheads to 4.7× and 6.6×, respectively, for the LUTs and FFs. The primary reason for the reduction is the use of KSA instead of RSA that reduces the latency of the adder and in turn also reduces the hardware cost for throughput optimization. `Variant-III`, which incorporates RSI-based shuffling along with masking, shows a negligible increase of 16 LUTs and 10 FFs compared to `Variant-II`.

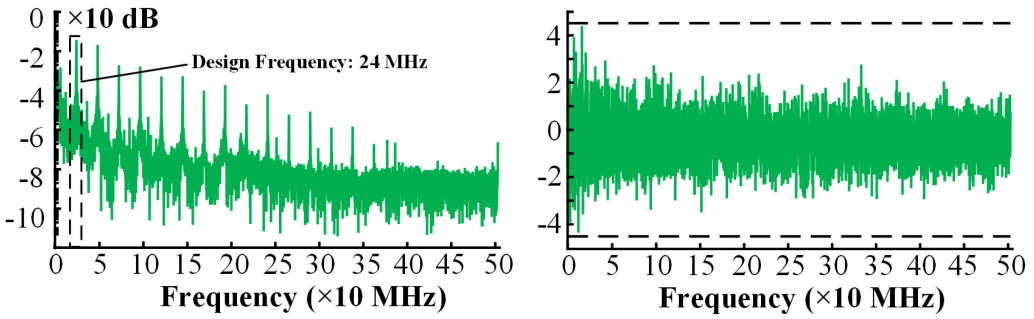


Fig. 17. Average frequency spectrum of masked `Variant-III` (left) and its TVLA result (right). The TVLA plot shows that there is no significant leakage in the frequency spectrum of the design when masked.

Table 1. Area (LUT/FF/BRAM) and Latency (in Cycles) Comparison of the Unmasked and Masked Implementations

| Variant | Area | Latency | Change |
|---|---|---|---|
| Unmasked | 1,833/1,125/163 | $2.85 \times 10^6$ | NA |
| `Variant-I` | 9,818/7,709/163 | $2.94 \times 10^6$ | 5.4×/6.8×/1× |
| `Variant-II` | 8,724/7,483/165 | $2.94 \times 10^6$ | 4.7×/6.6×/1× |
| `Variant-III` | 8,740/7,493/165 | $2.94 \times 10^6$ | 4.7×/6.6×/1× |

We also present a block-wise area distribution of the RCA and the KSA-based designs in Table 2 to better understand the reductions. The fourth column shows the increase or decrease in the area of a particular block in `Variant-III` compared to `Variant-I`. As expected, we notice an increase in the area of the KSA compared to the RSA due to more combinational logic like the generate and propagate circuitry. The number of PRNGs required is also higher in `Variant-III` due to an increase in the number of AND gates in the KSA compared to RSA that demand additional fresh random masks. However, we can observe the main contributor of the overall area in `Variant-I` is the Throughput Optimization circuitry, and we reduce the cost for this block by roughly 3× in `Variant-III`. Since the number of nodes in the hidden layers of `Variant-III` is 1,023 instead of 1,010 in `Variant-I`, we observe slight increase in the **read-only memories (ROM)** that store the parameters and the **read-write memories (RWM)** that store the node values. The latency in both the implementations increases only by a mere 1.03× compared to baseline unmasked design, primarily due to absence of the throughput optimization in the output layer.

Next, we compare the **area-delay product (ADP)** of the proposed Variants with the only work published on side-channel countermeasures for neural networks called MaskedNet [22]. Here, we define area as the sum of the number of LUTs and FFs, and delay as the latency in number of cycles. We emphasize that a direct comparison between these approaches in unfair, because MaskedNET inherently gains an advantage due to its parallelized design (compared to BoMaNET's sequentialized datapath). Indeed, the comparison of the baseline (unprotected) design of parallel versus sequential hardware reveals a 68× area-delay product advantage for the parallel hardware. The ADP increase in the masked versions, i.e., between `Variant-II` and protected MaskedNet design, is 80×, which is a relative increase of only 1.18×. Our improved design `Variant-III` reduces the relative increase in ADP to 1.08×.

Similar overheads were observed in previous works on bit-sliced masked AES implementations [97].

Table 2. Block-level Area Distribution of the Unmasked and Masked Implementations (LUT/FF/BRAM)

| Design Blocks | `Variant-I` | `Variant-III` | Fraction (%) |
|---|---|---|---|
| Adder | 929/1,135/0 | 1,572/2,074/0 | (+) 1.7/1.8/ |
| PRNGs | 1,188/1,314/0 | 3,112/2,916/0 | (+) 2.5/2.4/ |
| Output Layer | 54/82/0 | 50/82/0 | (−) 1.08/1/- |
| Throughput Optimization | 5,486/4,126/0 | 1,792/1,338/0 | (−) 3.1/3.1/- |
| Controller | 1,471/43/0 | 1,513/56/0 | (+) 1.02/1.3/- |
| ROMs | 1,009/670/159 | 1,027/677/161 | (+)1.01/1.01/1 |
| RWMs | 0/0/4 | 0/0/4 | -/-/1 |

"-" denotes no change in the area of the unmasked and masked design.
"(+)/(−)" in Fraction column denotes an increase/decrease in the area of `Variant-III` compared to `Variant-I`.

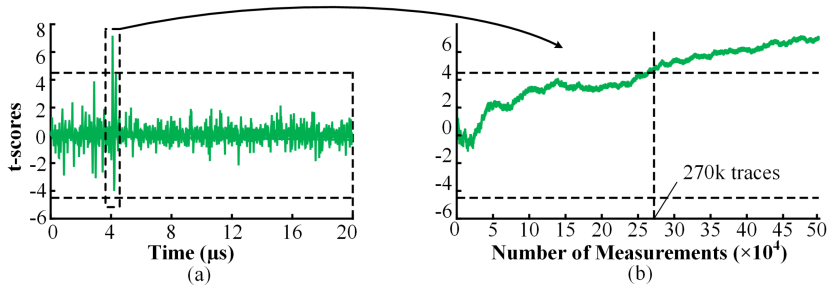


Fig. 18. TVLA results of LUT-based implementation of Trichina's AND gate.

## 7 DISCUSSIONS

### 7.1 Proof-of-Concept Versus Optimizations

Our solutions utilize simple yet effective techniques to mask an inference engine. But certainly, there is scope for improvement both in terms of the hardware design and the security countermeasures. In this section, we discuss possible optimizations/extensions of our work and alternate approaches taken in the field of privacy for ML.

*7.1.1 Design Optimizations.* We implement the Trichina's AND gate as a fully pipelined gate-level masked structure to resist leakages from the glitches. However, since it only utilizes 5 inputs and our target platform is an FPGA, it can also be implemented directly as a 5-input masked LUT. Such a design will significantly simplify the design, and reduce the latency of the masked AND operation and area costs due to pipeline registers. Reducing the latency of the adder will also decrease the area cost of the throughput optimization, which is what we observe when we switch from RCA to KSA as well. Therefore, we conducted TVLA tests on the LUT-based design as well to quantify the area costs and the security loss. Figure 18 shows the first-order t-test results on the masked full adder implemented using LUT-based Trichina's AND gates. We observe that the design starts leaking around 270k traces and area of the overall design decreases by 3× and 2.7× in the number of FFs and LUTs, respectively. The leakage is because of the glitches that happen inside the LUTs of the FPGA due to different arrival times of the inputs. These glitches can temporarily reconstruct the secret intermediates inside the LUT.

Prior literature discusses other techniques to resist glitches. For instance, the prior work on TI-based secure versions of RCA and KSA can be extended for this work [79]. Another potential optimization is to selectively allow only those glitches to happen that do not reconstruct the secret, and register every other combinational output. This idea has been well explored in the work by Gross et al. on Domain Oriented Masking [30]. Another prior work tried to prevent glitches inside the LUTs by disabling the toggles in the first half of the clock cycle using an enable signal as an input to the LUT [60]. However, it required implementing the AES SBox by manually hard instantiating each LUT making it somewhat tricky to be extended for larger designs like a neural network.

The ultimate goal of a side-channel defense is to make the design as secure as the primary-channel attack, which is theoretical cryptanalysis. For cryptography, this means making the design secure against an astronomical amount of tests (e.g., $2^{128}$ for AES). But this is not necessarily the case for machine learning. Recent works shows that a high-fidelity, cryptanalytic model extraction succeeds with a few million tests on the types of networks that are similar to those we evaluate in this work [13]. Therefore, it would be sufficient for the side-channel defense to be secure against a test using a few million traces, because the design will be broken anyhow with that many tests through a cryptanalytic attack. The lowered security requirement of ML (compared to cryptographic primitives) would enable designers to opt for a solution that use a glitch-vulnerable Trichina's AND gate.

*7.1.2 Limitations.* We reduce glitch-related vulnerabilities using registers at each stage, which is a low-cost, practical solution. Other works have proposed stronger countermeasures at the cost of higher performance and area overheads [30, 62]. The quest for stronger and more efficient countermeasures is never-ending; masking of AES is still being explored, even 20 years after the initial work [2], due to the advent of more efficient or secure masking schemes [16] and more potent attacks [58, 90].

Our solution is first-order secure but there is scope for construction of higher-order masked versions. However, higher-order security is a delicate task; Moos et al. recently showed that a straightforward extension of masking schemes to higher-order suffers from both local and compositional flaws [59] and masking extensions were proposed in another recent work [14]. This is the first work on fully masked neural networks, and we foresee follow ups as we have experienced in the cryptographic research of AES masking, even after 20 years of intensive study.

We consider fault attacks out of scope for this work but it would be interesting to explore defenses for such attacks. Numerous countermeasures have been proposed to defend cryptographic implementations against fault attacks [18]. Therefore, one promising direction could be to explore how to extend those ideas in the context of machine learning algorithms. Another potential research direction is to explore efficient side-channel attacks to extract the model weights and biases, which might become tedious for deeper networks. For example, if the neural network layers are similar, then adversary may create a template of power consumption for the initial layer computations and use it to attack the subsequent layers.

### 7.2 Measurement Challenges

We faced some unique challenges that are not generally seen with the symmetric-key cryptographic evaluations. Inference becomes a lengthy operation, especially for an area-optimized design—the inference latency of our design is roughly 3 million cycles. For a design frequency of 24 MHz, the execution time translates to 122 ms per inference. If the oscilloscope samples at 125 MHz, then the number of sample points to be captured per power trace is equal to 15 million. This significantly slows down the capturing of power traces. In our case, capturing 2 million power

traces took one week, which means capturing 100 million traces as AES evaluation [16] will take roughly a year. Performing TVLA on such large traces (28 TB, in our case) also takes a significant amount of time: it took 3 days to get one t-score plot during our evaluations on a high-end PC.[10] One possibility to avoid this problem is looking at a small subset of representative traces of the computation [12]. We conduct a comprehensive evaluation of our design in one variant and an evaluation on a smaller subset in the other variants.

### 7.3 Theoretical Versus Side-channel Attacks

Theoretical model extraction by training a proxy model on a synthetically generated dataset using the predictions from the unknown victim model is an active area of research [13, 41]. These attacks mostly assume a black-box access to the model and successfully extract the model parameters after a certain number of queries. This number ranges typically in the order of $2^{20}$ [13]. By contrast, physical side-channel attacks only require a few thousand queries to successfully steal all the parameters [22]. This is partly due to fact that physical side-channel attacks can extract information about intermediate computations even in a black-box setting. Physical side-channel attacks also do not require the generation of the synthetic dataset, unlike most theoretical attacks.

### 7.4 Orthogonal ML Defences

There has been some work on defending the ML models against stealing of inputs and parameters using other techniques like **Homomorphic Encryption (HE)** and **Secure Multi-Party Computation (SMPC)** [27, 56, 73], Watermarking [1, 76], and **Trusted Execution Engines (TEE)** [32, 38, 88]. We refer the interested readers to survey papers published on ML privacy for a more exhaustive list [10, 39, 63, 81]. Purely HE-based or HE+SMPC solutions initially incurred a large overhead (30,000× in CryptoNet [27] and 500× in SecureML [56]), but follow-up work made them more efficient (13× in XONN[73]). We propose masking, which is an extension of SMPC on hardware, and we believe that it is a promising direction for ML side-channel defenses as it has been on cryptographic applications. Watermarking techniques are punitive methods that cannot prevent physical side-channel attacks. TEEs are subject to ever-evolving microarchitectural attacks and typically are not available in edge/IoT nodes.

### 7.5 Extensions to More Complicated Networks

We believe that some of the solutions we propose can be used to mask other types of neural networks as well, and are not just confined to the specific MLP architecture we target in this work. The masked adder can be used to mask the convolution operation in Convolutional Neural Networks. Our masked output layer design can be used to construct a masked MaxPool hardware that can compute the maximum element in a given pooling window in masked fashion. MaxPool is typically implemented using OR gates if the network is binarized, because MaxPool becomes equivalent to an OR operation. Our proposed synchronized Trichina's AND gate design can be used to mask the OR operations too.

## 8 CONCLUSIONS AND FUTURE OUTLOOK

Physical side-channel analysis of neural networks is a new, promising direction in hardware security where the attacks are rapidly evolving compared to defenses. We proposed the first fully masked neural network, demonstrated the security with up to 2M traces, and quantified the overheads of a potential countermeasure. We addressed the key challenge of masking integer addition [22] through Boolean masking. We furthermore presented ideas on how to mask the unique

[10]Intel Core i9-9900K, 64 GB RAM.

linear and non-linear computations of a fully connected neural network that do not exist in cryptographic applications. We also demonstrated how to couple a first-order masked design with a lightweight shuffle countermeasure to provide additional second-order side-channel security. The combination of the two defenses can raise the side-channel resilience to a level that succeeding a theoretical/cryptanalytic attack may need fewer queries.

The large variety in neural network architectures in terms of the quantization-level and the types of layer operations (e.g., Convolution, Maxpool, Softmax), and activation functions (e.g., ReLU, Sigmoid, Tanh) presents a large design space for neural network side-channel defenses. This work focused on BNNs, because we feel that BNNs are both excellent candidates for resource-constrained devices deployed at the edge and the closest flavors of ML algorithms to block ciphers (on which a large chunk of the side-channel literature focuses). Our ideas serve as a benchmark to analyze the vulnerabilities that exist in neural network computations and to construct more robust and efficient countermeasures.

## ACKNOWLEDGMENTS

We thank Dr. Sohrab Aftabjahani and Dr. Avinash Varna for their valuable feedback on the design.

## REFERENCES

[1] Yossi Adi, Carsten Baum, Moustapha Cissé, Benny Pinkas, and Joseph Keshet. 2018. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security'18)*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1615–1631. Retrieved from https://www.usenix.org/conference/usenixsecurity18/presentation/adi.

[2] Mehdi-Laurent Akkar and Christophe Giraud. 2001. An implementation of DES and AES, secure against some attacks. In *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES'01) (Lecture Notes in Computer Science)*, Çetin Kaya Koç, David Naccache, and Christof Paar (Eds.), Vol. 2162. Springer, 309–318. https://doi.org/10.1007/3-540-44709-1_26

[3] Monjur Alam, Santosh Ghosh, M. J. Mohan, Debdeep Mukhopadhyay, Dipanwita Roy Chowdhury, and Indranil Sengupta. 2009. Effect of glitches against masked AES S-box implementation and countermeasure. *IET Info. Secur.* 3, 1 (2009), 34–44. https://doi.org/10.1049/iet-ifs:20080041

[4] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security'19)*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 515–532. Retrieved from https://www.usenix.org/conference/usenixsecurity19/presentation/batina.

[5] Benjamin Jun Gilbert Goodwill, Josh Jaffe, and Pankaj Rohatgi. 2011. A testing methodology for side-channel resistance validation. http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf.

[6] Johannes Blömer, Jorge Guajardo, and Volker Krummel. 2004. Provably secure masking of AES. In *Proceedings of the 11th International Workshop on Selected Areas in Cryptography (SAC'04) (Lecture Notes in Computer Science)*, Helena Handschuh and M. Anwar Hasan (Eds.), Vol. 3357. Springer, 69–83. https://doi.org/10.1007/978-3-540-30564-4_5

[7] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. 2018. Practical fault attack on deep neural networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 2204–2206. https://doi.org/10.1145/3243734.3278519

[8] Jakub Breier, Dirmanto Jap, Xiaolu Hou, Shivam Bhasin, and Yang Liu. 2020. SNIFF: Reverse engineering of neural networks with fault attacks. Retrieved from https://arxiv.org/abs/2002.11021.

[9] Rosario Cammarota, Indranil Banerjee, and Ofer Rosenberg. 2018. Machine learning IP protection. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'18)*, Iris Bahar (Ed.). ACM, 19. https://doi.org/10.1145/3240765.3270589

[10] Rosario Cammarota, Matthias Schunter, Anand Rajan, Fabian Boemer, Ágnes Kiss, Amos Treiber, Christian Weinert, Thomas Schneider, Emmanuel Stapf, Ahmad-Reza Sadeghi, Daniel Demmler, Huili Chen, Siam Umar Hussain, M. Sadegh Riazi, Farinaz Koushanfar, Saransh Gupta, Tajana Simunic Rosing, Kamalika Chaudhuri, Hamid Nejatollahi, Nikil D. Dutt, Mohsen Imani, Kim Laine, Anuj Dubey, Aydin Aysu, Fateme Sadat Hosseini, Chengmo Yang, Eric Wallace, and Pamela Norton. 2020. Trustworthy AI inference systems: An industry research view. Retrieved from https://arxiv.org/abs/2008.04449.

[11] Christophe De Cannière and Bart Preneel. 2008. Trivium. In *New Stream Cipher Designs—The eSTREAM Finalists*, Matthew J. B. Robshaw and Olivier Billet (Eds.). Lecture Notes in Computer Science, Vol. 4986. Springer, 244–266. https://doi.org/10.1007/978-3-540-68351-3_18

[12] Mathieu Carbone, Vincent Conin, Marie-Angela Cornelie, François Dassance, Guillaume Dufresne, Cécile Dumas, Emmanuel Prouff, and Alexandre Venelli. 2019. Deep learning to evaluate secure RSA implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 2 (2019), 132–161. https://doi.org/10.13154/tches.v2019.i2.132-161

[13] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. 2020. Cryptanalytic extraction of neural network models. In *Proceedings of the 40th Annual International Cryptology Conference (CRYPTO'20) (Lecture Notes in Computer Science)*, Daniele Micciancio and Thomas Ristenpart (Eds.), Vol. 12172. Springer, 189–218. https://doi.org/10.1007/978-3-030-56877-1_7

[14] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. 2020. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Comput.* (2020), 1. https://doi.org/10.1109/TC.2020.3022979

[15] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. 2015. Differential power analysis of a McEliece cryptosystem. In *Proceedings of the 13th International Conference on Applied Cryptography and Network Security (ACNS'15) (Lecture Notes in Computer Science)*, Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis (Eds.), Vol. 9092. Springer, 538–556. https://doi.org/10.1007/978-3-319-28166-7_26

[16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2016. Masking AES with d+1 shares in hardware. In *Proceedings of the 18th International Conference on Cryptographic Hardware and Embedded Systems (CHES'16) (Lecture Notes in Computer Science)*, Benedikt Gierlichs and Axel Y. Poschmann (Eds.), Vol. 9813. Springer, 194–212. https://doi.org/10.1007/978-3-662-53140-2_10

[17] Matthieu Courbariaux and Y. Bengio. 2016. BinaryNet: Training deep neural networks with weights and activations constrained to +1 or −1. Retrieved from https://abs/1602.02830.

[18] Lauren De Meyer, Victor Arribas Abril, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2018. M&m: Masks and macs against physical attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 1 (2018), 25–50.

[19] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.

[20] Gaofeng Dong, Ping Wang, Ping Chen, Ruizhe Gu, and Honggang Hu. 2019. Floating-point multiplication timing attack on deep neural network. In *Proceedings of the IEEE International Conference on Smart Internet of Things (SmartIoT'19)*. IEEE, 155–161. https://doi.org/10.1109/SmartIoT.2019.00032

[21] Anuj Dubey, Rosario Cammarota, and Aydin Aysu. 2020. BoMaNet: Boolean masking of an entire neural network. In *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD'20)*. IEEE, 1–9. Retrieved from https://ieeexplore.ieee.org/document/9256785.

[22] Anuj Dubey, Rosario Cammarota, and Aydin Aysu. 2020. MaskedNet: The first hardware inference engine aiming power side-channel protection. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST'20)*. IEEE, 197–208. https://doi.org/10.1109/HOST45689.2020.9300276

[23] Vasisht Duddu, Debasis Samanta, D. Vijay Rao, and Valentina Emilia Balas. 2018. Stealing neural networks via timing side channels. Retrieved from http://arxiv.org/abs/1812.11720.

[24] Munira Ferdous, Jui Debnath, and Narayan Ranjan Chakraborty. 2020. Machine learning algorithms in healthcare: A literature survey. In *Proceedings of the 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT'20)*. IEEE, 1–6. https://doi.org/10.1109/ICCCNT49239.2020.9225642

[25] Cheng Fu, Shilin Zhu, Hao Su, Ching-En Lee, and Jishen Zhao. 2019. Towards fast and energy-efficient binarized neural network inference on FPGA. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'19)*. Association for Computing Machinery, New York, NY, 306. https://doi.org/10.1145/3289602.3293990

[26] Hongbo Gao, Bo Cheng, Jianqiang Wang, Keqiang Li, Jianhui Zhao, and Deyi Li. 2018. Object classification using CNN-based fusion of vision and LIDAR in autonomous vehicle environment. *IEEE Trans. Ind. Inform.* 14, 9 (2018), 4224–4231. https://doi.org/10.1109/TII.2018.2822828

[27] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on Machine Learning (ICML'16) (JMLR Workshop and Conference Proceedings)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. JMLR.org, 201–210. http://proceedings.mlr.press/v48/gilad-bachrach16.html.

[28] Jovan Dj. Golic and Christophe Tymen. 2002. Multiplicative masking and power analysis of AES. In *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'02) (Lecture Notes in Computer Science)*, Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar (Eds.), Vol. 2523. Springer, 198–212. https://doi.org/10.1007/3-540-36400-5_16

[29] Louis Goubin and Jacques Patarin. 1999. DES and differential power analysis the "duplication" method. In *Cryptographic Hardware and Embedded Systems*, Çetin K. Koç and Christof Paar (Eds.). Springer, Berlin, 158–172. https://doi.org/10.1007/3-540-48059-5_15

[30] Hannes Groß, Stefan Mangard, and Thomas Korak. 2016. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *Proceedings of the ACM Workshop on Theory of Implementation Security (TIS@CCS'16)*, Begül Bilgin, Svetla Nikova, and Vincent Rijmen (Eds.). ACM, 3. https://doi.org/10.1145/2996366.2996426

[31] Hendra Guntur, Jun Ishii, and Akashi Satoh. 2014. Side-channel AttacK user reference architecture board. In *Proceedings of the IEEE 3rd Global Conference on Consumer Electronics (GCCE'14)*. IEEE, 271–274. https://doi.org/10.1109/GCCE.2014.7031104

[32] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Max Augustin, Michael Backes, and Mario Fritz. 2018. MLCapsule: Guarded offline deployment of machine learning as a service. Retrieved from http://arxiv.org/abs/1808.00590.

[33] Xing Hu, Ling Liang, Lei Deng, Shuangchen Li, Xinfeng Xie, Yu Ji, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. 2019. Neural network model extraction attacks in edge devices by hearing architectural hints. Retrieved from http://arxiv.org/abs/1903.03916.

[34] Weizhe Hua, Zhiru Zhang, and G. Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference (DAC'18)*. ACM, 4:1–4:6. https://doi.org/10.1145/3195970.3196105

[35] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). 4107–4115. http://papers.nips.cc/paper/6573-binarized-neural-networks.

[36] Vincent Immler, Robert Specht, and Florian Unterstein. 2017. Your rails cannot hide from localized EM: How dual-rail logic fails on FPGAs. In *Proceedings of the 19th International Conference on Cryptographic Hardware and Embedded Systems (CHES'17) (Lecture Notes in Computer Science)*, Wieland Fischer and Naofumi Homma (Eds.), Vol. 10529. Springer, 403–424. https://doi.org/10.1007/978-3-319-66787-4_20

[37] Deloitte Insights. 2020. Bringing AI to the device: Edge AI chips come into their own. Retrieved from https://www2.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/2020/ai-chips.html.

[38] Mihailo Isakov, Lake Bu, Hai Cheng, and Michel A. Kinsy. 2018. Preventing neural network model exfiltration in machine learning hardware accelerators. In *Proceedings of the Asian Hardware Oriented Security and Trust Symposium (AsianHOST'18)*. IEEE, 62–67. https://doi.org/10.1109/AsianHOST.2018.8607161

[39] Mihailo Isakov, Vijay Gadepally, Karen M. Gettings, and Michel A. Kinsy. 2019. Survey of attacks and defenses on edge-deployed neural networks. In *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC'19)*. IEEE, 1–8. https://doi.org/10.1109/HPEC.2019.8916519

[40] Yuval Ishai, Amit Sahai, and David A. Wagner. 2003. Private circuits: Securing hardware against probing attacks. In *Proceedings of the 23rd Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'03) (Lecture Notes in Computer Science)*, Dan Boneh (Ed.), Vol. 2729. Springer, 463–481. https://doi.org/10.1007/978-3-540-45146-4_27

[41] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. 2020. High accuracy and high fidelity extraction of neural networks. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security'20)*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 1345–1362. Retrieved from https://www.usenix.org/conference/usenixsecurity20/presentation/jagielski.

[42] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N. Asokan. 2019. PRADA: Protecting against DNN model stealing attacks. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P'19)*. IEEE, 512–527. https://doi.org/10.1109/EuroSP.2019.00044

[43] Phil C. Knag, Gregory K. Chen, H. Ekin Sumbul, Raghavan Kumar, Mark A. Anders, Himanshu Kaul, Steven K. Hsu, Amit Agarwal, Monodeep Kar, Seongjong Kim, et al. 2020. A 617 TOPS/W all digital binary neural network accelerator in 10nm FinFET CMOS. In *Proceedings of the IEEE Symposium on VLSI Circuits*. IEEE, 1–2.

[44] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO'99) (Lecture Notes in Computer Science)*, Michael J. Wiener (Ed.), Vol. 1666. Springer, 388–397. https://doi.org/10.1007/3-540-48405-1_25

[45] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. 2011. Introduction to differential power analysis. *J. Cryptogr. Eng.* 1, 1 (2011), 5–27. https://doi.org/10.1007/s13389-011-0006-y

[46] Wan Lei, Lihui Wang, Weijun Shan, Kun Jiang, and Qing Li. 2017. A frequency-based leakage assessment methodology for side-channel evaluations. In *Proceedings of the 13th International Conference on Computational Intelligence and Security (CIS'17)*. IEEE, 590–593.

[47] Daniel Lemire. 2019. Fast random integer generation in an interval. *ACM Trans. Model. Comput. Simul.* 29, 1 (2019), 3:1–3:12. https://doi.org/10.1145/3230636

[48] Chuan Li. 2020. OpenAI's GPT-3 language model: A technical overview. Retrieved from https://lambdalabs.com/blog/demystifying-gpt-3/.

[49] Xiaofan Lin, Cong Zhao, and Wei Pan. 2017. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, 345–353. Retrieved from http://papers.nips.cc/paper/6638-towards-accurate-binary-convolutional-neural-network.pdf.

[50] Xiaoning Liu, Bang Wu, Xingliang Yuan, and Xun Yi. 2020. Leia: A Lightweight Cryptographic Neural Network Inference System at the Edge. Cryptology ePrint Archive, Report 2020/463. Retrieved from https://eprint.iacr.org/2020/463.

[51] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. 2008. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Vol. 31. Springer Science & Business Media.

[52] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. 2005. Side-channel leakage of masked CMOS gates. In *Proceedings of the Cryptographers' Track at the RSA Conference on Topics in Cryptology (CT-RSA'05) (Lecture Notes in Computer Science)*, Alfred Menezes (Ed.), Vol. 3376. Springer, 351–365. https://doi.org/10.1007/978-3-540-30574-3_24

[53] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. 2005. Successfully attacking masked AES hardware implementations. In *Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'05) (Lecture Notes in Computer Science)*, Josyula R. Rao and Berk Sunar (Eds.), Vol. 3659. Springer, 157–171. https://doi.org/10.1007/11545262_12

[54] Javier Martinez, Carla Iglesias Comesaña, and Paulino José García Nieto. 2019. Review: Machine learning techniques applied to cybersecurity. *Int. J. Mach. Learn. Cybern.* 10, 10 (2019), 2823–2836. https://doi.org/10.1007/s13042-018-00906-1

[55] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. 2010. Fresh Re-keying: Security against side-channel and fault attacks for low-cost devices. In *Proceedings of the 3rd International Conference on Cryptology in Africa on Progress in Cryptology (AFRICACRYPT'10) (Lecture Notes in Computer Science)*, Daniel J. Bernstein and Tanja Lange (Eds.), Vol. 6055. Springer, 279–296. https://doi.org/10.1007/978-3-642-12678-9_17

[56] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'17)*. IEEE Computer Society, 19–38. https://doi.org/10.1109/SP.2017.12

[57] Shayan Moini, Shanquan Tian, Jakub Szefer, Daniel E. Holcomb, and Russell Tessier. 2020. Remote power side-channel attacks on CNN accelerators in FPGAs. Retrieved from https://arxiv.org/abs/2011.07603.

[58] Thorben Moos, Amir Moradi, and Bastian Richter. 2017. Static power side-channel analysis of a threshold implementation prototype chip. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*, David Atienza and Giorgio Di Natale (Eds.). IEEE, 1324–1329. https://doi.org/10.23919/DATE.2017.7927198

[59] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. 2019. Glitch-resistant masking revisited or why proofs in the robust probing model are needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 2 (2019), 256–292. https://doi.org/10.13154/tches.v2019.i2.256-292

[60] Amir Moradi and Oliver Mischke. 2012. Glitch-free implementation of masking in modern FPGAs. In *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST'12)*. IEEE Computer Society, 89–95. https://doi.org/10.1109/HST.2012.6224326

[61] Maxime Nassar, Shivam Bhasin, Jean-Luc Danger, Guillaume Duc, and Sylvain Guilley. 2010. BCDL: A high speed balanced DPL for FPGA with global precharge and no early evaluation. In *Proceedings of the Design, Automation and Test in Europe (DATE'10)*, Giovanni De Micheli, Bashir M. Al-Hashimi, Wolfgang Müller, and Enrico Macii (Eds.). IEEE Computer Society, 849–854. https://doi.org/10.1109/DATE.2010.5456932

[62] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. 2006. Threshold implementations against side-channel attacks and glitches. In *Proceedings of the 8th International Conference on Information and Communications Security (ICICS'06) (Lecture Notes in Computer Science)*, Peng Ning, Sihan Qing, and Ninghui Li (Eds.), Vol. 4307. Springer, 529–545. https://doi.org/10.1007/11935308_38

[63] NIST. 2019. A Taxonomy and Terminology of Adversarial Machine Learning. Retrieved from https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8269-draft.pdf.

[64] Seong Joon Oh, Bernt Schiele, and Mario Fritz. 2019. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller (Eds.). Lecture Notes in Computer Science, Vol. 11700. Springer, 121–144. https://doi.org/10.1007/978-3-030-28954-6_7

[65] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. 2005. A side-channel analysis resistant description of the AES S-box. In *Proceedings of the 12th International Workshop on Fast Software Encryption (FSE'05)*

*(Lecture Notes in Computer Science)*, Henri Gilbert and Helena Handschuh (Eds.), Vol. 3557. Springer, 413–423. https://doi.org/10.1007/11502760_28

[66] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* 22, 10 (2010), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

[67] Aesun Park, Kyung-Ah Shim, Namhun Koo, and Dong-Guk Han. 2018. Side-channel attacks on post-quantum signature schemes based on multivariate quadratic equations—Rainbow and UOV. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 500–523. https://doi.org/10.13154/tches.v2018.i3.500-523

[68] Thomas Popp and Stefan Mangard. 2005. Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In *Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'05) (Lecture Notes in Computer Science)*, Josyula R. Rao and Berk Sunar (Eds.), Vol. 3659. Springer, 172–186. https://doi.org/10.1007/11545262_13

[69] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proceedings of the14th European Conference on Computer Vision (ECCV'16) (Lecture Notes in Computer Science)*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.), Vol. 9908. Springer, 525–542. https://doi.org/10.1007/978-3-319-46493-0_32

[70] Francesco Regazzoni, Shivam Bhasin, Amir Ali Pour, Ihab Alshaer, Furkan Aydin, Aydin Aysu, Vincent Beroulle, Giorgio Di Natale, Paul Franzon, David Hely, et al. 2020. Machine learning and hardware security: Challenges and opportunities-invited talk. In *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD'20)*. IEEE, 1–6.

[71] Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. 2019. Efficiently stealing your machine learning models. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society (WPES@CCS'19)*, Lorenzo Cavallaro, Johannes Kinder, and Josep Domingo-Ferrer (Eds.). ACM, 198–210. https://doi.org/10.1145/3338498.3358646

[72] Oscar Reparaz, Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. 2016. Additively homomorphic ring-LWE masking. In *Proceedings of the 7th International Workshop on Post-Quantum Cryptography (PQCrypto'16) (Lecture Notes in Computer Science)*, Tsuyoshi Takagi (Ed.), Vol. 9606. Springer, 233–244. https://doi.org/10.1007/978-3-319-29360-8_15

[73] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-based oblivious deep neural network inference. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security'19)*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 1501–1518. https://www.usenix.org/conference/usenixsecurity19/presentation/riazi.

[74] Mauro Ribeiro, Katarina Grolinger, and Miriam A. M. Capretz. 2015. MLaaS: Machine learning as a service. In *Proceedings of the 14th IEEE International Conference on Machine Learning and Applications (ICMLA'15)*, Tao Li, Lukasz A. Kurgan, Vasile Palade, Randy Goebel, Andreas Holzinger, Karin Verspoor, and M. Arif Wani (Eds.). IEEE, 896–902. https://doi.org/10.1109/ICMLA.2015.152

[75] Riscure. 2019. Riscure Inspector. Retrieved from https://www.riscure.com/uploads/2017/08/inspector_brochure.pdf.

[76] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. 2019. DeepSigns: An end-to-end watermarking framework for ownership protection of deep neural networks. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*, Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck (Eds.). ACM, 485–497. https://doi.org/10.1145/3297858.3304051

[77] Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi Baradaran Tahoori. 2018. An inside job: Remote power analysis attacks on FPGAs. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*, Jan Madsen and Ayse K. Coskun (Eds.). IEEE, 1111–1116. https://doi.org/10.23919/DATE.2018.8342177

[78] Tobias Schneider and Amir Moradi. 2015. Leakage assessment methodology—A clear roadmap for side-channel evaluations. In *Proceedings of the 17th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'15) (Lecture Notes in Computer Science)*, Tim Güneysu and Helena Handschuh (Eds.), Vol. 9293. Springer, 495–513. https://doi.org/10.1007/978-3-662-48324-4_25

[79] Tobias Schneider, Amir Moradi, and Tim Güneysu. 2015. Arithmetic addition over boolean masking—Towards first- and second-order resistance in hardware. In *Proceedings of the 13th International Conference on Applied Cryptography and Network Security (ACNS'15) (Lecture Notes in Computer Science)*, Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis (Eds.), Vol. 9092. Springer, 559–578. https://doi.org/10.1007/978-3-319-28166-7_27

[80] Security.org. 2020. The Best Indoor Cameras for Artificial Intelligence. Retrieved from https://www.security.org/security-cameras/best/artificial-intelligence.

[81] Muhammad Shafique, Mahum Naseer, Theocharis Theocharides, Christos Kyrkou, Onur Mutlu, Lois Orosa, and Jungwook Choi. 2020. Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead. *IEEE Des. Test* 37, 2 (2020), 30–57. https://doi.org/10.1109/MDAT.2020.2971217

[82] Arvind Singh, Monodeep Kar, Sanu K. Mathew, Anand Rajan, Vivek De, and Saibal Mukhopadhyay. 2018. Improved power/EM side-channel attack resistance of 128-bit AES engines with random fast voltage dithering. *IEEE J. Solid-State Circ.* 54, 2 (2018), 569–583.

[83] Go Takatoi, Takeshi Sugawara, Kazuo Sakiyama, and Yang Li. 2020. Simple electromagnetic analysis against activation functions of deep neural networks. In *Proceedings of the Satellite Workshops on Applied Cryptography and Network Security Workshops (ACNS'20) (Lecture Notes in Computer Science)*, Jianying Zhou, Mauro Conti, Chuadhry Mujeeb Ahmed, Man Ho Au, Lejla Batina, Zhou Li, Jingqiang Lin, Eleonora Losiouk, Bo Luo, Suryadipta Majumdar, Weizhi Meng, Martín Ochoa, Stjepan Picek, Georgios Portokalidis, Cong Wang, and Kehuan Zhang (Eds.), Vol. 12418. Springer, 181–197. https://doi.org/10.1007/978-3-030-61638-0_11

[84] Techcrunch. 2020. Apple buys edge-based AI startup Xnor.ai for a reported $200M. Retrieved from https://techcrunch.com/2020/01/15/apple-buys-edge-based-ai-startup-xnor-ai-for-a-reported-200m/.

[85] Pico Technology. 2020. Retrieved from https://www.picotech.com/oscilloscope/3000/picoscope-3000-oscilloscope-specifications.

[86] Kris Tiri and Patrick Schaumont. 2006. Changing the odds against masked logic. In *Proceedings of the 13th International Workshop on Selected Areas in Cryptography (SAC'06) (Lecture Notes in Computer Science)*, Eli Biham and Amr M. Youssef (Eds.), Vol. 4356. Springer, 134–146. https://doi.org/10.1007/978-3-540-74462-7_10

[87] Kris Tiri and Ingrid Verbauwhede. 2004. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *Proceedings of the Design, Automation and Test in Europe Conference and Exposition (DATE'04)*. IEEE Computer Society, 246–251. https://doi.org/10.1109/DATE.2004.1268856

[88] Florian Tramèr and Dan Boneh. 2019. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *Proceedings of the 7th International Conference on Learning Representations (ICLR'19)*. OpenReview.net. https://openreview.net/forum?id=rJVorjCcKQ.

[89] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction APIs. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security'16)*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 601–618. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer.

[90] Elena Trichina, Tymur Korkishko, and Kyung-Hee Lee. 2004. Small size, low power, side channel-immune AES coprocessor: Design and synthesis results. In *Proceedings of the 4th International Conference on Advanced Encryption Standards (AES'04) (Lecture Notes in Computer Science)*, Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa (Eds.), Vol. 3373. Springer, 113–127. https://doi.org/10.1007/11506447_10

[91] Yaman Umuroglu et al. 2017. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*.

[92] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. 2012. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *Proceedings of the 18th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'12) (Lecture Notes in Computer Science)*, Xiaoyun Wang and Kazue Sako (Eds.), Vol. 7658. Springer, 740–757. https://doi.org/10.1007/978-3-642-34961-4_44

[93] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. 2018. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC'18)*. ACM, 393–406. https://doi.org/10.1145/3274694.3274696

[94] Xiaodan Xi, Aydin Aysu, and Michael Orshansky. 2018. Fresh re-keying with strong PUFs: A new approach to side-channel security. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST'18)*. IEEE Computer Society, 118–125. https://doi.org/10.1109/HST.2018.8383899

[95] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. 2020. Open DNN box by power side-channel attack. *IEEE Trans. Circ. Syst.* 67-II, 11 (2020), 2717–2721. https://doi.org/10.1109/TCSII.2020.2973007

[96] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. 2020. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *Proceedings of the 29th USENIX Security Symposium (USENIX Security'20)*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2003–2020. Retrieved from https://www.usenix.org/conference/usenixsecurity20/presentation/yan.

[97] Yuan Yao, Mo Yang, Conor Patrick, Bilgiday Yuce, and Patrick Schaumont. 2018. Fault-assisted side-channel analysis of masked implementations. In *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST'18)*. IEEE Computer Society, 57–64. https://doi.org/10.1109/HST.2018.8383891

[98] Pengyuan Yu and Patrick Schaumont. 2007. Secure FPGA circuits using controlled placement and routing. In *Proceedings of the 5th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'07)*, Soonhoi Ha, Kiyoung Choi, Nikil D. Dutt, and Jürgen Teich (Eds.). ACM, 45–50. https://doi.org/10.1145/1289816.1289831

[99] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. 2020. DeepEM: Deep neural networks model recovery through EM side-channel information leakage. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020*. IEEE, 209–218. https://doi.org/10.1109/HOST45689.2020.9300274

[100] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Netw. Learn. Syst.* 30, 9 (2019), 2805–2824. https://doi.org/10.1109/TNNLS.2018.2886017

[101] Mark Zhao and G. Edward Suh. 2018. FPGA-based remote power side-channel attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (SP 2018)*. IEEE Computer Society, 229–244. https://doi.org/10.1109/SP.2018.00049

[102] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. Retrieved from https://arXiv:1606.06160.

[103] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* 107, 8 (2019), 1738–1762. https://doi.org/10.1109/JPROC.2019.2918951

Received December 2020; revised May 2021; accepted May 2021