




HWGN²: Side-Channel Protected NNs Through Secure and Private Function Evaluation

Mohammad Hashemi¹(✉) , Steffi Roy², Domenic Forte², and Fatemeh Ganji¹

¹ Worcester Polytechnic Institute, Worcester, MA 01609, USA
{mhashemi, fgangi}@wpi.edu

² University of Florida, Gainesville, FL 32611, USA
steffiroy@ufl.edu, dforte@ece.ufl.edu

<https://www.ece.ufl.edu/people/faculty/domenic-forte/>,
<http://vernam.wpi.edu/>

Abstract. Recent work has highlighted the risks of intellectual property (IP) piracy of deep learning (DL) models from the side-channel leakage of DL hardware accelerators. In response, fundamental cryptographic approaches, specifically built upon the notion of secure and private function evaluation, could potentially improve the robustness against side-channel leakage. To examine this and weigh the costs and benefits, we introduce hardware garbled NN (HWGN²), a DL hardware accelerator implemented on FPGA. HWGN² also provides NN designers with the flexibility to protect their IP in real-time applications, where hardware resources are heavily constrained, through a hardware-communication cost trade-off. Concretely, we apply garbled circuits, implemented using a MIPS architecture that achieves up to 62.5× fewer logical and 66× less memory utilization than the state-of-the-art approaches at the price of communication overhead. Further, the side-channel resiliency of HWGN² is demonstrated by employing the test vector leakage assessment (TVLA) test against both power and electromagnetic side-channels.

Keywords: Side-channel analysis · Deep learning · Secure function evaluation · Private function evaluation

1 Introduction

An ever-increasing number of applications are demanded from machine learning and, in particular, deep learning (DL). These applications, among other compute-intensive services, have been supported by cloud platforms equipped with hardware acceleration [9]; however, cloud platforms are not the only hosts of DL algorithms and modules. IoT edge devices have embodied modules to perform many tasks, for instance, image classification or speech recognition as required by wearable devices for augmented reality and virtual reality [27]. In addition to those, so-called mobile and wearable devices, low-cost DL chips (e.g., sensors

or actuators) have been employed to support DL-inference in cameras, medical devices, appliances, autonomous surveillance, ground maintenance systems, and even toys. Under DL-inference scenarios, trained neural networks (NNs) are made available to users. To obtain such a trained NN, a large training dataset is used in a time-consuming process to tune NN hyperparameters, which cannot be repeated in a straightforward manner. Therefore, it can be tempting for an adversary to target the DL-inference accelerator and extract those parameters. Besides hyperparameters, the architecture of NNs is another asset to protect as it may (even partially) reveal private information [16,17] or at least help the adversary to reconstruct the NN [1,4]. Since physical access can make it further easier for attackers to reverse-engineer and disclose the assets (i.e., architecture and hyperparameters) corresponding to NNs, usual protections, e.g., blocking binary readback, blocking JTAG access, code obfuscation, etc. could be applied to prevent binary analysis [4]. These, of course, would not stop an attacker from leveraging the information that leaks through side-channels [4,14,52,55].

These attacks have resulted in considerable efforts to devise countermeasures. Intuitively, masking schemes developed to protect cryptographic modules against SCA have been one of the first solutions discussed in the literature [12,13]. These methods come with their own set of challenges, e.g., being limited to a pre-defined level of security associated with the masking order or even to a particular modality. Moreover, evidently, masking cannot stop the attacker from disclosing the architecture of the NN under attack. The natural question to be asked is why fundamental cryptographic concepts that can provide NNs with robustness against SCA have not yet been examined. Concretely, secure function evaluation (SFE), specifically garbled circuits evaluation, has been considered to prevent side-channel leakage cf. [21,33]. Nevertheless, in practice, SFE has not been considered to stop side-channel attacks, perhaps, due to the high overhead initially observed in [21]. Their implementation on a field-programmable gate array (FPGA) is a combination of tamper-resistant hardware with Yao's garbling scheme [53], which comes with an overhead of about factor $10^6 \times$ compared to an unprotected AES embedded in an FPGA.

Apart from the leakage properties of SFE and its realization garbled circuits, they have been developed to ensure the security of users' data, when two parties jointly evaluate a known function. Therefore, in a natural way, garbled circuits have been investigated to put forward the notion of privacy-preserving inference-as-a-service [39,40]. In spite of these results, the gap between these studies is evident: design of countermeasures against SCA, software implementation of garbled NNs [20,39,41], and hardware implementations of garbled circuits [44]. To narrow this gap, this paper introduces HWGN² (hardware garbled NN) and contributes to the following aims.

- *A secure and private DL-inference hardware accelerator, resilient to SCA.* To protect the NN model (including its architecture and parameters) against SCA, HWGN² relies on the principles of private function evaluation (PFE) and SFE, realized through a general purpose processor cf. [44,46]. Interestingly enough, as opposed to the argument in [21,33] suggesting the side-channel resiliency of

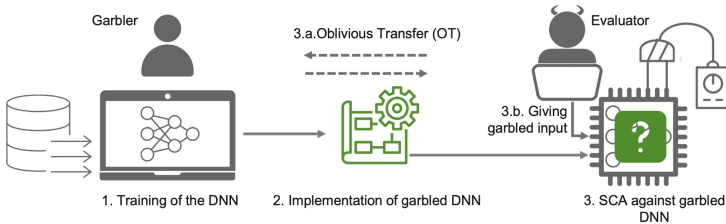


Fig. 1. HWGN² framework: The process begins with training the NN as done for a typical DL task. The second step corresponds to the implementation of the garbled NN hardware accelerator along with running the OT protocol. The end-user poses the accelerator and attempts to collect the side-channel traces to extract information on the NN (architecture, hyperparameters, etc.).

garbled circuits, Levi et al. have recently demonstrated a side-channel attack against garbling schemes leveraging the free-XOR optimization [28]. HWGN² is not susceptible to this attack since PFE is taken into account to make the function private. It is noteworthy that the privacy of the NN model is understudied even in existing software garbled DL-inference [2, 39, 41]. Our instruction set-based HWGN² is model-agnostic. Moreover, in the most cost-efficient setting with a DL-inference realized by using XNOR operators, our implementation does not require any modification to the NN, in contrast to what has been proposed as software garbled DL-inference [39].

- *Effectiveness and cost-efficiency of SCA protection relying on SFE/PFE.* To evaluate the feasibility of our approach, we identify two implementation scenarios, namely (1) resource- and (2) communication-efficient. In the first category, compared to the unprotected NN, the overhead is up to $0.0011\times$ and $0.018\times$ more logical and memory hardware resources, respectively; however, this relatively low overhead is achieved at the cost of communication between the user and the inference service provider. If communication constitutes a burden on the system, it can be dealt with, even though compared to the unprotected design, the overhead increases to $52.4\times$ and $40.8\times$ more logical and memory hardware resources, respectively. However, even under the communication-efficient scenario, HWGN² utilizes up to $62.5\times$ fewer logical and $66\times$ less memory, respectively, compared to the most relevant study [41]. Additionally, the side-channel resiliency of HWGN² implementation on the FPGA is assessed by applying T-test leakage detection.

2 Adversary Model

Valuable assets of NNs, as intellectual property (IP), include their NN architectures, hyperparameters, and the parameters critical to achieving reasonable accuracy [4]. On the other hand, these NNs might be used in applications in which their inputs contain sensitive information (e.g., medical or defense records [34]). Hence, the security of inputs given to NNs along with the privacy of the networks themselves must be guaranteed. Note that here the definitions of security and privacy

are borrowed from SFE- and private function evaluation- (PFE) related literature [6]. Classically, two threat models have been considered in prior works in the contexts of SFE and PFE: (i) *semi-honest* (so-called Honest-but-curious (HbC)) and (ii) *malicious* (active) adversary. An HbC adversary is expected to follow the protocol execution and does not deviate from the protocol specifications. To be more specific, the HbC adversary may only be able to learn information without interfering with the protocol execution. On the contrary, a malicious adversary may attempt to cheat or deviate from the protocol execution specifications.

In our work, we consider the HbC adversary, whose role is played by Bob (i.e., the evaluator), whereas Alice is the garbler [6, 29] (see Fig. 1). Following the definition of the attack model presented in the state-of-the-art (SOTA), e.g., [12, 13], the DL model provider (garbler) trains the DL model in an offline fashion, and the evaluator performs the inference. It is important to stress that the hardware implementation encompasses solely the evaluator engine, i.e., neither garbling nor encryption module is implemented on the hardware platform. To evaluate the garbled DL accelerator, the evaluator feeds her garbled inputs prepared in an offline manner. The evaluator can collect power/EM traces from the device either via direct access or remotely, see e.g., [43, 56]. For this, the evaluator follows a chosen-plaintext-type attack model, where she sends her inputs to the device for classification and readily captures multiple traces. These traces will be then used to launch power/EM-based side-channel attacks [8, 10, 25]. The goal of the garbler is to protect the NN architectures, hyperparameters, and parameters from the HbC evaluator. HWGN² fulfills this requirement through SFE/PFE techniques (see Sect. 5).

3 Related Work

3.1 SCA Against NNs

The main goals of SCA targeting DL hardware accelerators can be: (i) extraction of model architectures, and (ii) revealing NN parameters (i.e., weights and biases). For this purpose, Xiang et al. [52] presented a power side-channel attack to extract the model architectures. Using these power consumption models built for different model components, an SVM-based classifier was trained to reveal the model architectures running on the hardware accelerator. This line of research has also been pursued by Batina et al. [4] who introduced an attack scenario based on the EM and timing side-channel to extract the number of layers, the number of neurons in each layer, weights, and activation functions (AF). First, they modeled the timing side-channel of all possible AF (e.g., Relu or Tanh) and extracted the AF used in the NN by comparing the response time of the DL hardware accelerator when it executed the AF and the timing model of each possible AF. This is followed by analyzing EM traces captured when the DL hardware accelerator runs, where the EM patterns determine the number of layers and number of neurons in each layer. By feeding different random inputs to the accelerator and capturing the EM traces, it was possible to launch a Correlation Power Analysis (CPA) to reveal the weights. In another approach,

Table 1. SOTA approaches vs. HWGN² (**P**arameters Secrecy of DL Model. **U**ppgrade-able to/supporting malicious security model. **A**rchitecture protection of DL model. **C**onstant-round complexity. **I**ndependence of a secondary server). Inspired by [39].

Approach	P	U	A	C	I
DeepSecure [42]	✓	✓	✗	✓	✓
Chameleon [40]	✓	✗	✗	✗	✗
XONN [39]	✓	✓	✗	✓	✓
BoMaNET [13]	✓	✗	✗	✓	✓
ModuloNET [12]	✓	✗	✗	✓	✓
TinyGarble2 [20]	✓	✓	✗	✓	✓
RedCrypt [41]	✓	✓	✗	✓	✗
HWGN² [This paper]	✓	✓	✓	✓	✓

Breier et al. [7] have presented a reverse engineering attack to extract the DL model weights and biases (parameters) with the help of fault injection on the last hidden layer of the network (see Table 5 in Appendix A).

3.2 Security-Preserving DL Accelerators

To protect NNs against SCA, Liu et al. [32] introduced a shuffling and fake memory-based approach to mitigate reverse engineering attacks that increase the run time of a DL hardware accelerator when the depth of the NN increases. Regarding the similarity between SCA launched against cryptographic implementations and DL accelerators, in a series of work, Dubey et al. have proposed hiding and masking techniques to protect NNs [12–14]. Yet, the differences between these implementations make the adaptation of known side-channel defenses challenging; for instance, integer arithmetic used in neural network computations that is different from modular arithmetic in cryptography, which has been addressed in [12, 14]. Despite the impressive achievements presented in these studies, the approaches suffer from the known limitations of masking, i.e., their restriction to a specific side-channel security order. Furthermore, the implementation of masked DL models (i.e., a new circuit should be designed/implemented for different NNs) would be a challenging task. Moreover, masking cannot protect the architecture of DL models.

3.3 Garbled Accelerators

Among proposals put forward to make SFE practical, GarbledCPU [46] and RedCrypt [41] are of great importance to our work since they consider a hardware implementation of garbled circuits, whereas other relevant studies such as [3, 20, 39, 40, 42, 44] devoted to software-based garbling engine/evaluator (other implementations have been compared in Table 6 in Appendix A). [46] has demonstrated a hardware garbling evaluator implemented on general-purpose sequential processors, where the privacy of NN architectures is also ensured.

While benefiting from the simplicity of programming a processor, their design is specific to Microprocessor without Interlocked Pipelined Stages (MIPS) architecture. This has been addressed by introducing ARM2GC framework, where the circuit to be garbled/evaluated is the synthesized ARM processor circuit that can support pervasiveness and conditional execution [45]. The efficiency in terms of hardware resources and communication cost has been reported as well.

RedCrypt attempts to enable cloud servers to provide high-throughput and power-efficient services to their clients in a real-time manner [41]. For this, FPGA platforms (Virtex UltraSCALE VCU108) have been used as a garbling core to present an efficient GC architecture with precise gate-level control per clock cycle, which ensures minimal idle cycles. This results in a multiple-fold improvement in the throughput of garbling operation compared to the previous hardware garbled circuit accelerator [44, 46]. In their scenario, a host CPU is involved in an OT to communicate the evaluator labels/input with the client, which may need high bandwidth. Although RedCrypt [41] has achieved significant improvement in computational efficiency, the DL model implemented on the FPGA cannot be easily diversified. Their proposed hardware DL accelerator suits a specific type of DL model and is built on the assumption that the network architecture is publicly available, which allows an adversary to launch an SCA attack easier [4]. These shortcomings are tackled by HWGN² that is NN-agnostic and guarantees the privacy of the DL model, i.e., the secrecy of its architecture. A qualitative comparison between SOTA approaches and HWGN² is provided in Table 1. HWGN² shares similarities with TinyGarble2 [20], although they are software and hardware accelerators, respectively.

4 Background

4.1 SFE/PFE Protocols

SFE protocols enable a group of participants to compute the correct output of some agreed-upon function f applied to their secure inputs without revealing anything else. One of the commonly-applied SFE protocols is Yao’s garbled circuit [53], a two-party computation protocol. To formalize this protocol, we employ the notions and definitions provided in [6] to support modular and simple but effective analyses. In this regard, a garbling algorithm Gb is a randomized algorithm, i.e., involves a degree of randomness. $Gb(f)$ is a triple of functions $(F, e, d) \leftarrow Gb(f)$ that accepts the function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ and the security parameter k . $Gb(f)$ exhibits the following properties. The encoding function e converts an initial input $x \in \{0, 1\}^n$ into a garbled input $X = e(x)$, which is given to the function F to generate the garbled output $Y = F(X)$. In this regard, e encodes a list of tokens (so-called labels), i.e., one pair for each bit in $x \in \{0, 1\}^n$: $En(e, \cdot)$ uses the bits of $x = x_1 \cdots x_n$ to select from $e = (X_0^1, X_1^1, \dots, X_n^0, X_n^1)$ and obtain the sub-vector $X = X_1^{x_1}, \dots, X_n^{x_n}$. By reversing this process, the decoding function d generates the final output $y = d(Y)$, which must be equal to $f(x)$. In other words, f is a combination of probabilistic functions $d \circ F \circ e$. More precisely, the garbling scheme

$G = (Gb, En, De, Ev, ev)$ is composed of five algorithms as shown in Fig. 2, where the strings d , e , f , and F are used by the functions De , En , ev , and Ev (see Sect. 5 for a concrete protocol flow in the case of NNs).

Security of Garbling Schemes: For a given scheme, the security can be roughly defined as the impossibility of acquiring any information beyond the final output y if the party has access to (F, X, d) . Formally, this notion is explained by defining the side-information function $\Phi(\cdot)$. Based on the definition of this function, an adversary cannot extract any information besides y and $\Phi(f)$ when the tuple (F, X, d) is accessible. As an example of how the function $\Phi(\cdot)$ is determined, note that for an SFE protocol, where the privacy of the function f is not ensured, $\Phi(f) = f$. Thus, the only thing that leaks is the function itself. On the other hand, when a PFE protocol is run, $\Phi(f)$ is the circuit/function’s size, e.g., number of gates.

Oblivious Transfer (OT): This is a two party protocol where party 2 transfers some information to party 1 (so-called evaluator); however, party 2 remains oblivious to what information party 1 actually obtains. A form of OT widely used in various applications is known as “chosen one-out-of-two”, denoted by 1-out-of-2 OT. In this case, party 2 has bits X^0 and X^1 , and party 1 uses one private input bit s . After running the protocol, party 1 only gets the bit X^s , whereas party 2 does not obtain any information on the value of s , i.e., party 2 does not know which bit has been selected by party 1. This protocol can be extended to support the n -bit case, where party 1 bits x_1, \dots, x_n are applied to the input of party 2 $X_1^0, X_1^1, \dots, X_n^0, X_n^1$ to obtain $X_1^{x_1}, \dots, X_n^{x_n}$. This is possible by sequential repetition of the basic protocol [6]. It has been proven that 1-out-of-2 OT is universal for 2-party SFE, i.e., OT schemes can be the main building block of SFE protocols [24].

4.2 Neural Networks (NNs)

An NN is one of the main categories of machine learning, referring to learning a non-linear function through multiple layers of neurons with the goal of predicting the output corresponding to a given input. To perform such prediction, the input is fed to the first layer of the network (so-called *input layer*), whereas in the next layers (so-called *hidden layers*) the abstraction of the data takes place. For a *multi-layer perceptron* (MLP) that is a fully connected NN, each layer’s input (including the input layer) is multiplied by neuron weights, added to the bias, and finally given to a commonly-applied *activation functions* at the output of each layer (excluding the input layer), e.g., Sigmoid, Tanh, and Rectified Linear Unit (ReLU). The activation functions that might be used in DL models include linear, Sigmoid, and softmax.

5 Foundations of HWGN²

Protocol Flow: Here we provide insight into how SFE/PFE schemes can be tailored to the needs of a secure and private DL accelerator. According to the general

flow illustrated in Fig. 2, the goal of a garbling protocol G is to evaluate a function f against some inputs x to obtain the output y . The evaluator (i.e., the attacker) is never in possession of the raw NN binaries. Let $f = f_{NN}$ denote the function corresponding to the NN. The attacker aims to obtain the information on f_{NN} by collecting the side-channel traces. To achieve this, here we give an example of SFE protocol G that has OT at its core and follows Yao’s garbling principle, i.e., the garbling protocol $G = (Gb, En, De, Ev, ev)$ as shown in Fig. 2. To execute the protocol, the designer of the NN accelerator (garbler) conducts $(F, e, d) \leftarrow Gb(1^k, f)$ on inputs 1^k and f and parses $(X_0^1, X_1^1, \dots, X_n^0, X_n^1) \leftarrow e$. Afterward, the garbler sends F to the evaluator, i.e., the attacker. In order to perform the function Ev , the attacker and the garbler run the OT, where the former has the selection string x and the latter party has already parsed $(X_0^1, X_1^1, \dots, X_n^0, X_n^1)$. Hence, the evaluator can obtain $X = X_1^{x_1}, \dots, X_n^{x_n}$ and consequently, $y \leftarrow De(d, Ev(F, X))$. Note that even with the tuple (F, X, d) in hand, the attacker cannot extract any information besides y and $\Phi(f)$. Moreover, although the NN provider has access to (F, e, d) , no information on x leaks. In an inference scenario, x represents the evaluator’s input data. Nevertheless, if G is an SFE scheme, $\Phi(f) = f$.

To construct a PFE scheme protecting the architecture, parameters, and hyperparameters of the NN that relies on the scheme G , we first define a polynomial algorithm Π that accepts the security parameter k and the (private) input of the party [6]. The PFE scheme is a pair $\mathcal{F} = (\Pi, ev)$, where ev is as defined for the garbling scheme (see Sect. 6 for more information about Π). The scheme \mathcal{F} enable us to securely compute the *class* of functions $\{ev(f, \cdot) : f \in \{0, 1\}^*\}$, i.e., any function that G can garble. The security of the PFE scheme \mathcal{F} relies on the security of the SFE protocol underlying \mathcal{F} (see Sect. 4.1); however, $\Phi(f)$ is the circuit size, i.e., the function f remains private when executing the SFE protocol. In other words, the NN, its architecture, parameters and hyperparameters are now kept private from the attacker.

Oblivious Inference: Oblivious inference tackles the problem of running the DL model on the user’s input without revealing the input or the result to the other party (i.e., garbler in our case). For the latter, another interesting characteristic of SFE/PFE schemes is their ability to adapt to specific scenarios, where the output y should also be protected. This would not be interesting in our case, where the security of the NN against SCA mounted by the evaluator is the objective. Nonetheless, for the sake of completeness, if the decryption of Y should be performed securely, the privacy of inference results can easily be preserved by applying a one-time message authentication code (MAC) to the output and XORing the result with a random input to hide the outcome. These operations can be included in the design of the NN and naturally increase its size and the input fed by the garbler; however, the increase is linear in the number of output bits and considered inexpensive [30].

5.1 Implementation of HWGN²

When defining the PFE scheme \mathcal{F} , it is mentioned that \mathcal{F} can securely and privately compute *any* function, which can be garbled by running the garbling

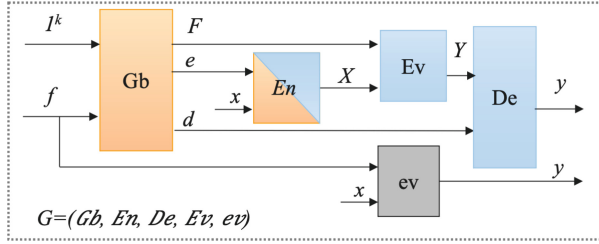


Fig. 2. A generic garbling scheme $G = (Gb, En, De, Ev, ev)$ cf. [6]. Our proposed secure and private DL accelerator is built upon G . For HWGN², the blocks in orange show the operations performed by the NN vendor, whereas the gray ones indicate the evaluator operations. ev denotes the typical, unprotected evaluation of the function f against the input x .

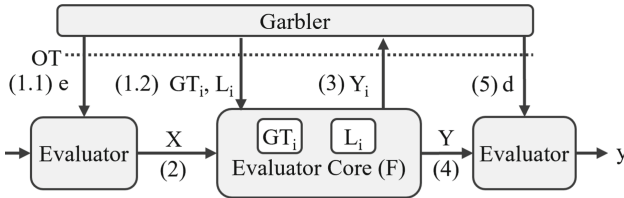


Fig. 3. Flow of HWGN² (L : garbled wire labels, GT : garbled tables, e and d : encryption and decryption labels, X : evaluator’s garbled input, Y : garbled output, Y_i, X_i, GT_i, L_i : garbled input, output, tables, wire labels corresponding to i^{th} sub-netlist, respectively, y : evaluator’s raw output).

scheme G . Our garbled universal circuit \mathcal{F} depends on the fact that a universal circuit is similar to a universal Turing machine [18], which can be realized by a general purpose processor cf. [44, 46]. Note the difference between our goal, i.e., realizing \mathcal{F} , and one achieved in [50]: optimizing the emulation of an entire *public* MIPS program. Although we implemented a MIPS-based scheme, the prototypes can be extended to ARM processors. HWGN² garbles the MIPS instruction set with a minimized memory and logical hardware resource utilization (see Sect. 5.1).

Similarities Between HWGN² and TinyGarble2: One of the state-of-the-art GC frameworks is TinyGarble2 [20] offering solely *software* DL inference, without ensuring the privacy of the NN. HWGN² remedies these shortcomings; however, it shares similarities with TinyGarble2, namely regarding the flow of the protocol. The technique presented in TinyGarble2 is based on the division of a large netlist, such as DL models, into i smaller sub-netlists and evaluating them one after another. The size of the sub-netlists could be either one gate or equal to the total number of gates in the f netlist. The fewer gates included in each sub-netlist, the less memory utilization the gates require to be evaluated.

Figure 3 illustrates the flow of HWGN² in the presence of an HbC adversary. First, the garbler chooses input encryption labels (e) (Step 1.1). Afterward, instead of sending the complete set of GTs and L to the evaluator, in each cycle

the garbler sends the evaluator a subset GT_i, L_i (Step 1.2), and either e (if the sub-netlist includes the gate with the inputs connected to the f netlist) or X_i (the garbled input corresponding to the sub-netlist). These subsets can be prepared offline and independent from the input of the evaluator. The evaluator also garbles her inputs as shown in Step 2, which is done offline as well. In the next step, the evaluator evaluates the gate and sends the garbler the garbled output Y_i , i.e., garbled output of the i^{th} sub-netlist (Step 3). This process repeats until all i sub-netlists, excluding the gates whose output is connected to the NN outputs, are evaluated. Then in Step 4, the garbler sends the garbled tables and labels related to the gates that are connected to the NN output (so-called NN output layer). After the evaluator evaluates all output layer-related gates, the garbler sends the decryption label (Step 5) along with the concatenated garbled outputs to the evaluator. Finally, the evaluator decrypts the concatenated garbled output Y and achieves his raw output y . Also, instead of sending the complete set of GTs, L, and e through one OT interaction, TinyGarble2 requires one OT interaction per sub-netlist. The trade-off of minimizing the memory utilization using TinyGarble2 is the communication cost.

What Makes HWGN² superior: Parallel and simultaneous evaluation of all input gates might result in the side-channel leakages due to the secret collision; therefore, all input gates must be evaluated one after another without parallelization. However, the rest of the gates (without dependencies) have no information about the secrets, and thus, they can be evaluated simultaneously. Moreover, we have noticed that each gate evaluation (excluding reading/writing its inputs/output from/to the memory) requires one operation code (OP-code) which is an 8-bit part of a MIPS instruction. As we have assigned the reading and writing tasks to the memory handler module, it is possible to combine a set of four gates (non-input gates) and construct one modified MIPS instruction from them. In doing so, in the evaluation phase, all these four OP-codes can be executed using four parallel arithmetic logic units (ALU) on FPGA while this is an impractical task for central processing unit (CPU) due to its limited resources and operating system (OS) limitations. HWGN², contrary to the previous software and hardware accelerators including TinyGarble family [20, 44], leverages these parallelization techniques. It also gives the flexibility of tuning the communication costs and hardware resource utilization to the garbler (e.g., NN provider). In the applications where communication cost poses a limitation (such as real-time applications), one can implement DL hardware accelerators by sending the complete set of GTs, L, and e through one OT interaction. This minimizes the communication cost while hardware resources are utilized at the maximum amount. In contrast, in the application with the limitation of hardware resources, one can use the HWGN² that implements DL hardware accelerators with the sub-netlist size of one or a small number of gates. As opposed to TinyGarble2, HWGN² implementation is based on the garbled MIPS architecture, making the circuit private (i.e., no information about the NN architecture leaks) as explained next.

MIPS Evaluator in HWGN². As explained before, in order to ensure the privacy of the NNs, the Boolean function representing the NN (so-called netlist) is converted to a set of reduced instruction set computing (RISC) instruction

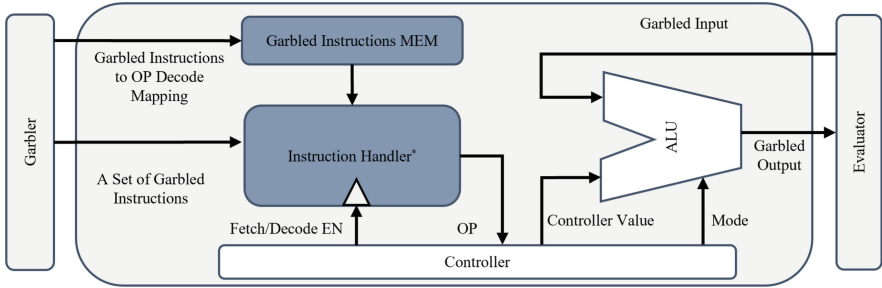


Fig. 4. Garbled MIPS evaluator, able to process any given number of instructions instead of a determined number. The black modules are extended and improved versions of memory and instruction handler in Lite_MIPS architecture [44]; the instruction handler prepares the controller sequence by comparing the garbled MIPS instructions and the OP mapping. The controller runs the process sequence by generating the ALU mode and executing the read and write operations.

set architecture (ISA) and evaluated on a core that executes the MIPS instructions [23]. It might be thought that a subset of instructions required to execute the NN is sufficient to be garbled in order to reduce the overhead; however, this could increase the probability of guessing which instructions are used and, consequently, violates the privacy of the NN.

To implement HWGN² on an FPGA, we modify Plasma [38] MIPS execution core emulating a RISC instruction set on the FPGA, to act as the garbled MIPS evaluator. Figure 4 illustrates the architecture of our garbled MIPS evaluator. The garbled evaluator receives three inputs: (i) a set of garbled instructions, (ii) the mapping for the instruction handler to fetch/decode the garbled instructions, and (iii) the evaluator’s garbled input. The combination of the first and second ones (i, ii) is the set of garbled tables and labels described before. Our garbled MIPS evaluator can evaluate the garbled MIPS instructions in two modes: (a) by receiving only one instruction and the operation code (OP) mapping and its corresponding instruction each cycle, i.e., the garbled evaluator with the capacity of one instruction per OT interaction, or (b) by receiving the complete set of instructions and their corresponding OP mapping at once. To achieve the resource-efficient implementation (mode i), we have modified the Lite_MIPS instruction handler module in a way that the memory size related to the received garbled instructions (not the OP code mapping) decreases from 128 cells to only one cell. The controller is further enhanced by discarding the unnecessary scheduler, SCD storage memory and its parsing modules and tailoring the core to need of only one instruction conversion per OT. Moreover, we include the erase state in the instruction MEM controller, which sets all memory blocks to 0 after converting each garbled instruction to the OP code. To take advantage of the resource-efficient implementation, an extra step should be taken to divide the netlist into the sub-netlists with the number of gates selected by the user. The sub-netlists are fed to HWGN² in the same order provided in the SCD file. This allows the user to make a trade-off between the resource efficiency and performance of the HWGN².

Table 2. Hardware resource utilization and OT cost of approaches applied against BM1.

Approach	LUT	FF	OT interaction
Plasma [38]	1773	1255	N/A
GarbledCPU [46]	21229	22035	2
RedCrypt [41] (One MAC Unit)	111000	84000	2
BoMaNET [13]	9833	7624	N/A
ModulaNET [12]	5635	5009	N/A
HWGN² (1 instruction per OT interaction)	1775	1278	2346
HWGN² Complete set of instructions	94701	52534	2

Specifically, in mode (a), in the first step, the instruction handler module receives one garbled instruction and OP mapping (all possible combinations of garbled MIPS instructions necessary to follow SFE protocol), which are stored in the instructions memory (MEM). In the next step, the instruction handler compares the given garbled instructions with garbled instructions MEM information and converts each garbled instruction to a set of OPs. Finally, the instruction handler sends the OP to the arithmetic-logic unit (ALU), erase instructions MEM, and repeats above-mentioned steps for the next garbled instructions. In mode (b), however, the instruction handler module works similarly to the Lite_MIPS architecture cf. [44]. As both instruction sets and decode mapping are garbled on the garbler side, the evaluator cannot decrypt the garbled instructions. Therefore, the garbler’s inputs and the DL model parameters are secure following the SFE and PFE protocols.

6 Evaluation of HWGN²

6.1 Resource Utilization

To understand the interplay between communication cost, hardware resources utilization, and performance, we have synthesized the garbled evaluator with the capacity of 1 and 2345 (complete set of instructions) garbled MIPS instructions per one OT interaction. We have used Xilinx Vivado 2021 to synthesize our design and generate a bitstream. To ensure the bitstream correctness, we have disabled place-and-route optimization and also utilize the DONT-TOUCH attribute. The garbling framework considered in our implementations is Just-Garble [5], also embedded in TinyGarble2 framework [20], which enjoys garbling optimization techniques such as Free-XOR [26], Row Reduction [36], and Garbling with a Fixed-key Block Cipher [5]. Our implementation is applied against three typical MLPs: the first one, with 784 neurons in its input layer, three hidden layers each with 1024 neurons, and an output layer with 10 neurons that is trained on MNIST (hereafter called **BM1**). The results for applying SOTA approaches against BM1 have been presented in [12, 41, 46]. The second MLP,

Table 3. Execution time and communication cost comparison between HWGN² and the SOTA approaches (for BM1). Results for [46] and HWGN² are reported based on FPGA with clock frequency equals to 12.5MHz. (N/R: not reported, inst.: instructions).

Approach	Time (Sec)	Communication (MB)
GarbledCPU [46]	1.74	N/R
RedCrypt [41]	0.63	5520
HWGN² (Complete set of inst. per OT interaction)	0.68	619
TinyGarble2 [20]	9.1	7.16
HWGN² (1 inst. per OT interaction)	3.25	12.39

BM2, has 784, 5, 5, and 10 neuron in its input, 2 hidden, and output layers, respectively. The third MLP, **BM3**, consists of 784, 6, 5, 5, and 10 neurons in its input, 3 hidden, and output layers, respectively.

Table 2 shows a comparison between the hardware utilization and OT cost of an unprotected MIPS evaluator core (Plasma [38]), HWGN² and the SOTA approaches applied to BM1. To give an insight into how much overhead cost the protection approaches impose, we have implemented Plasma core, an unprotected MIPS evaluator core on an Artix-7 FPGA. Note that we choose this architecture for the sake of a better comparison with the SOTA solutions, e.g., [12]. It is also worth mentioning that since the ultimate goal of our paper is to demonstrate the applicability of garbling techniques for side-channel resiliency, the network mentioned above is chosen to serve as a proof of concept. As the HWGN² processes the garbled instructions and inputs with the width of 32-bits, to have a fair comparison, we include the 32-bit MAC unit [41] in the resource utilization reported in Table 2.

In Table 2, BoMaNET and ModulaNET do not use OT to exchange their inputs. RedCrypt uses two OT interactions, one for the evaluator’s input and another for the evaluator’s output. However, in HWGN², in addition to the input and output labels exchange OT requirement, HWGN² requires M more OT interactions, where M is the number of sub-netlists. There is an important observation made from Table 2: HWGN² with the capacity of one instruction per OT interaction utilizes $0.0011\times$ and $0.018\times$ more logical and memory hardware resources, respectively, compared to an unprotected MIPS evaluator. The reason behind this efficiency is the size of instruction memory which stores only one instruction per OT interaction instead of the complete set of instructions. As mentioned in Sect. 5.1, to minimize resource utilization, one should sacrifice the communication cost, leading to an increased execution time. Hence, we set the size of the sub-netlist to just one gate, and every four gates are converted to a garbled instruction: $M = N_{gate}/4$, where N_{gate} is the number of gates in the netlist. In this setting, HWGN² requires $2 + 9380/4 = 2346$ OT interactions, where 9380 is the number of gates included in the BM1 netlist. In real-time applications where the execution time is the bottleneck, the OT interactions must be minimum [41]. Therefore, in Table 2, we also have reported the hardware resource utilization in two cases:

Table 4. Execution time and communication cost of HWGN² applied to BM1 accelerator and its XNOR-based implementation.

Architecture	#Instructions	OT Interaction	Execution Time (Sec)	Communication (MB)
BM1	2345	2346	3.25	12.39
XNOR-based BM1	1629	1631	2.31	9.71

(i) when the number of OT interactions is maximum (6th row) and (ii) when the number of OT interactions is minimum (7th row). The results in Table 2 are for the implementation of BM1. As shown in Table 3, HWGN² with the maximum performance is $2.5\times$ faster than GarbledCPU [46]. Performance of HWGN² is close to the performance of Redcrypt [41], the fastest SOTA approach, while utilizing $62.5\times$ fewer logical and $66\times$ less memory than Redcrypt [41].

Execution Time and Communication Cost Evaluation. To evaluate the cost of HWGN² in terms of execution time, we have used a machine with Intel Core i7-7700 CPU @ 3.60 GHz (GHz), 16 Gigabyte (GBs) RAM, and Linux Ubuntu 20 as the garbler and an ARTIX7 FPGA board as the evaluator, which has a clock frequency of 12.5 MHz (MHz). All the garbled instructions, their MEM values, and labels are generated offline and not included in the execution time. To communicate with the FPGA, for the sake of comparison, we have used HostCPU presented in [41]. Note that in a real-world application, where the communication is performed over high latency links, the protocol execution remains fast due to the constant number of rounds in Yao’s GC underlying our design cf. [22,31]. Moreover, we have used the EMP-toolkit [51] to establish the OT interaction between the garbler and the HostCPU. Table 3 shows the execution time and communication cost comparison between HWGN² and the SOTA approaches employed against BM1. The memory footprint of classical GC approaches is $O(I + N_{gate})$, where I is the number of input wires and N_{gate} is the number of gates in the netlist. In contrast, the memory footprint of HWGN² and TinyGarble2 is the same: $O(I + N_{gate,m} + i_m)$ where $N_{gate,m}$ is the number of gates in the largest among sub-netlists included in the design, and i_m is the number of inputs of the sub-netlist, which equals 1 and 2, respectively, in the case of HWGN² with the instruction capacity 1 per OT interaction.

To compare the execution time and communication cost of TinyGarbled2 with our approach, we have chosen the semi-honest mode when using their framework. HWGN² outperforms the TinyGarble2 implemented on CPU thanks to the parallel implementation made possible by the FPGA. On the other hand, when minimizing the OT interactions by investing more hardware resource utilization, HWGN² has a performance close to the RedCrypt with $62.5\times$ fewer logical and $66\times$ less memory utilization.

As an optimization technique, we have implemented the XNOR-based BM1. As the XOR operation is free in the garbling protocol [26], it is possible to decrease the size of the garbled netlists, which results in fewer instructions to be executed. Table 4 shows a comparison between two architectures. Using an

XNOR-based implementation of a DL hardware accelerator decreases the number of instructions, leading to a less OT cost and execution time. The only limitation of this optimization is that the weights of the DL model must be binarized, and such binarization may slightly decrease the DL hardware accelerator output accuracy. Nevertheless, there are methods devised to deal with this, which can be adopted to bring significant benefits to garbled DL accelerators in terms of both OT cost and execution time.

6.2 Side-Channel Evaluation

Side-Channel Measurement Setup. HWGN² has been implemented on Artix-7 FPGA device XC7AT100T with package number FTG256. We have captured the power and EM traces (see Appendix C) using Riscure setup, including LeCroy wavePro 725Zi as the setup oscilloscope. We have set our design frequency to 12.5 MHz, the maximum possible clock frequency of Chipwhisperer CW305 target board, and the oscilloscope sampling frequency to 12.5 GHz. For each clock cycle, we have acquired 8100 sample points. Acquiring high-resolution side-channel traces made our design execution time 3.25 s for each classification performed by BM1. For this network, acquiring side-channel traces in the order of millions has high time complexity. Therefore, similar to [12], another MLP architecture, namely, **BM2** is used for traces collection. The changes in MLP architecture hyperparameters allowed us to execute each classification in 31 ms. As HWGN² executes each instruction separately in a sequential manner and the nature of the NNs is repetitive, we argue that the smaller MLP architecture can represent a larger one in terms of leakage.

Leakage Evaluation. We have used a common methodology, namely Test Vector Leakage Assessment (TVLA) test, to evaluate HWGN² leakage resiliency. Although the TVLA test is subject to two disadvantages – false positive/negative results and limited ability to reveal all points of interests [15, 35, 47] – it is still the most common methodology used in recent papers to evaluate the resiliency of the approach against side-channel leakage.

In the TVLA test methodology, Welch’s t-test is used to check the similarity between two trace groups captured from two populations of inputs. Welch’s t-test calculates the t-score as $t = (\mu_1 - \mu_2) / \sqrt{(s_1^2/n_1) + (s_2^2/n_2)}$, where μ_1 and μ_2 are the means, s_1 and s_2 are the standard deviations, and n_1 and n_2 are the total number of the captured traces for first and second population, respectively. Based on the null-hypothesis, if two populations are chosen from one distribution, their corresponding t-score must be less than ± 4.5 . Exceeding t-score magnitude of 4.5 (so-called null-hypothesis) means the design is subject to side-channel leakage with probability greater than 99.99%. In our setup, we choose the non-specific fixed vs. random t-test in a way that our setup, first, captures the power consumption/EM traces from a fixed input computation for all the traces; then, the experiment repeats for a set of randomly generated inputs. Based on the two captured traces, for fixed and random inputs, our setup calculates the t-score based on the aforementioned equation.

Power Side-Channel Leakage Assessment. To illustrate the side-channel protection offered by HWGN², we have mounted the TVLA test on power traces of an unprotected MIPS core, Plasma core presented by Opencores projects [38], and HWGN², with the capacity of one instruction per OT interaction. Figure 5 (a) and (b) shows the TVLA results of an unprotected MIPS core and HWGN², with the capacity of one instruction per OT interaction, respectively. The t-scores are calculated based on 10000 captured traces, 5000 for each fixed and random input population. As one can observe, an unprotected MIPS core t-score has exceeded the ± 4.5 threshold with only 10000 traces, while the HWGN²'s t-score remains below the threshold.

To have a design with leakage resiliency, the t-score results must remain below the threshold with the traces populations in the order of millions [11, 12, 35, 47]. Hence, in the next experiment, we have captured a total of 2 million (2M) traces, 1M traces for each fixed and random input populations. A low t-score, less than ± 4.5 , calculated from a trace population in the order of millions confirms the protection strength of HWGN². It should be noted that these traces are captured in the low-noise setup (i.e., more optimistic for the attacker) while in the actual scenario, the number of traces to break the garbling scheme should be significantly higher due to more noisy environments.

As a proof of concept that HWGN² side-channel resiliency is independent of the function or architecture we also mount the TVLA test on two more implementations: XNOR-based DL hardware accelerator and DL hardware accelerator. Figure 5, (c) and (e), illustrates the t-score of HWGN² applied to XNOR-based BM2, with the capacity of complete set of instructions per OT interaction and one instruction per OT, respectively. As can be seen, the t-scores of HWGN² stay below the threshold of ± 4.5 for different cases of instruction capacity per OT interaction and the function or architecture implemented on an FPGA using HWGN². The t-scores in Fig. 5, (d) and (f), indicate that not only HWGN² with the capacity of one instruction per OT interaction provides a strong protection against power side-channel attacks but also changes in the number of instruction capacity per OT interaction does not affect this protection (for results of EM leakage detection, see Appendix C).

Can we see the Architecture-Related Patterns? Based on the attack presented by Batina et al. [4], revealing the DL model architecture can enhance the attacker's ability to obtain DL model parameters. They showed that the EM trace captured from an unprotected DL model implementation on Atmel ATmega328P microcontroller, which follows the MIPS architecture same as HWGN², with three hidden layers containing 6, 5, and 5 neurons, respectively, has a pattern in which the number of layers and neurons can be revealed. They have used LeCroy WaveRunner 610Zi oscilloscope and RF-U 5-2 near-field EM probe to capture EM traces. To examine if we observe the same patterns as reported in [4], we have implemented the same DL model, **BM3**, and captured 100K EM traces. Figure 6(a) illustrates the captured EM traces from an Atmel ATmega328P microcontroller taken from [4], whereas Fig. 6(b) show the traces collected from our unprotected MIPS evaluator core [38], and Fig. 6(c) presents

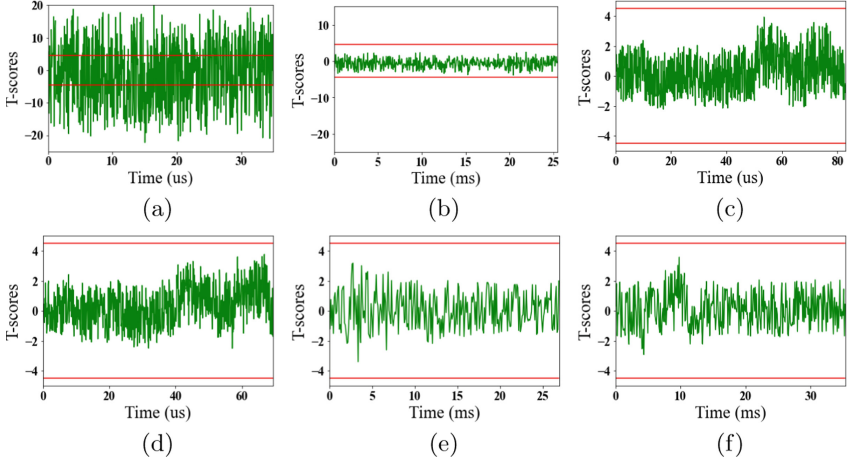


Fig. 5. TVLA test results for implementation of BM2 on (a) an unprotected MIPS core and (b) HWGN² with the capacity of one instruction per OT (calculated for 10K traces) (c) HWGN² applied to XNOR-based BM2 (capacity whole set of instructions per OT) (d) BM2 with the capacity of complete set of instructions per OT interaction, (e) HWGN² applied to XNOR-based BM2 (capacity 1 instruction per OT), and (f) BM2 with the capacity of 1 instruction per OT (calculated for 2M power traces).

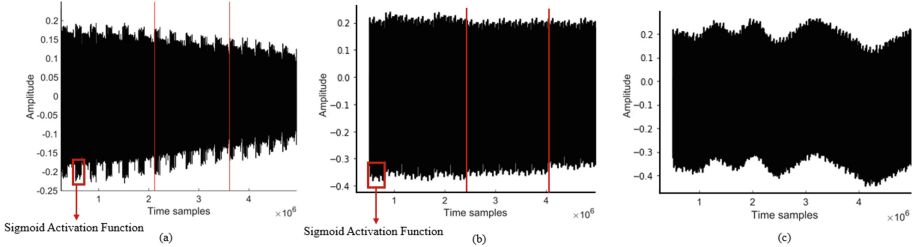


Fig. 6. A randomly chosen EM trace pattern captured from the implementation of BM3 on (a) Atmel ATmega328P microcontroller [4] (b) FPGA with unprotected MIPS evaluator [38] (c) with HWGN². Red lines correspond to time-samples, where the unprotected evaluators start the next layer evaluation.

the captured EM traces of HWGN² for a randomly chosen EM trace. From Fig. 6, it is observable that there exists a pattern, in which the number of the layers and neurons can be seen, similar to the observation made by Batina et al. [4]: the red lines indicate the borders when MIPS evaluator starts the next hidden layer evaluation and the red squares correspond to the EM peak of Sigmoid AF evaluation. In the case of the HWGN², EM traces do not follow a pattern, which could result in revealing the DL model architecture. The reason behind these irregular patterns is that each garbled instruction is encrypted; therefore, in the evaluation phase, the HWGN² treats them as two nonidentical instruc-

tions, although the generated OP corresponding to them is the same. Note that in addition to this observation, we further conduct t-tests, where no EM leakage is detected (see Appendix C).

7 Conclusion

In this paper, we have examined the feasibility of garbling to prevent attackers from launching SCA attacks against DL hardware accelerators. We have implemented HWGN² as a garbled DL hardware accelerator on an Artix-7 FPGA. By tailoring the concepts known only for software garbled DL accelerator [20] to the needs of a hardware DL accelerator, the implementation of such accelerator is enhanced: HWGN² requires up to $62.5\times$ fewer logical and $66\times$ less memory utilization compared to the state-of-art approaches. This is indeed possible at the price of more communication overhead. HWGN² provides users the flexibility to protect their NN IP both in real-time applications and in applications where the hardware resources are limited by hardware resource utilization or communication cost. As our leakage evaluation results indicated, for both EM and power side-channels, the t-scores are below the threshold (± 4.5), which shows the side-channel leakage resiliency of HWGN² with trace population in the order of millions. Another strength of HWGN² is the DL model architecture thanks to the SFE/PFE protocol realized through MIPS instructions.

Acknowledgements. This work was supported partially by Semiconductor Research Corporation (SRC) under Task IDs 2991.001 and 2992.001.

Appendix A. Summary of Relevant Studies

This appendix covers recent attacks mounted against NNs as well as the similarities and differences between HWGN² and garbled DL accelerators proposed to offer *security of users' data* in Tables 5 and 6.

Appendix B. TinyGarble-Based Implementation of HWGN²

TinyGarble [44] is a garbling framework that supports Yao's protocol and uses hardware-synthesis tools to generate circuits for secure computation automatically. The main advantage of TinyGarble is the scalability enabled by exploiting a sequential circuit description for garbled circuits and garbling optimization techniques such as Free-XOR [26], Row Reduction [36], and Garbling with a Fixed-key Block Cipher [5]. Figure 7a illustrates the flow of HWGN² following TinyGarble [44] approach. At first, garbler chooses input encryption labels (e) (Step 1.1) and constructs the GC of function f by generating garbled tables (GT) of all gates, garbled labels (L) of all wires, and a custom circuit description (SCD) file (Step 1.2), which is the mapping between the GC and function f .

Table 5. Summary of most recent side-channel attacks against DL accelerators.

Paper	Targets	Side-channel modality	Attack scenario	Implementation platform
Xiang et al. [52]	DL Model Architecture	Power	<ul style="list-style-type: none"> Modeling the power consumption of different DL hardware accelerator components based on the number of additions and multiplications Trained a classifier to reveal the DL Model architecture based on the captured power consumption traces 	Raspberry Pi
DeepEM [55]	DL Model Architecture	EM	<ul style="list-style-type: none"> Presumption of a layer computations Finding the number of parameters through each layer based on EM traces 	Pynq-Z1
CSI NN [4]	DL Model Architecture +Weights +AF	Timing + EM	<ul style="list-style-type: none"> Modeling all possible AF timing side-channel Extracting the AF used in the DL Model Architecture Distinguishing the EM patterns to find the number of layers and neurons Launching CPA to reveal the weights 	ARM Cortex-M3 + Atmel ATmega328P
Dubey et al. [14]	DL Model Weights	Power	<ul style="list-style-type: none"> Capturing the power consumption traces from changing status of pipeline registers Launching a CPA based attack to reveal weights 	SAKURA-X FPGA board
Yoshida et al. [54]	DL Model Weights	Power	<ul style="list-style-type: none"> Launching a CPA based attack to reveal weights 	Xilinx Spartan3-A

Table 6. Summary of garbled DL accelerators and their features.

Paper	Adversary model	Approach	Contribution	Implementation platform
DeepSecure [42]	HbC	Garbling	<ul style="list-style-type: none"> Presentation of pre-processing approach pre-processing step would reveal some information about the network parameters and structure of data cf. [39] 	Intel Core i7 CPUs
Chameleon [40]	HbC	Hybrid	<ul style="list-style-type: none"> Performs linear operations using additive secret sharing and nonlinear operations using Yao's Garbled Circuits 	8-Core AMD CPU 3.7GHz
Ball et al. [2]	HbC	Hybrid	<ul style="list-style-type: none"> Improvement of the BMR scheme [3] to support Non-linear operations 	Intel Core i7-4790 CPUs
XONN [39]	HbC	Garbling	<ul style="list-style-type: none"> Support Binary NNs Conversion of Matrix Multiplication to XNOR PopCount 	Intel Xeon CPU E5-2650
TinyGarble2 [20]	HbC + Malicious	Garbling	<ul style="list-style-type: none"> Provision of protection against malicious adversary Alleviation garbling memory cost 	Intel Xeon CPU E5-2650
GarbledCPU [46]	HbC	Garbling	<ul style="list-style-type: none"> Presentation of FPGA accelerator for GC evaluation 	Virtex-7 FPGA
RedCrypt [41]	HbC	Garbling	<ul style="list-style-type: none"> Minimizing the hardware architecture idle cycles to achieve scalable garbling 	Virtex UltraSCALE VCU108

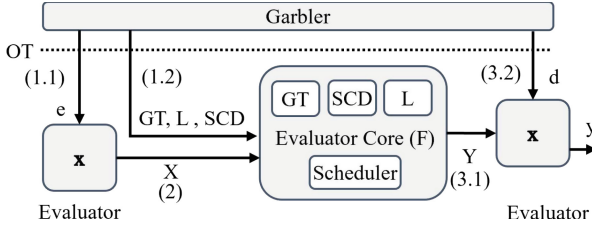


Fig. 7. TinyGarble-based implementation [44] of HWGN² (L : wires garbled labels, GT : garbled tables, e : encryption labels, d : decryption labels, x : evaluator’s raw input, X : evaluator’s garbled input, Y : garbled output, Y_i, X_i, GT_i, L_i : garbled input, output, garbled tables, wire labels corresponding to i^{th} sub-netlist, respectively, y : evaluator’s raw output, and SCD : A custom circuit description which allows TinyGarble to evaluate the Boolean circuit).

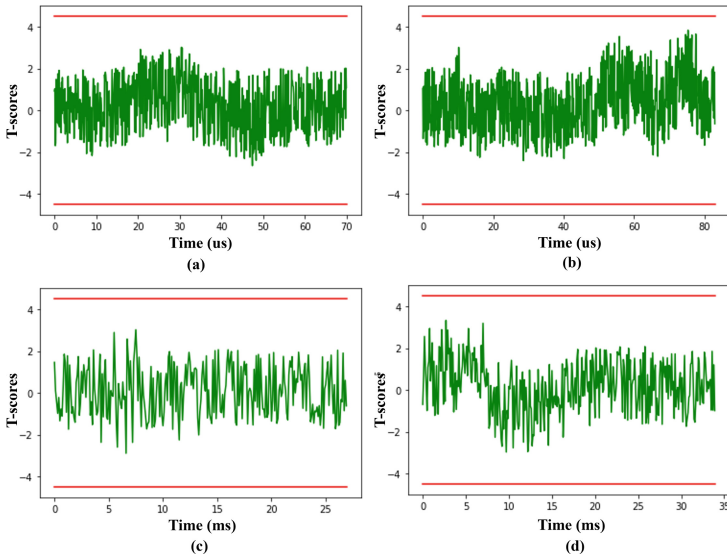


Fig. 8. TVLA test results (a) HWGN² applied to XNOR-based BM2 (capacity whole set of instructions per OT) (b) BM2 with the capacity of complete set of instructions per OT interaction, (c) HWGN² applied to XNOR-based BM2 (capacity 1 instruction per OT), and (d) BM2 with the capacity of 1 instruction per OT (calculated for $2M$ EM traces).

GT, L, SCD, e are sent through one OT interaction to the evaluator for further garbling protocol process. e is then used by the evaluator to generate garbled input X from the evaluator’s input x (Step 2). Afterward in Step 3.1, GT, L , and SCD are used by the evaluator to evaluate the GC based on the given X sequentially using the scheduler module (cf. [44] for more information). In the final step (Step 3.2), output decryption labels (d) are sent to the evaluator to decrypt the

evaluator core's garbled output Y and obtain its raw output y . The sequential evaluation supported by TinyGarble provides the GC protocol the scalability of evaluation of larger netlists. However, when one implements the DL hardware accelerator in the garbled format which has a large netlist, memory and logical resource utilization become burdens for DL hardware accelerators [21] (see Sect. 6.1).

Appendix C. TVLA Test Evaluation of EM Side-Channel

One of the first studies that has compared the capabilities of attackers launching power vs. EM SCA is [37], where it is suggested that the EM leakage can provide more information than the power consumption of the same chip cf. [48]. This has been further justified in [48] through the evaluation of the information theoretic and security metrics [49]. Therefore, it might be thought that the EM side-channel could offer some information about the secret, i.e., the weights of the garbled NN. To collect the EM traces, it has been already verified that measurements from the frontside of a chip can offer a high signal-to-noise ratio [19]; hence, we stick to this setting to perform measurements. Our setup described in Sect. 6.2 is equipped with HP EM probe 125 (SN126 0.2 mm). Figure 8 shows the t-scores computed for HWGN² applied against BM2. As shown in Fig. 8, the t-scores of EM traces are below the threshold (± 4.5) which is the proof of the EM leakage resiliency of HWGN².

References

1. Ateniese, G., Mancini, L.V., Spognardi, A., Villani, A., Vitali, D., Felici, G.: Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *Int. J. Secur. Netw.* **10**(3), 137–150 (2015)
2. Ball, M., Carmer, B., Malkin, T., Rosulek, M., Schimanski, N.: Garbled neural networks are practical. *Cryptology ePrint Archive* (2019)
3. Ball, M., Malkin, T., Rosulek, M.: Garbling gadgets for boolean and arithmetic circuits. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 565–577 (2016)
4. Batina, L., Bhasin, S., Jap, D., Picek, S.: CSI NN: reverse engineering of neural network architectures through electromagnetic side channel. In: *28th USENIX Security Symposium (USENIX Security 2019)*, pp. 515–532 (2019)
5. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: *2013 IEEE Symposium on Security and Privacy*, pp. 478–492. *IEEE* (2013)
6. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pp. 784–796 (2012)
7. Breier, J., Jap, D., Hou, X., Bhasin, S., Liu, Y.: SNIFF: reverse engineering of neural networks with fault attacks. *IEEE Trans. Reliab.* (2021)
8. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) *CHES 2004*. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2

9. Chakraborti, A., et al.: Cloud computing security: foundations and research directions. *Found. Trends® Privacy Secur.* **3**(2), 103–213 (2022)
10. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) *CHES 2002*. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
11. De Cnudde, T., Ender, M., Moradi, A.: Hardware masking, revisited. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 123–148 (2018)
12. Dubey, A., Ahmad, A., Pasha, M.A., Cammarota, R., Aysu, A.: Modulonet: neural networks meet modular arithmetic for efficient hardware masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 506–556 (2022)
13. Dubey, A., Cammarota, R., Aysu, A.: Bomanet: boolean masking of an entire neural network. In: *2020 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE (2020)
14. Dubey, A., Cammarota, R., Aysu, A.: Maskednet: the first hardware inference engine aiming power side-channel protection. In: *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 197–208. IEEE (2020)
15. Durvaux, F., Standaert, F.-X.: From improved leakage detection to the detection of points of interests in leakage traces. In: Fischlin, M., Coron, J.-S. (eds.) *EUROCRYPT 2016*. LNCS, vol. 9665, pp. 240–262. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_10
16. Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., Ristenpart, T.: Privacy in pharmacogenetics: an {End-to-End} case study of personalized warfarin dosing. In: *23rd USENIX Security Symposium (USENIX Security 2014)*, pp. 17–32 (2014)
17. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: applying neural networks to encrypted data with high throughput and accuracy. In: *International Conference on Machine Learning*, pp. 201–210. PMLR (2016)
18. Herken, R.: *The Universal Turing Machine: A Half-Century Survey*. Springer, Heidelberg (1988)
19. Heyszl, J., Merli, D., Heinz, B., De Santis, F., Sigl, G.: Strengths and limitations of high-resolution electromagnetic field measurements for side-channel analysis. In: Mangard, S. (ed.) *CARDIS 2012*. LNCS, vol. 7771, pp. 248–262. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37288-9_17
20. Hussain, S., Li, B., Koushanfar, F., Cammarota, R.: TinyGarble2: smart, efficient, and scalable Yao’s garble circuit. In: *Proceedings of the 2020 WKSP on Privacy-Preserving Machine Learning in Practice*, pp. 65–67 (2020)
21. Järvinen, K., Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: Garbled circuits for leakage-resilience: hardware implementation and evaluation of one-time programs. In: Mangard, S., Standaert, F.-X. (eds.) *CHES 2010*. LNCS, vol. 6225, pp. 383–397. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_26
22. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: {GAZELLE}: a low latency framework for secure neural network inference. In: *27th USENIX Security Symposium (USENIX Security 2018)*, pp. 1651–1669 (2018)
23. Kane, G.: *MIPS RISC Architecture*. Prentice-Hall Inc. (1988)
24. Kilian, J.: Founding cryptography on oblivious transfer. In: *Proceedings of the Annual ACM Symposium on Theory of Computing*, pp. 20–31 (1988)
25. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) *CRYPTO 1999*. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25

26. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
27. LeCun, Y.: 1.1 deep learning hardware: past, present, and future. In: 2019 IEEE International Solid-State Circuits Conference-(ISSCC), pp. 12–19. IEEE (2019)
28. Levi, I., Hazay, C.: Garbled-circuits from an SCA perspective: free XOR can be quite expensive. Cryptology ePrint Archive (2022)
29. Lindell, Y.: Fast cut-and-choose-based protocols for malicious and covert adversaries. *J. Cryptol.* **29**(2), 456–490 (2016)
30. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_4
31. Lindell, Y., Pinkas, B., Smart, N.P., Yanai, A.: Efficient constant-round multi-party computation combining BMR and SPDZ. *J. Cryptol.* **32**(3), 1026–1069 (2019)
32. Liu, Y., Dachman-Soled, D., Srivastava, A.: Mitigating reverse engineering attacks on deep neural networks. In: 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 657–662. IEEE (2019)
33. Mantel, H., Scheidel, L., Schneider, T., Weber, A., Weinert, C., Weißmantel, T.: RiCaSi: rigorous cache side channel mitigation via selective circuit compilation. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 2020. LNCS, vol. 12579, pp. 505–525. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65411-5_25
34. Mittal, S., Gupta, H., Srivastava, S.: A survey on hardware security of DNN models and accelerators. *J. Syst. Archit.* **117**, 102163 (2021)
35. Moradi, A., Richter, B., Schneider, T., Standaert, F.X.: Leakage detection with the x2-test. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 209–237 (2018)
36. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce, pp. 129–139 (1999)
37. Peeters, E., Standaert, F.X., Quisquater, J.J.: Power and electromagnetic analysis: improved model, consequences and comparisons. *Integration* **40**(1), 52–60 (2007)
38. Rhoads, S.: Plasma - most MIPS I(TM) opcodes (2001). <https://opencores.org/projects/plasma>. Accessed 9 Mar 2022
39. Riazi, M.S., Samragh, M., Chen, H., Laine, K., Lauter, K., Koushanfar, F.: {XONN}:{XNOR-based} oblivious deep neural network inference. In: 28th USENIX Security Symposium (USENIX Security 2019), pp. 1501–1518 (2019)
40. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: a hybrid secure computation framework for machine learning applications. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, pp. 707–721 (2018)
41. Rouhani, B.D., Hussain, S.U., Lauter, K., Koushanfar, F.: ReDCrypt: real-time privacy-preserving deep learning inference in clouds using FPGAs. *ACM Trans. Reconfigurable Technol. Syst. (TRETTS)* **11**(3), 1–21 (2018)
42. Rouhani, B.D., Riazi, M.S., Koushanfar, F.: Deepsecure: scalable provably-secure deep learning. In: Proceedings of the 55th Annual Design Automation Conference, pp. 1–6 (2018)
43. Schellenberg, F., Gnad, D.R., Moradi, A., Tahoori, M.B.: An inside job: remote power analysis attacks on FPGAs. In: 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1111–1116. IEEE (2018)

44. Songhori, E.M., Hussain, S.U., Sadeghi, A.R., Schneider, T., Koushanfar, F.: Tinygarble: highly compressed and scalable sequential garbled circuits. In: 2015 IEEE Symposium on Security and Privacy, pp. 411–428. IEEE (2015)
45. Songhori, E.M., Riazi, M.S., Hussain, S.U., Sadeghi, A.R., Koushanfar, F.: ARM2GC: succinct garbled processor for secure computation. In: Proceedings of the 56th Annual Design Automation Conference 2019, pp. 1–6 (2019)
46. Songhori, E.M., Schneider, T., Zeitouni, S., Sadeghi, A.R., Dessouky, G., Koushanfar, F.: GarbledCPU: a MIPS processor for secure computation in hardware. In: 2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE (2016)
47. Standaert, F.-X.: How (not) to use Welch’s T-test in side-channel security evaluations. In: Bilgin, B., Fischer, J.-B. (eds.) CARDIS 2018. LNCS, vol. 11389, pp. 65–79. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-15462-2_5
48. Standaert, F.-X., Archambeau, C.: Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 411–425. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_26
49. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_26
50. Wang, X., Gordon, S.D., McIntosh, A., Katz, J.: Secure computation of MIPS machine code. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 99–117. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_6
51. Wang, X., Malozemoff, A.J., Katz, J.: Faster secure two-party computation in the single-execution setting. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10212, pp. 399–424. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_14
52. Xiang, Y., et al.: Open DNN box by power side-channel attack. *IEEE Trans. Circuits Syst. II: Express Br.* **67**(11), 2717–2721 (2020)
53. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (SFCS 1986), pp. 162–167. IEEE (1986)
54. Yoshida, K., Kubota, T., Okura, S., Shiozaki, M., Fujino, T.: Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5. IEEE (2020)
55. Yu, H., Ma, H., Yang, K., Zhao, Y., Jin, Y.: DeepEM: deep neural networks model recovery through EM side-channel information leakage. In: 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 209–218. IEEE (2020)
56. Zhao, M., Suh, G.E.: FPGA-based remote power side-channel attacks. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 229–244. IEEE (2018)