MaskedNet: The First Hardware Inference Engine Aiming Power Side-Channel Protection

Anuj Dubey*, Rosario Cammarota†, Aydin Aysu*
*Department of Electrical and Computer Engineering, North Carolina State University
{aanujdu, aaysu}@ncsu.edu
†Intel AI, Privacy and Security Research
rosario.cammarota@intel.com

***Abstract*—Differential Power Analysis (DPA) has been an active area of research for the past two decades to study the attacks for extracting secret information from cryptographic implementations through power measurements and their defenses. The research on power side-channels have so far predominantly focused on analyzing implementations of ciphers such as AES, DES, RSA, and recently post-quantum cryptography primitives (e.g., lattices). Meanwhile, machine-learning applications are becoming ubiquitous with several scenarios where the Machine Learning Models are Intellectual Properties requiring confidentiality. Expanding side-channel analysis to Machine Learning Model extraction, however, is largely unexplored.**

This paper expands the DPA framework to neural-network classifiers. First, it shows DPA attacks during inference to extract the secret model parameters such as weights and biases of a neural network. Second, it proposes the *first countermeasures* against these attacks by augmenting *masking*. The resulting design uses novel masked components such as masked adder trees for fully-connected layers and masked Rectifier Linear Units for activation functions. On a SAKURA-X FPGA board, experiments show that the first-order DPA attacks on the unprotected implementation can succeed with only 200 traces and our protection respectively increases the latency and area-cost by 2.8× and 2.3×.

***Index Terms*—Machine Learning, Neural Networks, Side-Channel Analysis, Masking.**

I. Introduction

Since the seminal work on Differential Power Analysis (DPA) [1], there has been an extensive amount of research on power side-channel analysis of cryptographic systems. Such research effort typically focus on new ways to break into various implementations of cryptographic algorithms and countermeasures to mitigate attacks. While cryptography is obviously an important target driving this research, it is not the only scenario where asset confidentiality is needed—secret keys in the case of cryptographic implementations.

In fact, Machine Learning (ML) is a critical new target with several motivating scenarios to keep the internal ML model secret. The ML models are considered trademark secrets, e.g., in Machine-Learning-as-a-Service applications, due to the difficulty of training ML models and privacy concerns about the information embedded in the model coefficients such as weights and biases in the case of neural networks. If leaked, the model, including weights, biases and hyper-parameters can violate data privacy and intellectual property rights. Moreover, knowing the ML classifier details makes it more susceptible to adversarial ML attacks [2], and especially to test-time evasion attacks[3], [4]. Finally, ML has also been touted to replace cryptographic primitives [5]—under this scenario, learning the ML classifier details would be equivalent to extracting secrets from cryptographic implementations.

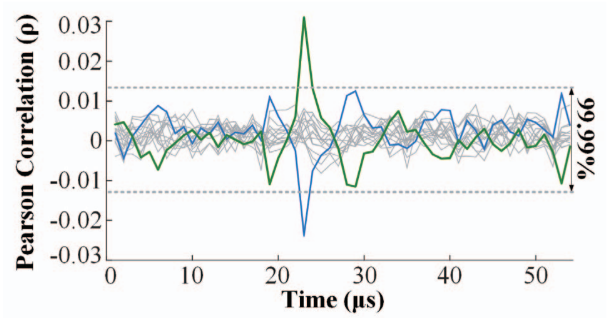

Fig. 1. Motivation of this work: DPA of the BNN hardware with 100k measurements. Green plot is the correlation trace for the correct 4-bit weight guess, which crosses the 99.99% confidence threshold revealing the significant leak. Blue plot is for the 2's complement of the correct guess, which is an expected false positive of the target signed multiplication. Other 14 guesses do not show correlation.

In this work, we extend the side-channel analysis framework to ML models. Specifically, we apply power-based side-channel attacks on a hardware implementation of a neural network and propose the *first* side-channel countermeasure. Fig. 1 shows the vulnerability of a Binarized Neural Network (BNN)—an efficient network for IoT/edge devices with binary weights and activation values [6]. Following the DPA methodology [7], the adversary makes hypothesis on 4 bits of the secret weight. For all these 16 possible weight values, the adversary computes the corresponding power activity on an intermediate computation, which depends on the known input and the secret weight. This process is repeated multiple times using random, known inputs. The correlation plots between the calculated power activities for the 16 guesses and the obtained power measurements reveal the value of the secret weight.

Fig. 1 shows that at the exact time instant where the targeted computation occurs, a significant information leakage exists between the power measurements and the correct key guess. The process can be repeated reusing the same power measurements to extract other weights. Hence, it shows that implementations of ML Intellectual Properties are also susceptible to side-channel attacks like ciphers.

Given this vulnerability, the objective of this paper is to examine it and propose the first countermeasure attempt for neural network inference against power-based side-channel attacks. A neural network inference is a sequence of repeated linear and non-linear operations, similar in essence to cryptographic algorithms, but has unique computations such as row-reduction (i.e., weighted summation) operations and activation functions. Unlike the attack scenario, the defense exhibits challenges due to the presence of these operations in neural networks, which introduce an additional and subtle type of leak. To address the vulnerability, we propose a countermeasure using the concepts of message blinding and secret sharing. This countermeasure style is called masking [8], which is an *algorithm-level* defense that can produce resilient designs independent of the implementation technology [9]. We tuned this countermeasure for the neural networks in a cost effective way and complement it with other techniques.

The main contributions of the paper include the following:

- We demonstrate attacks that can extract the secret weights of a BNN in a highly-parallelized hardware implementation.
- We formulate and implement the *first* power-based side-channel countermeasures for neural networks by adapting masking to the case of neural networks. This process reveals new challenges and solutions to mask unique neural network computations that do not occur in the cryptographic domain.
- We validate both the insecurity of the baseline design and the security of the masked design using power measurement of an actual FPGA hardware and quantify the overheads of the proposed countermeasure.

We note that while there is prior work on theoretical attacks [10], [11], [12], [13], [14] and digital side-channels [15], [16], [17], [18] of neural networks, their physical side-channels are largely unexplored. Such research is needed because physical side-channels are orthogonal to these threats, fundamental to the Complementary Metal Oxide Semiconductor (CMOS) technology, and require extensive countermeasures as we have learned from the research on cryptographic implementations. A white paper is recently published on model extraction via physical side-channels [19][1]. This work does not study mitigation techniques and focuses on 8-bit/32-bit microcontrollers. We further analyze attacks on parallelized hardware accelerators and investigate the *first* countermeasures.

II. Threat Model and Relation to Prior Work

This work follows the typical DPA threat model [21]. The adversary has physical access to the target device or has a remote monitor [22], [23], [24] and obtains power measurements while the device processes secret information. We assume that the security of the system is not based on the secrecy of the software or hardware design. This includes the details of neural network algorithm and its hardware implementation such as the data flow, parallelization and pipelining—in practice, those details are typically public but **what remains a secret is the model parameters obtained via training.** For unknown implementations, adversary can use prior techniques to locate certain operations, which work in the context of physical [25], [26] and digital side-channels [27], [28] covering both hardware and software realizations. This aspect of reverse engineering logic functions from bitstream [29], [30] is independent of our case and is generic to any given system.

[1]After we submitted our work to HOST'20, that paper has been published at USENIX'19 [20].

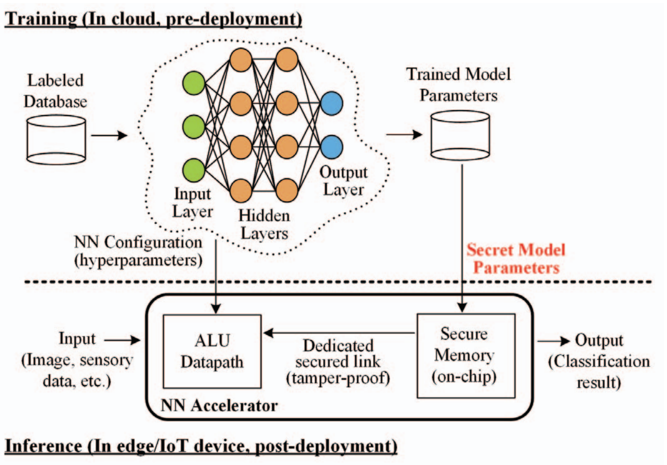
Training (In cloud, pre-deployment)
Labeled Database
Input Layer
Hidden Layers
Output Layer
Trained Model Parameters
NN Configuration (hyperparameters)
Secret Model Parameters
Input (Image, sensory data, etc.)
ALU Datapath
Dedicated secured link (tamper-proof)
Secure Memory (on-chip)
NN Accelerator
Output (Classification result)
Inference (In edge/IoT device, post-deployment)

Fig. 2. The adversary's goal is to extract the secret model parameters on the IoT/edge device during inference using side-channels.

Fig. 2 outlines the system we consider. The adversary in our model **targets the ML inference with the objective of learning the secret model parameters.** This is different than attacks on training set [31] or the data privacy problem during inference [32]. We assume the training phase is trusted but the obtained model is then deployed to operate in an untrusted environment. Our attack is similar to a known-plaintext (input) attack and does not require knowing the inference output or confidence score, making it more powerful than theoretical ML extraction attacks [10], [11], [12], [13], [14].

Since edge/IoT devices are the primary target of DPA attacks (due to easy physical access), we focus on BNNs that are suitable on such constrained devices [6]. A BNN also allows realizing the entire neural network on the FPGA without having external memory access. Therefore, memory access pattern side-channel attacks on the network [15] cannot be mounted. We furthermore consider digitally-hardened accelerator designs that execute in constant time and constant flow with no shared resources, disabling timing-based or other control-flow identification attacks [16], [17], [18]. This makes the attacks we consider more potent than prior work.

III. BNN and the Target Implementation

The following subsections give a brief introduction to BNN and discuss the details of the target hardware implementation.

A. Neural Network Classifiers

Neural networks consist of layers of neurons that take in an input vector and ultimately make a decision, e.g., for a classification problem. Neurons at each layer may have a different use, implementing linear or non-linear functions to make decisions, applying filters to process the data, or selecting specific patterns. Inspired from human nervous system, the

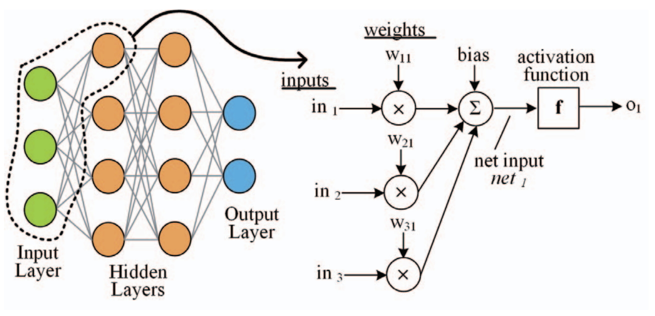

Fig. 3. A simple neural network and a single neuron's function.

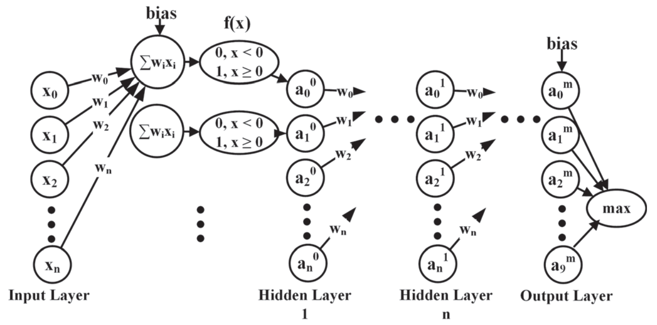

Fig. 4. Overview of the unprotected neural network inference. The adder tree first computes the weighted sum of input pixels. The activation function then binarizes the sum used by the next layer. Finally, the output layer returns the classification result by computing a maximum of last layer activations.

neurons in a neural network transmit information from one layer to the other typically in a feed-forward manner.

Fig. 3 shows a simple network with two fully-connected hidden layers and depicts the function of a neuron. In a feed-forward neural network, each neuron takes the results from its previous layer connections, computes a weighted summation (row-reduction), adds a certain bias, and finally applies a non-linear transformation to compute its *activation* output. The resulting activation value is used by the next layer's connected neurons in sequence. The connections between neurons can be strong, weak, or non-existent—the strength of these connections is called *weight*, which is a critical parameter for the network. The entire neural network model can be represented with these parameters, and with hyperparameters that are high-level parameters such as the number or type of the layers.

Neural networks have two phases: *training* and *inference*. During training, the network self-tunes its parameters for the specific classification problem at hand. This is achieved by feeding pre-classified inputs to the network together with their classification results, and by allowing the network to converge into acceptable parameters (based on some conditions) that can compute correct output values. During inference, the network uses those parameters to classify new (unlabeled) inputs.

B. BNNs

A BNN works with binary weights and activation values. This is our starting point as the implementations of such networks have similarities with the implementation of block ciphers. BNN reduces the memory size and converts a floating point multiplication to a single-bit XNOR operation in the inference [33]. Therefore, such networks are suitable for constrained IoT nodes where some of the detection accuracy can be traded for efficiency. Several variants of this low-cost approach exist to build neural networks with reasonably high accuracy [33], [34], [6].

The following is the mathematical equation (1) for a typical neuron:

$$a = f(\sum w_i x_i + b) \tag{1}$$

where a is the activation value, w_i is the weight, x_i is the activation of the previous layer and b is the bias value for the node. w_i, x_i, and a have binary values of 0 and 1 respectively representing the actual values of -1 and +1. The function $f(x)$ is the non-linear activation function (2), which in our case is defined as follows:

$$f(x) = \left\{ \begin{array}{ll} 0, & \text{for } x \leq 0 \\ 1, & \text{for } x > 1 \end{array} \right\} \tag{2}$$

Equations (1) and (2) show that the computation involves a summation of weighted products with binary weights with a bias offset and an eventual binarization.

We build the BNN inference hardware which performs Modified National Institute of Standards and Technology (MNIST) classification (hand written digit recognition from 28-by-28 pixel images) using 3 feed-forward, fully-connected hidden layers of 1024 neurons. The implementation computes up to 1024 additions (i.e., the entire activation value of a neuron) in parallel.

C. Unprotected Hardware Design

Fig. 4 illustrates the sequence of operations in the neural network. One fully-connected layer has two main steps: (1) calculating the weighted sums using an adder tree and (2) applying the non-linear activation function f. To classify the image, the hardware sends out the node number with maximum sum in the output layer.

The input image pixels are first buffered or negated based on whether the weight is 1 or 0 respectively, and then fed to the adder tree shown in Fig. 5. We implemented a fully-pipelined adder tree of depth 10 as the hardware needs up to 1024 parallel additions to compute the sum. The activation function binarizes the final sum. After storing all the first layer activations, the hardware computes the second layer activations using the first layer layer activations. The output layer consists of 10 nodes, each representing a classification class (0-9). The node index with the maximum confidence score becomes the output of the neural network.

The hardware reuses the same adder tree for each layer's computation, similar to a prior architecture [35]. Hence, the hardware has a throughput of approximately 3000 cycles per image classification. The reuse is not directly feasible as the adder tree (Fig. 5) can only support 784 inputs (of 8-bits) but it receives 1024 outputs from each of the hidden layers. Therefore, the hardware converts the 1024 1-bit outputs from each hidden layer to 512 2-bit outputs using LUTs. These LUTs take the weights and activation values as input, and produces the corresponding 512 2-bit sums, which is within

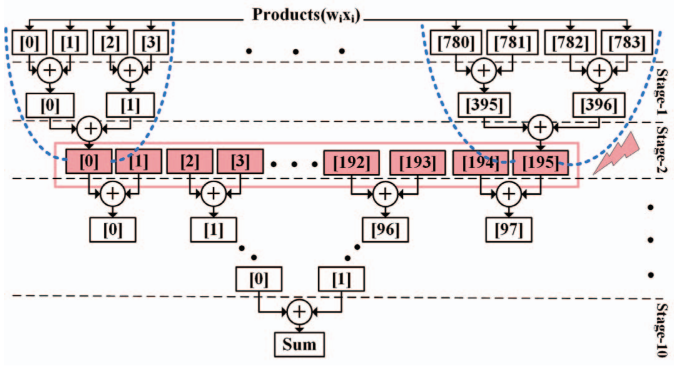

Fig. 5. Adder Tree used in HW Implementation. The figure shows the scenario where the 2nd stage registers(red) are targeted for DPA. This results in 16 possible key guesses corresponding to the 4 input pixels involved in the computation of each second stage register, grouped by the dotted blue line.

the limits of the adder tree. Adding bias and applying the batch normalization is integrated to the adder tree computations. We adopt the batch-normalization free approach [36], the bias values are thus integer unlike the binary weights.

IV. An Example of DPA on BNN Hardware

This section describes the attack we performed on the BNN hardware implementation that is able to extract secret weights.

To carry out a power based side-channel attack, the adversary has to primarily focus on the switching activity of the registers, as they have a significant power consumption compared to combinational logic (especially in FPGAs). The pipeline registers of the adder tree store the intermediate summations of the product of weights and input pixels. Therefore, the value in these registers is directly correlated to the secret—model weights in our case.

Fig. 5 shows an example attack. 4 possible values can be loaded in the output register $[0]$ of stage-1: $-[0]-[1]$, $-[0]+[1]$, $[0]-[1]$ and $[0]+[1]$ corresponding to the weights of (0,0), (0,1), (1,0) and (1,1), respectively. These values will directly affect the computation and the corresponding result stored in stage-2 registers. Therefore, a DPA attack with known inputs (x_i) on stage-2 registers (storing w_ix_i accumulations) can reveal 4-bits of the secret weights (w_i). The attack can target any stage of the adder tree but the number of possible weight combinations grows exponentially with depth.

Since the adder tree is pipelined, we need to create a model based on hamming distance of previous cycle and current cycle summations in each register. To aid the attack, we developed a cycle-accurate hamming-distance simulator for the adder pipeline. It first computes the value in each register every cycle for all possible weight combinations given a fixed input. Next, it calculates the hamming distance of individual registers for each cycle using the previous and current cycle values. Finally, it adds the hamming distances of all the registers for a cycle to model the overall power dissipation for that cycle.

Fig. 6 illustrates the result of the attack on stage-2 registers. There is a strong correlation between the correct key guess and the power measurements crossing the 99.99% confidence threshold after 45k measurements. The false positive leak is due to signed multiplication and is caused by the additive

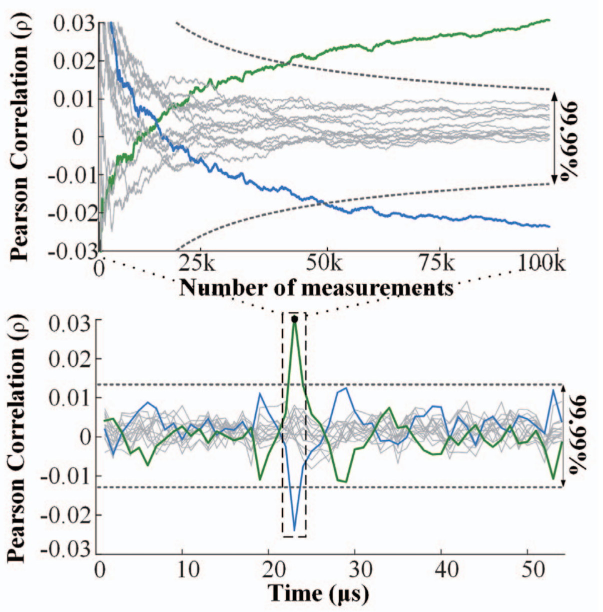

Fig. 6. Pearson Correlation Coefficient versus time and number of traces for DPA on weights. Lower plot shows a high correlation peak at the time of target computation, for the correct weight guess denoted in green. The upper plot shows that approximately 40k traces are needed to get a correlation of 99.99% for the correct guess. The confidence intervals are shown in dotted lines. The blue plot denotes the 2's complement of the correct weight guess

inverse of the correct key, which is expected and thus does not affect the attack. Using this approach, the attacker can successively extract the value of weights and biases for all the nodes in all the layers, starting from the first node and layer. The bias, in our design, is added after computing the final sum in the 10^{th} stage, before sending the result to the activation function. Therefore the adversary can attack this addition operation by creating a hypothesis for the bias. Another way extract bias is by attacking the activation function output since the sign of the output correlates to the bias.

V. Side-Channel Countermeasures

This section presents our novel countermeasure against side-channel attacks. The development of the countermeasure highlights unique challenges that arise for masking neural networks and describes the implementation of the entire neural network inference.

Masking works by making *all* intermediate computations independent of the secret key—i.e., rather than preventing the leak, by encumbering adversary's capability to correlate it with the secret value. The advantage of masking is being implementation agnostic. Thus, they can be applied on any given circuit style (FPGA or ASIC) without manipulating back-end tools but they require algorithmic tuning, especially to mask unique non-linear computations.

The main idea of masking is to split inputs of all key-dependent computations into two randomized shares: a one-time random mask and a one-time randomized masked value. These shares are then independently processed and are re-constituted at the final step when the final output is generated. This would effectively thwart first-order side-channel attacks probing a single intermediate computation. Higher-order attacks probing multiple computations [37]—masks and masked computations—can be further mitigated by splitting inputs of key-dependent operations into more shares [38]. Our implementation is designed to be first-order secure but can likewise be extended for higher order attacks.

Fig. 4 highlights that a typical neural network inference can be split into 3 distinct types of operations: (1) the adder tree computations, (2) the activation function and the (3) output layer max function. Our goal is to mask these functions to increase resilience against power side-channel attacks. These specific functions are unique to a neural network inference. Hence, we aim to construct novel masked architectures for them using the lessons learned from cryptographic side-channel research. We will explain our approach in a *bottom-up fashion* by describing masking of individual components first, and the entire hardware architecture next.

A. Masking the Adder Tree

Using the approach in Fig. 5, the adversary can attack any stage of the adder tree to extract the secret weights. Therefore, the countermeasure has to break the correlation between the summations generated at each stage and the secret weights. We use the technique of message blinding to mask the input of the adder tree.

Blinding is a technique where the inputs are randomized before sending to the target unit for computation. This prevents the adversary from knowing the actual inputs being processed, which is usually the basis for known-plaintext power-based side channel attacks. Fig. 7 shows our approach, that uses this concept by splitting each of the 784 input pixels a_i into two arithmetic shares r_i and a_i–r_i, where each r_i is a unique 8-bit random number. These two shares are therefore independent of the input pixel value, as r_i is a fresh random number never reused again for the same node. The adder tree can operate on each share individually due to additive homomorphism—it generates the two final summations for each branch such that their combination (i.e., addition) will give the original sum. Since the adder tree is reused for all layers, hardware simply repeats the masking process for subsequent hidden layers using fresh randomness in each layer.

1) A Unique and Fundamental Challenge for Arithmetic Masking of Neural Networks: The arithmetic masking extension to the adder tree is unfortunately non-trivial due to differences in the fundamental assumptions. Arithmetic masking aims at decorrelating a variable x by splitting it into two statistically independent shares: r and $(x - r)$ mod k. The modulo operation in $(x - r)$ mod k exists in most cryptographic implementations because most of them are based on finite fields. In a neural network, however, subtraction means computing the actual difference *without* any modulus. This introduces the notion of sign in numbers, which is absent in modulo arithmetic, and is the source of the problem.

Consider two uniformly distributed 8-bit unsigned numbers a and r. In a modulo subtraction, the result will be $(a - r)$ mod 256, which is again an 8-bit unsigned number

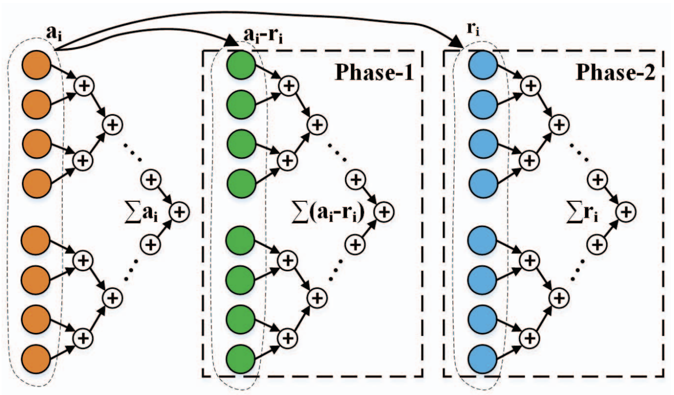

Fig. 7. Masking of adder tree. Each input pixel depicted in orange is split into two arithmetic shares depicted by the green and blue nodes with unique random numbers (r_is). The masked adder tree computes branches sequentially.

lying between 0 and 255. In an actual subtraction, however, the result will be $(a - r)$, which is a 9-bit number with MSB being the sign bit.

TABLE I
PROBABILITY OF $a - r$ BEING POSITIVE OR NEGATIVE

Scenario	Positive	Negative
$a > 128$ & $r > 128$	50%	50%
$a > 128$ & $r < 128$	100%	0%
$a < 128$ & $r > 128$	0%	100%
$a < 128$ & $r < 128$	50%	50%

Table I lists four possible scenarios of arithmetic masking based on the magnitude of the two unsigned 8-bit shares. In a perfect masking scheme, probability of $a - r$ being either positive or negative should be 50%, irrespective of the magnitude of the input a. Let's consider the case when $a > 128$, which has a probability of 50%. If $r < 128$, which also has a 50% probability, the resulting sum $a - r$ is always positive. Else if $r > 128$, the value $a-r$ can both be positive or negative with equal probabilities due to uniform distribution. Therefore, given $a > 128$, the probability of the arithmetic mask being positive is $(50{+}25)\% = 75\%$ and being negative is 25%. Table I lists the other case when $a < 128$, which results a similar correlation between a and $a - r$. This is showing a clear information leak through the sign bit of arithmetic masks.

The discussed vulnerability does not happen in modulo arithmetic as *there is no sign bit; the modulo operation wraps around the result if it is out of bounds, to obey the closure property*. Evaluating the correlation of $(a{+}r)$ instead of $(a{-}r)$ yields similar results. Likewise, shifting the range of r based on a, to uniformly distribute $(a - r)$ between -128 to 127, would not resolve the problem and further introduces a bias in both shares.

2) Addressing the Vulnerability with Hiding: The arithmetic masking scheme can be augmented to decorrelate the sign bit from the input. We used hiding to address this problem. We used hiding *just for the sign bit computation*. Hiding techniques target constant power consumption, irrespective of the inputs, which makes it harder for an attacker to correlate the intermediate variables. Power equalized building blocks using techniques like Wave Differential Dynamic

Logic (WDDL) [39] can achieve close to a constant power consumption to mitigate the vulnerability.

The differential part of WDDL circuits aims to make the power consumption constant throughout the operation, by generating the complementary outputs of each gate along with the original outputs. Differential logic makes it difficult for an attacker to distinguish between a $0 \to 1$ and a $1 \to 0$ transition, however, an attacker can still distinguish between a $0 \to 0$ and a $0 \to 1$ transition or a $1 \to 1$ and a $1 \to 0$ transition. Therefore, the differential logic alone is still susceptible to side channel leakages, as the power activity is easily correlated to the input switching pattern. This vulnerability is reduced by dynamic logic, where all the gates are pre-charged to 0, before the actual computation.

We use the WDDL gates to solve our problem of sign bit leakages, by modifying the adders to compute the sign bit in WDDL style. Following is the equation of the addition, when two 8-bit signed numbers a and b, represented as $(a_7a_6 \cdots a_0)$ and $(b_7b_6 \cdots b_0)$ are added to give a 9-bit signed sum s represented by $(s_8s_7 \cdots s_0)$:

$$s = a + b \tag{3}$$

After sign-extending a and b,

$$\{s_8s_7s_6 \cdots s_0\} = \{a_7a_7a_6 \cdots a_0\} + \{b_7b_7b_6 \cdots b_0\} \tag{4}$$

Performing regular addition on the leftmost 8 bits of a and b, and generating a carry c, the equation of s_8 becomes

$$s_8 = a_7 \oplus b_7 \oplus c \tag{5}$$

Expanding the above expression in terms of AND, OR and NOT operators results:

$$s_8 = (\overline{a_7} \cdot b_7 \cdot \overline{c}) | (a_7 \cdot \overline{b_7} \cdot \overline{c}) | (a_7 \cdot b_7 \cdot c) | (\overline{a_7} \cdot \overline{b_7} \cdot c) \tag{6}$$

Representing the expression only in terms of NAND, so that we can replace all the NANDs by WDDL NAND gates reveals:

$$s_8 = \overline{\overline{(\overline{a_7} \cdot b_7 \cdot \overline{c})} \cdot \overline{(a_7 \cdot \overline{b_7} \cdot \overline{c}} \cdot \overline{(a_7 \cdot b_7 \cdot c)} \cdot \overline{(\overline{a_7} \cdot \overline{b_7} \cdot c)}} \tag{7}$$

Fig. 8 depicts the circuit diagram for the above implementation. The WDDL technique is applied to the MSB computation by replacing each NAND function in Eq (7) with WDDL NAND gates. The pipeline registers of the adder tree are replaced by Simple Dynamic Differential Logic (SDDL) registers [39]. Each WDDL adder outputs the actual sum s and the complement of its MSB s_8', which go as input to the WDDL adder in the next stage of the pipelined tree. Therefore, we construct a resilient adder tree mitigating the leakage in the sign-bit.

B. Masking the Activation Function

The binary sign function (Eq. 2) is the activation function of BNN. This function generates +1 if the weighted sum is positive, else -1 if the sum is negative. In the unmasked implementation, the sign function receives the weighted sum of the 784 original input pixels, whereas in the masked implementation, it receives the two weighted sums corresponding to each masked share. So, the masked function has to compute the sign of the sum of two shares *without* actually adding them. Using the fact that the sign only depends on the MSB

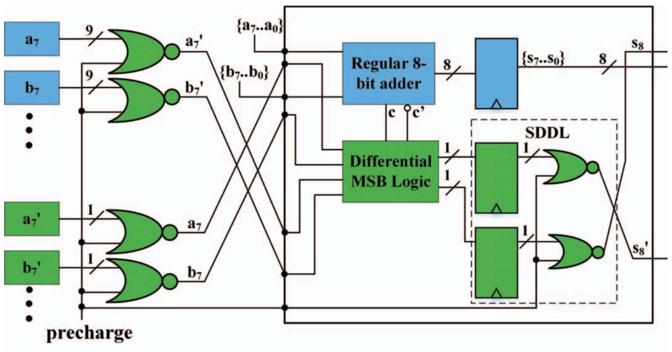

Fig. 8. Circuit diagram of the proposed adder with MSB computed in WDDL style as described in Eq.(3)-(7). Each of the 784 arithmetic shares $(a, b, \ldots)$ are fed to these adders. All the bits except the MSB undergo a regular addition. The MSBs of the two operands along with the generated carry are fed to the Differential MSB Logic block, which computes the MSB and its complement by replacing the NAND gates in Eq (7) by WDDL gates. The pipeline registers in the tree are replaced by SDDL registers. The NOR gates generate the pre-charge wave at the start of the logic cones. The regular and the WDDL specific blocks are depicted in blue and green respectively

of the final sum, we propose a novel masked sign function that sequentially computes and propagates the masked carry bits in a ripple carry fashion.

Fig. 9 shows the details of our proposed masked sign function hardware. This circuit generates the first *masked carry* using a Look-up-Table (LUT) that takes in the LSB of both shares and a fresh random bit (r_i) to ensure the randomization of the intermediate state, similar in style to prior works on masked LUT designs [40]. LUT function computes the masked function with the random input and generates two outputs: one is the bypass of the random value (r_i) and the other is the masked output $(r_i \oplus f(x))$ where $f(x)$ is the carry output. The entire LUT function for each output can fit into a single LUT to reduce the effects of glitches [9]. To further reduce the impact of glitches, the hardware stores each LUT output in a flip-flop. These are validated empirically in Section VI.

The outputs of an LUT are sent to the next LUT in chain and the next masked carry is computed accordingly. From the second LUT and onward, each LUT has to also take the masked carry and mask value generated from the prior step. The output r_o is simply the input r_i like a forward bypass, because the mask value is also needed for the next computation. This way the circuit processes all the bits of the shares and finally outputs the last carry bit which decides the sign of the sum. Each LUT computation is masked by a fresh random number. More efficient designs may also be possible using a masked Kogge-Stone adder (e.g., by modifying the ones described in [41]).

Fig. 9 illustrates that the first LUT is a 3-bit input 2-bit output LUT because there is no carry-in for LSB, and all the subsequent LUTs have 5-bit inputs and 2-bit outputs since they also need previous stage outputs as their inputs. After the final carry is generated, which is also the sign bit of the sum, the hardware can binarize the sum to 1 or 0 based on whether the sign bit is 0 or 1 respectively. This is the functionality of the final LUT, which is different from the usual masked carry generators in the earlier stages.

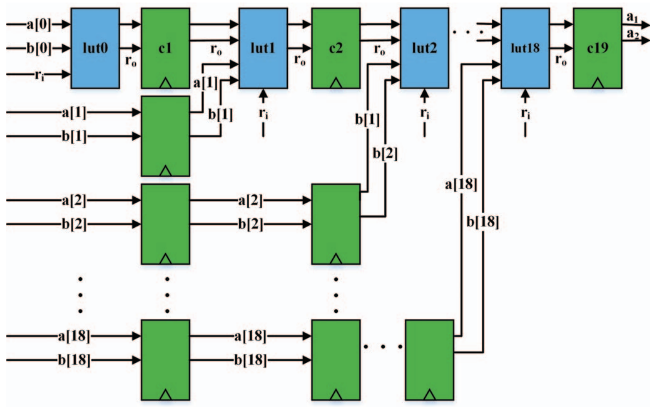

Fig. 9. Hardware design of the masked binarizer. It comprises of a chain of LUTs (lut0-lut18) denoted in blue, computing the carry in ripple carry fashion. Each LUT is masked by a fresh random number (r_i). The whole design is fully pipelined to maintain the original throughput by adding flip flops (in green) at each stage.

The circuit has 19 LUTs in serial; each LUT output is registered for timing and side-channel resilience against glitches. This design, however, adds a latency of 19 cycles to compute each activation value, increasing the original latency. Therefore, instead of streaming each of the 19 bits on the top row of LUTs sequentially in Fig. 9, the entire 19 bit sum is registered in the first stage, and each bit is used sequentially throughout the 19 cycles. This avoids the 19 cycle wait time for consecutive sums and brings back the throughput to 1 activation per cycle.

C. Boolean to Arithmetic Share Conversion

Each layer generates 1024 pairs of Boolean shares, which requires two changes in the hardware. First, the adder tree supports 784 inputs which cannot directly process 1024 shares. Second, the activation values are in the form of two Boolean shares while the masking of adder tree requires arithmetic shares as discussed in Section V-A. Using the same strategy of the unmasked design, the hardware adds 1024 1-bit shares pairwise to produce 512 2-bit shares before sending them to the adder tree. To resolve the conversion of Boolean to arithmetic conversion, the hardware can generate R such that

$$x = x_1 \oplus x_2 \tag{8}$$

$$x = R + x_2 \tag{9}$$

Using masked LUTs, the hardware performs signed addition of 1024 shares to 512 shares, and it also produces the arithmetic shares. The LUTs take in two consecutive activation values already multiplied by the corresponding weights, and a 2-bit signed random number to generate the arithmetic shares. Since multiplication in binary neural network translates to an XNOR operation [33], the hardware performs an XNOR operation using the activation value with its corresponding weight before sending it to the LUT. Since the activation value is in the form of 2 Boolean shares, the hardware performs XNOR only on one of the shares as formulated below:

$$a \overline{\oplus} w = b \tag{10}$$

$$a = a_1 \oplus a_2 \tag{11}$$

$$(a_1 \oplus a_2) \overline{\oplus} w = (a_1 \overline{\oplus} w) \oplus a_2 \tag{12}$$

The LUTs have five inputs: two shares that are not XNORed, two shares that are XNORed and a 2-bit signed random number. If the actual sum of the two consecutive nodes is a_i, then the LUT outputs r_i and a_i-r_i range from -2 to +1 since it is a 2-bit signed number and weighted sum of two nodes will range from -2 to +2. Therefore, a_i-r_i can range from -3 to +4 and should be 4-bit wide. The hardware has 512 LUTs that convert the 1024 pairs of Boolean shares to 512 pairs of arithmetic shares. After the conversion, the hardware reuses the same adder tree masking that was described in Section V-A. The arithmetic shares have a leakage in MSB as discussed in Subsection V-A1, but because the same WDDL style adder tree is reused, this is addressed for all layers.

D. Output Layer

In the output layer, for an unmasked design, the index of the node with maximum confidence score is generated as the output of the neural network. In the masked case, however, the confidence values are split in two arithmetic shares, which by definition cannot be combined. Equations (14–16) formulate the masking operations of the output layer. Basically, we check if the sum of two numbers is greater than the sum of another two numbers, without looking at the terms of each sum at the same time. Therefore, instead of adding the two shares of the confidence values and comparing them, we subtract one share of a confidence value from another share of the other confidence value. This still solves the inequality, but looks at the shares of two different confidence scores.

$$a_1 + a_2 \geq b_1 + b_2 \tag{13}$$

$$a_1 - b_2 \geq b_1 - a_2 \tag{14}$$

$$(a_1 - b_2) + (a_2 - b_1) \geq 0 \tag{15}$$

This simplifies the original problem to the previous problem of finding the sign of the sum of two numbers without combining them. Hence, in the final layer computation, the hardware reuses the masked carry generator explained in Section V-B.

E. The Entire Inference Engine

Fig. 10 illustrates all the components of the masked neural network. First, the network arithmetically masks the input pixel a_i using fresh r_i generated by the PRNG. Next, the WDDL style adder tree processes each of the masks (r_i) and the masked values ($a_i - r_i$) in two sequential phases. The demultiplexer at the adder tree output helps to buffer the first phase final summations, and pass the second phase summations to the masked activation function directly. The masked activation function produces the two Boolean shares of the actual activation using fresh randomness from the PRNG. The network XNORs one share with the weights and sends the second share directly to the Boolean to arithmetic converters. The converters produce 512 arithmetic shares from the 1024 Boolean shares using random numbers generated by the PRNG. The hardware feeds the arithmetic shares $a_i - r_i$

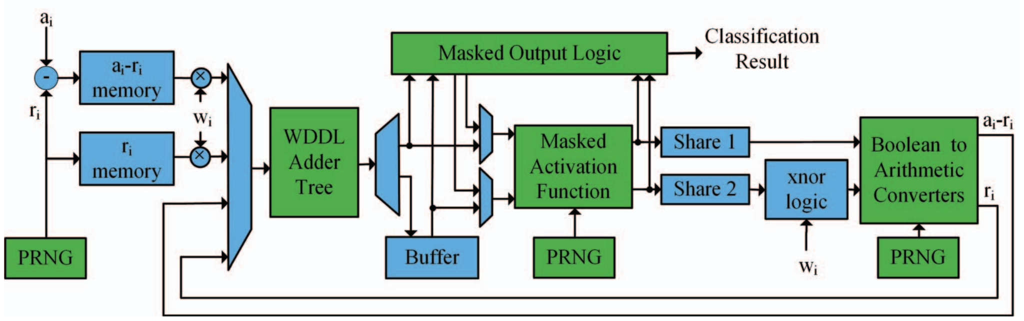

Fig. 10. Components of the masked BNN. Blocks in green represent the proposed masking blocks.

and r_i to the adder tree and repeats the whole process for each layer. Finally, the output layer reuses the masked activation function to find the node with maximum confidence from the arithmetic shares of the confidence values and computes the classification result output as described in Subsection V-D.

VI. Leakage and Overhead Evaluation

This section describes the measurement setup for our experiments, the evaluation methodology used to validate the security of the unprotected and protected implementations, and the corresponding results with overhead quantification.

A. Hardware Setup

Our evaluation setup used the SAKURA-X board [42], which includes a Xilinx Kintex-7 (XC7K160T-1FBGC) FPGA for processing and enables measuring the voltage drop on a 10mΩ shunt-resistance while making use of the on-board amplifiers to measure FPGA power consumption. The clock frequency of the design was 24MHz. We used the Picoscope 3206D oscilloscope to take measurements with the sampling frequency set to 250MHz. To amplify the output of the Kintex-7 FPGA, we used a low-noise amplifier provided by Riscure (HD24248) along with the current probe setup. The experiment can also be conducted at a lower sampling rate by increasing the number of measurements [43].

B. Side-Channel Test Methodology

Our leakage evaluation methodology is built on the prior test efforts on cryptographic implementations [44], [25], [40]. We performed DPA on the 4 main operations of an inference engine as stated before, viz. adder tree, activation function, Boolean to arithmetic share conversion, and output layer max function. Pseudo Random Number Generators (PRNG) produced the random numbers required for masking—any cryptographically-secure PRNG can be employed to this end. We first show the first-order DPA weight recovery attacks on the masked implementation with PRNG disabled. With PRNG off, the design's security is equivalent to that of an unmasked design. We illustrate that such a design leaked information for all the three operations, which ensured that our measurement setup and recovery code was sound. Next, we turned on the PRNG and performed the same attack which failed for all the operations. We also performed a second-order attack to validate the number of traces used in the first-order analysis. The power model was based on hamming distance of registers that was generated using our HD simulator for the neural network and the tests used the Pearson correlation coefficient to compare the measurement data with the hypothesis. In practice, we leave it as an open problem to use more advanced tools like profiled attacks (model-less), MCP-DPA attacks [45] for the masking parts, information theoretic tools (MI/PI/HI) [46], and more advanced high-dimensional attacks/filtering or post-processing.

C. Attacks with PRNG off

The PRNG generates the arithmetic shares for the adder tree, feeds the masked LUTs of the activation function and the Boolean to arithmetic converters. Turning off the PRNG unmasks all these operations making a first-order attack successful at all these points. Fig. 11 shows the mean power plot on the top for orientation, which is followed below by the 3 attack plots with PRNG disabled. We attack the second stage of the adder tree, the first node's activation result, and the first node of the output layer, respectively, shown in the next three plots. In all the plots, we observe a distinct correlation peak for the targeted variable corresponding to the correct weight and bias values. Fig. 12 shows the evolution of the correlation coefficient as the number of traces increase. The attack is successful at 200 traces with PRNG off. This validates our claim on the vulnerability of the baseline, unprotected design.

D. First-order Attacks with PRNG on

We used the same attack vectors from the case of PRNG off, but with the PRNG turned on this time. This armed all the countermeasures implemented for each operation. Fig. 11 shows that the distinct peaks seen in the PRNG OFF plots do not appear in the PRNG ON plots for first-order attacks. Fig. 12 shows that the first-order attack is unsuccessful with 100k traces. This validates our claim on the resiliency of the masked, protected design.

E. Second Order Attacks with PRNG on

We also performed a second-order DPA on the activation function to demonstrate that we used sufficient number of traces in the first-order attack. Again, we used the same attack vectors used in the first-order analysis experiments, but applied the attack on centered-squared traces. Fig. 11 shows that we observed a distinct correlation peak at the correct point in time. Fig. 12 shows the evolution of the correlation coefficient for the second-order attack. We can see that the attack is successful around 3.7k traces which confirms that 100k traces are sufficient for a first-order attack.

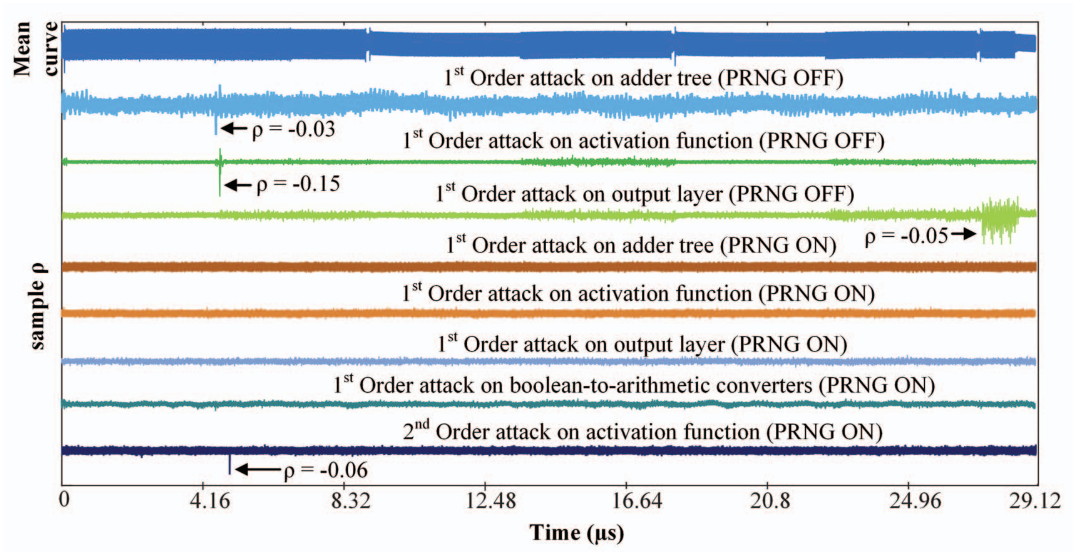

Fig. 11. Side-channel evaluation tests. First-order attacks on the unmasked design (PRNG off) show that it leaks information while the masked design (PRNG on) is secure. The second-order attack on the masked design succeeds and validates the number of measurements used in the first-order attacks.

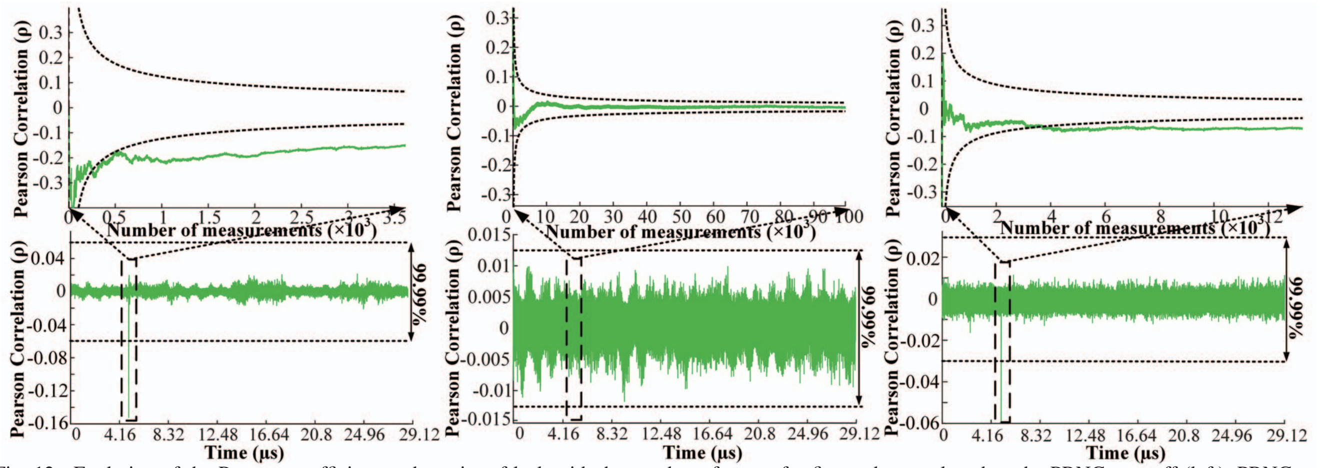

Fig. 12. Evolution of the Pearson coefficient at the point of leak with the number of traces for first-order attacks when the PRNGs are off (left), PRNGs are on (middle), and for second order attacks with PRNGs on (right). The first-order attack with PRNGs off succeeds around 200 traces but fails when PRNGs are on even with 100k traces, which shows that the design is masked successfully. The second-order attack becomes successful around 3.7k traces, which validates that we used sufficient number of traces in the first-order attacks.

F. Attacks on Hiding

We applied a difference of means test with 100k traces to test the vulnerability of hiding used for the MSB of arithmetic shares. The partition is thus based on the binary value of MSB. Fig. 13 shows the attack on the targeted clock cycle and quantifies that after 40k traces the adversary is able to distinguish the two cases with 99.99% confidence. Note that this number is significantly higher than the number of measurements required to succeed for the second order attack. The number of traces for all the attacks are relatively low due to the very friendly scenario created for the adversary; the platform is low noise. In a real-life setting, the noise would be much higher and consequently all attacks would require more traces.

G. Masking Overheads

Table II summarizes the area and latency overheads of masking in our case. As expected, due to the sequential processing with the two shares in the masked implementation, the inference latency is approximately doubled, from 3192 cycles for the baseline to 7248 cycles. Table II also compares the area utilization of the unmasked vs. masked implementations in terms of the various blocks present in the FPGA. The increase in the number of the LUTs, flip flops and BRAMs in the design is approximately 2.7x, 1.7x and 1.3x. The significant increase in the number of LUTs is mainly due to the masked LUTs used to mask the activation function and convert the Boolean shares of each layer to arithmetic shares. The increase in the number of flip flops and BRAM utilization is caused by additional storage structures of the masked implementation such as the randomness buffered at the start to mask the various parts of the inference engine. Furthermore, the arithmetic masks are also stored in the first phase, to be sent together to the masked activation function later. Each layer also stores twice the number of activations in the form of two Boolean shares increasing the memory overhead.

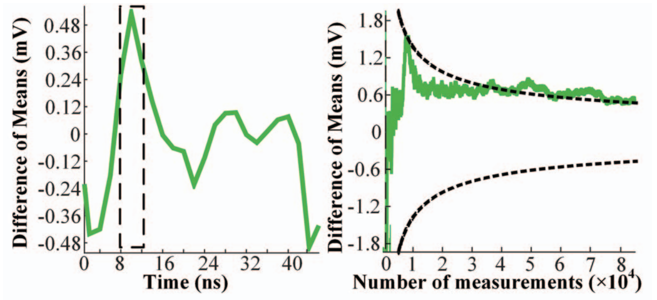

Fig. 13. The difference-of-means test on the WDDL based signed-bit computation. The figure shows that the difference of means between the power traces corresponding to MSB=0 and MSB=1 cases cross the 99.99% confidence interval (represented by the dotted lines) around 40k traces.

TABLE II
AREA AND LATENCY COMPARISON OF UNMASKED VS. MASKED IMPLEMENTATIONS

Design Type	LUT/FF	BRAM/DSP	Cycles
Unmasked	20296/18733	81/0	3192
Masked	55508/33290	111/0	7248

VII. DISCUSSIONS

This section discusses the orthogonal aspects together with the limitations of our approach and comments on how they can be complemented to improve our proposed effort.

A. Limitations of The Proposed Defense

Masking is difficult—after 20 years of AES masking, there is still an increasing number of publications (e.g. CHES'19 papers [47], [48], [49]) on better/more efficient masking. This, in part, is due to ever-evolving attacks [50]. The paper's focus is on empirical evaluation of security. We provide a proof-of-concept which can be extended towards building more robust and efficient solutions. We emphasize the importance of theoretical proofs [51] and the need to conduct further research on adapting them to the machine learning framework.

We have addressed the leakage in the sign bit of arithmetic share generation of the adder tree through hiding for cost-effectiveness. This is the only part in our hardware design that is not masked and hence may be vulnerable due to composability issues or implementation imbalances (especially for sophisticated EM attacks [52]). We highlight this issue as an open problem, which may be addressed through extensions of gate level masking. But such an implementation will incur significant overheads in addition to what we already show.

Our evaluations build on top of model-based approaches, which can be corroborated with more sophisticated attacks such as template based [53], moments-correlating based [45], deep-learning based [54], or horizontal methods [55]. More research is needed to design efficient masking components for neural network specific computations, extending first-order masks to higher-order, and on investigating the security against such side-channel attacks.

B. Comparison of Theoretical Attacks, Digital Side-Channels, and Physical Side-Channels

We argue that a direct comparison of the physical side-channels to digital and theoretical attack's effectiveness (in terms of number of queries) is currently unfair due to *immaturity* of the model extraction field and due to different target algorithms. Analyzing and countering theoretical attacks improve drastically over time. This has already occurred in cryptography: *algorithms have won* [56]. Indeed, there has been no major breakthrough on the cryptanalysis of encryption standards widely-used today. But side-channel attacks are still commonplace and are even of growing importance. While digital side-channels are imminent, they are relatively easier to address in application-specific hardware accelerators/IPs that enforce constant time and constant flow behavior (as opposed to general purpose architectures that execute software). For example, the hardware designs we provide in this work has no digital side-channel leaks. Physical side-channels, by contrast, are still observed in such hardware due to their data-dependent, low-level nature; and therefore require more involved mitigation techniques.

C. Scaling to other Neural Networks

The primary objective of this paper is to provide the first proof-of-concept of both power side-channel attacks and defenses of NNs in hardware. To this end, we have designed a neural network that encompasses all the basic features of a binarized neural network, like binarized weights and activations, and the commonly used sign function for non-linearity. When extended to other neural networks/datasets, like CIFAR-10, the proposed defences will roughly scale linearly with the node, layer count and bit-precision (size) of neurons. To deploy the countermeasures on constrained devices, the area overheads can be traded off for throughput, or vice versa. Any algorithm, independent of its complexity, can be attacked with physical side-channels. But the attack success will depend on the parallelization level in hardware. In a sequential design, increasing the weight size (e.g. moving from one bit to 8-bits or floating point) may even improve the attack because there is more signal to correlate.

VIII. CONCLUSION

Physical side-channel leaks in neural networks call for a new line of side-channel analysis research because it opens up a new avenue of designing countermeasures tailored for the deep learning inference engines. In this paper, we provided the first effort in mitigating the side-channel leaks in neural networks. We primarily apply masking style techniques and demonstrate new challenges together with opportunities that originate due to the unique topological and arithmetic needs of neural networks. Given the variety in neural networks with no existing standard and the apparent, enduring struggle for masking, there is a critical need to heavily invest in securing deep learning frameworks.

IX. ACKNOWLEDGEMENTS

We thank the anonymous reviewers of HOST for their valuable feedback and to Itamar Levi for helpful discussions. This project is supported in part by NSF under Grants No. 1850373 and SRC GRC Task 2908.001.

REFERENCES

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in cryptology–CRYPTO'99*. Springer, 1999, pp. 789–789.

[2] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. New York, NY, USA: ACM, 2004, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/1014052.1014066

[3] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, March 2016, pp. 372–387.

[4] G. Wang, T. Wang, H. Zheng, and B. Y. Zhao, "Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 239–254. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/wang

[5] I. Kanter, W. Kinzel, and E. Kanter, "Secure exchange of information by synchronization of neural networks," *Europhysics Letters (EPL)*, vol. 57, no. 1, pp. 141–147, jan 2002. [Online]. Available: https://doi.org/10.1209%2Fepl%2Fi2002-00552-9

[6] M. Courbariaux and Y. Bengio, "BinaryNet: training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[7] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.

[8] J.-S. Coron and L. Goubin, "On boolean and arithmetic masking against differential power analysis," in *Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 231–237.

[9] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *Information and Communications Security*, P. Ning, S. Qing, and N. Li, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 529–545.

[10] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, ser. KDD '05. New York, NY, USA: ACM, 2005, pp. 641–647. [Online]. Available: http://doi.acm.org/10.1145/1081870.1081950

[11] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proceedings of the 25th USENIX Conference on Security Symposium*, ser. SEC'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 601–618. [Online]. Available: http://dl.acm.org/citation.cfm?id=3241094.3241142

[12] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 506–519. [Online]. Available: http://doi.acm.org/10.1145/3052973.3053009

[13] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 36–52.

[14] M. Juuti, S. Szyller, A. Dmitrenko, S. Marchal, and N. Asokan, "PRADA: protecting against DNN model stealing attacks," *CoRR*, vol. abs/1805.02628, 2018. [Online]. Available: http://arxiv.org/abs/1805.02628

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

[16] S. Tople, K. Grover, S. Shinde, R. Bhagwan, and R. Ramjee, "Privado: Practical and secure DNN inference," *CoRR*, vol. abs/1810.00602, 2018. [Online]. Available: http://arxiv.org/abs/1810.00602

[17] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," *CoRR*, vol. abs/1808.04761, 2018. [Online]. Available: http://arxiv.org/abs/1808.04761

[18] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: GPU side channel attacks are practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 2139–2153. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243831

[19] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI neural network: Using side-channels to recover your artificial neural network information," *CoRR*, vol. abs/1810.09076, 2018. [Online]. Available: http://arxiv.org/abs/1810.09076

[20] ——, "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 515–532. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/batina

[21] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, Apr 2011. [Online]. Available: https://doi.org/10.1007/s13389-011-0006-y

[22] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 1111–1116.

[23] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *2018 IEEE Symposium on Security and Privacy, SP 2018, IEEE Computer Society*, 2018, pp. 839–854.

[24] C. Ramesh, S. B. Patil, S. N. Dhanuskodi, G. Provelengios, S. Pillement, D. Holcomb, and R. Tessier, "FPGA side channel attacks without physical access," in *International Symposium on Field-Programmable Custom Computing Machines*, Boulder, United States, Apr. 2018, p. paper#116. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01787439

[25] J. Balasch, B. Gierlichs, O. Reparaz, and I. Verbauwhede, *DPA, Bitslicing and Masking at 1 GHz*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 599–619.

[26] T. Eisenbarth, C. Paar, and B. Weghenkel, *Building a Side Channel Based Disassembler*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 78–99. [Online]. Available: https://doi.org/10.1007/978-3-642-17499-5_4

[27] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM side channels and their use to extract private keys," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 305–316.

[28] M. S. İnci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, *Cache Attacks Enable Bulk Key Recovery on the Cloud*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 368–388.

[29] J.-B. Note and E. Rannaud, "From the bitstream to the netlist," in *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, ser. FPGA '08. New York, NY, USA: ACM, 2008, pp. 264–264. [Online]. Available: http://doi.acm.org/10.1145/1344671.1344729

[30] F. Benz, A. Seffrin, and S. A. Huss, "Bil: A tool-chain for bitstream reverse-engineering," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2012, pp. 735–738.

[31] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 3–18.

[32] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC '18. New York, NY, USA: ACM, 2018, pp. 393–406. [Online]. Available: http://doi.acm.org/10.1145/3274694.3274696

[33] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[34] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 65–74. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021744

[35] E. Nurvitadhi, D. Sheffield, Jaewoong Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of

FPGA, CPU, GPU, and ASIC," in *2016 International Conference on Field-Programmable Technology (FPT)*, Dec 2016, pp. 77–84.

[36] H. Yonekawa and H. Nakahara, "On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an FPGA," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2017, pp. 98–105.

[37] T. S. Messerges, "Using second-order power analysis to attack DPA resistant software," in *Cryptographic Hardware and Embedded Systems — CHES 2000*, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 238–251.

[38] M.-L. Akkar and L. Goubin, "A generic protection against high-order differential power analysis," in *Fast Software Encryption*, T. Johansson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 192–205.

[39] K. Tiri and I. Verbauwhede, "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, Feb 2004, pp. 246–251 Vol.1.

[40] O. Reparaz, S. Sinha Roy, F. Vercauteren, and I. Verbauwhede, *A masked Ring-LWE implementation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 683–702.

[41] T. Schneider and A. Moradi, "Arithmetic addition over boolean masking towards first- and second-order resistance in hardware."

[42] Y. Hori, T. Katashita, A. Sasaki, and A. Satoh, "SASEBO-GIII: A hardware security evaluation board equipped with a 28-nm fpga," in *The 1st IEEE Global Conference on Consumer Electronics 2012*, Oct 2012, pp. 657–660.

[43] D. R. E. Gnad, J. Krautter, and M. B. Tahoori, "Leaky noise: New side-channel attack vectors in mixed-signal IoT devices," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 3, pp. 305–339, May 2019. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8297

[44] O. Reparaz, R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Additively homomorphic ring-LWE masking," in *Post-Quantum Cryptography*, T. Takagi, Ed. Cham: Springer International Publishing, 2016, pp. 233–244.

[45] A. Moradi and F.-X. Standaert, "Moments-correlating DPA," in *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, ser. TIS '16. New York, NY, USA: ACM, 2016, pp. 5–15. [Online]. Available: http://doi.acm.org/10.1145/2996366.2996369

[46] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2008, pp. 426–442.

[47] G. Cassiers and F.-X. Standaert, "Towards globally optimized masking: From low randomness to low noise rate," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, pp. 162–198, Feb. 2019. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7389

[48] T. Sugawara, "3-share threshold implementation of AES s-box without fresh randomness," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 123–145, Nov. 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7336

[49] L. De Meyer, V. Arribas, S. Nikova, V. Nikov, and V. Rijmen, "MM: Masks and macs against physical attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 25–50, Nov. 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7333

[50] I. Levi, D. Bellizia, and F.-X. Standaert, "Reducing a masked implementation's effective security order with setup manipulations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, pp. 293–317, Feb. 2019. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/7393

[51] T. Moos, A. Moradi, T. Schneider, and F.-X. Standaert, "Glitch-resistant masking revisited," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 256–292, 2019.

[52] V. Immler, R. Specht, and F. Unterstein, "Your rails cannot hide from localized EM: how dual-rail logic fails on FPGAs," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 403–424.

[53] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 13–28.

[54] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2016, pp. 3–26.

[55] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil, "Horizontal correlation analysis on exponentiation," in *International Conference on Information and Communications Security*. Springer, 2010, pp. 46–61.

[56] P. Kocher, "Obvious in hindsight: From side channel attacks to the security challenges ahead," 2016, invited talk at CHES & Crypto 2016. [Online]. Available: https://iacr.org/workshops/ches/ches2016/presentations/CHES16_invited.pdf