Securing Binarized Neural Networks via PUF-Based Key Management in Memristive Crossbar Arrays

Gokulnath Rajendran, Debajit Basak, Suman Deb, and Anupam Chattopadhyay, *Senior Member, IEEE*

***Abstract*—Binarized neural networks (BNNs) are a subset of deep neural networks proposed to consume less computational resources with a smaller energy budget. Recent studies showed that memristor-based in-memory computing architectures can be constructed to accelerate BNNs, with better performance compared to traditional CMOS technologies. The memristor nonvolatility utilized for in-memory computing poses a notable threat to theft attacks in the presence of adversaries with physical access. This motivates us to introduce two novel protection methodologies to safeguard the model parameters of BNNs in the memristive crossbar. We propose to take advantage of physical unclonable functions (PUFs), which can be implemented using memristor-based crossbars for protecting BNN. This feature provides superior security compared to the traditional stored-key-based schemes. We provide circuit-level hardware designs to implement our methodologies with negligible additional overhead compared to an unprotected design and detailed supporting analysis to validate our security claims.**

***Index Terms*—Binary neural networks, crossbar, memristor, physical unclonable function (PUF), security, weights.**

I. Introduction

DEEP neural networks (DNNs) are used in many applications today and have tremendous business value. The data used for training often concerns privacy, and training a high-accuracy model requires considerable resources. Hence, the DNN model is considered intellectual property and a valuable asset to protect from theft. The deployment of the trained model to the dedicated inference hardware in the edge can be handled well through the standard key exchange and encryption schemes. Hence, man-in-the-middle attacks can be prevented. However, the vulnerability of retrieving model parameters from the inference hardware still poses a considerable threat.

Binarized neural networks (BNNs) are an emerging class of neural networks in which weights and activations are constrained to take values $+1$ and -1. BNN is attractive for edge applications because its binary computation during inference reduces computational complexities and saves energy. Implementing BNN with emerging nonvolatile memory (NVM) devices, especially resistive random access memory (RRAM), is one of the primary research areas, and several methodologies and optimizations have been proposed to achieve better inference performance [1], [2]. Accelerating DNN with RRAM-based in-memory computing is challenging because RRAM nonidealities hinder multibit computing. However, for BNN, only two RRAM states are required and are more robust to variability. Implementing BNN in RRAM crossbars has the advantage that a current comparator can replace the resource-hungry analog-to-digital converter [1], [2]. Hence, the BNN crossbar arrays also necessitate significantly fewer peripheral resources. Consequently, RRAM-based BNN accelerators could be preferred over their multibit DNN counterparts for edge implementations.

Other than for NN accelerators, emerging NVM devices are also studied for hardware security primitives, such as physical unclonable function (PUF) and true random number generator (TRNG) [3]. A PUF works by producing a response when a challenge is given to it. The response can be either used as a key or to derive new key(s). A single PUF device can take many input challenges and produce corresponding responses, making it flexible to tailor for applications after deployment. PUFs are more secure than stored keys that are prone to theft by probing. Unlike the stored-key paradigm, PUFs can also be reconfigured after deployment. After reconfiguration, the PUF will produce a different response for the same applied challenge. Hence, if the keys are compromised, PUFs can still be used after reconfiguration [3].

Related Works: To avoid communication overhead, the BNN accelerators are often deployed in edge devices, which makes those physically accessible to the attackers. There are few previous works on securing multibit DNN in RRAM crossbars against theft attacks but not on BNN. They essentially transform the weight matrix before mapping it into the crossbar using a stored key. The first technique proposed is shuffling, in which the weight matrix is shuffled along the rows, columns, or both and the shuffled weights are mapped into the crossbar [4], [5], [6]. The shuffling details, such as the actual position of the weights, are used as the key. This method has several drawbacks: 1) it is not PUF-friendly because the weight matrix indices are used as keys. Since the PUF output is a binary response, it cannot be directly utilized as a key for this method. Also, the index-based key is very long, which is unnecessary even if it gives factorial security strength in brute-force attacks and 2) the peripheral overhead is very high, penalizing runtime, area, and energy. The second technique proposed with regular DNN is the weight matrix transformation by 1's complement [7] along the column. This scheme is attractive compared to the shuffling technique due to its low resource overhead. However, this technique is unexplored for BNNs and optimized architectures, such as integrating the batch normalization (BN) layer with

Manuscript received 20 April 2024; revised 7 June 2024; accepted 24 June 2024. Date of publication 2 July 2024; date of current version 13 February 2025. This work was supported by the National Research Foundation (NRF) of Singapore for the NRF-CRP under Grant NRF-CRP21-2018-003. *(Corresponding author: Gokulnath Rajendran.)*

Gokulnath Rajendran, Suman Deb, and Anupam Chattopadhyay are with the College of Computing and Data Science, Nanyang Technological University, Singapore 639798 (e-mail: gokulnat002@e.ntu.edu.sg; sumandeb@ntu.edu.sg; anupam@ntu.edu.sg).

Debajit Basak is with Atom Semiconductor, Hong Kong, China (e-mail: debajit.basak@atom-semiconductor.com).

Digital Object Identifier 10.1109/LES.2024.3422294

the matrix-vector multiplication (MVM) layer [2]. The BN-integrated architectures of BNNs differ significantly from standard DNN architectures. Hence, schemes proposed for standard DNN cannot be directly extended for BNNs. The third technique proposed is to pad the actual weight matrix with extra weights [6]. This technique has no quantifiable security benefits except in hiding the network topology.

The main contributions of this letter are as follows. *First*, we propose two novel PUF-based weight transformation techniques for BNNs deployed in memristive crossbars: 1) inversion and 2) swapping. The row inversion and swapping techniques are discussed for the first time, even from the context of RRAM-based multibit DNN with stored keys. *Second*, we present hardware implementations of each methodology with detailed discussions on the inference performance. We also conducted power analyses to estimate the additional overhead incurred.

II. BNN in RRAM Crossbars

The trained binarized weights $W^b_{j,k}$ of the BNN are directly mapped to the RRAM crossbar as resistance states of the memristors. The standard mapping uses two RRAM cells to represent a single binary state arranged along the columns. The binary weight value $+1$ is implemented by programming the top cell to LRS and the bottom cell to HRS. For -1, it is implemented oppositely. It is equivalent to specifying $+1$ as (1, 0) and -1 as (0, 1). The binary inputs also follow the same convention for $+1$ and -1. The output of the MVM operation y_k is given as input to the activation unit, which is the sign function of the BNN. One of the principal strategies for implementing BNN is to integrate the BN and the MVM layers in a single crossbar and use the BN layer as a threshold B_k in calculating the neuron output y^b_k with a comparator [2]. It is achieved by rewriting the BN layer parameters through the sign function as given in (1) and (2). Here, k refers to the column index, and β, σ, ϵ, γ, and μ are the parameters of the BN layer

$$B_k = -\beta\frac{\sqrt{\sigma^2+\epsilon}}{\gamma} + \mu \tag{1}$$

$$y^b_k = \begin{cases} -1, & \text{if } y_k < B_k \\ 1, & \text{otherwise.} \end{cases} \tag{2}$$

The B_k can take decimal values. Hence, it cannot be mapped to the RRAM crossbar directly. One of the notable properties of BNN is that if the weight matrix is of even number size, then its corresponding MVM output y_k will always be an even number. Hence, utilizing this property, the B_k can be modified to $2\lceil(B_k/2)\rceil$ and mapped to the RRAM devices in the crossbar. There are two ways to implement it. The first way is to have two separate columns for MVM and BN and compare their output currents using the comparator to generate the output bit y^b_k. We refer to this architecture as a double-column BN in this letter. The second way is to integrate the BN and MVM in a single column and compare it to a reference value -1. The B_k is sign inverted before integrating into a single column. We refer to this architecture as a single-column BN.

Attacker Model: The attacker can access the edge hardware with an RRAM crossbar, in which the trained model has been deployed for execution. The attacker knows the topology of the deployed BNN. The attacker can read the conductance states of RRAM devices using micro-probing, which has been identified as a viable method [4], [5], [6], [7], [8], [9]. The attacker can also employ other physical attacks, such as elemental analysis using energy dispersive x-ray spectroscopy and imaging, as described in earlier studies with experimental validation, to find the conductance state of RRAM devices [9], [10]. However, the PUF and its response in the edge device are always hidden and protected from the attacker at any point in time. Hence, the attacker cannot access PUF and collect its responses making machine learning attacks not applicable. Here, it should be noted that the responses are never stored as memory states with PUF. The PUF always computes them during runtime. Our goal is to prevent the attacker from correctly reading the BNN model parameters from the RRAM crossbar.

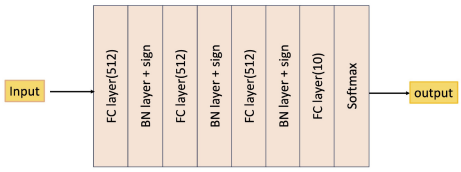

Fig. 1. Architecture of neural network trained used for MNIST dataset.

III. Proposed Security Schemes

For the BNN inference, the trained weight-matrix $W^b_{j,k}$ of each layer is directly mapped to the RRAM crossbar. We propose transforming the $W^b_{j,k}$ into a new matrix $W^{*b}_{j,k}$ with a PUF response and then storing it in the crossbar. Hence, when $W^{*b}_{j,k}$ is used for inference without the PUF response used for transformation, the classification will be incorrect, and the model will be useless. We previously proposed a reconfigurable Parity RRAM PUF with a 1T1R crossbar and a procedure to extend it to a random number generator [11]. We used a similar PUF construction for analysis in this letter. We propose two methodologies with PUF to protect the trained BNN parameters of each layer. We analyse our schemes using the BNN architecture shown in Fig. 1 and train it on the MNIST dataset. In this study, we used the RRAM model discussed in [12] for the simulations. We need to determine the level of difficulty an attacker would face when attempting to recover $W^b_{j,k}$ from the $W^{*b}_{j,k}$—after gathering $W^{*b}_{j,k}$ from the RRAM crossbar. We estimate this security strength for our proposed methodologies in terms of the brute-force approach and the effectiveness of weight matrix transformation in inference accuracy following similar state-of-the-art works [4], [5], [6], [7].

A. Transformation by Inversion

We propose two inversion strategies for $W^b_{j,k}$—one is along the row j, which we refer to as the row inversion, and the other is along the column k, which we refer to as a column inversion. The length of the PUF response required for row and column inversion equals the row size, l_j and column size, l_k of $W^b_{j,k}$, respectively. The security strength of the $W^{*b}_{j,k}$ against brute force attack depends on the length of the response. For a response size of length l, it is 2^l.

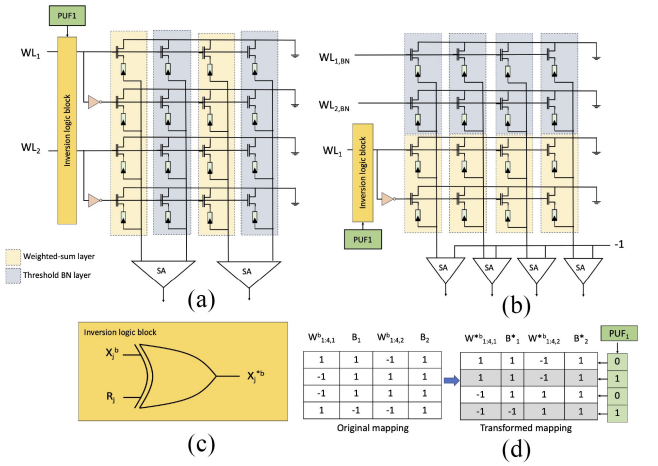

Fig. 2. Layout of PUF-protected row inversion of (a) double-column and (b) single-column BN architectures, (c) inversion logic to recover the inference output and (d) illustration of transformation, including BN layer.

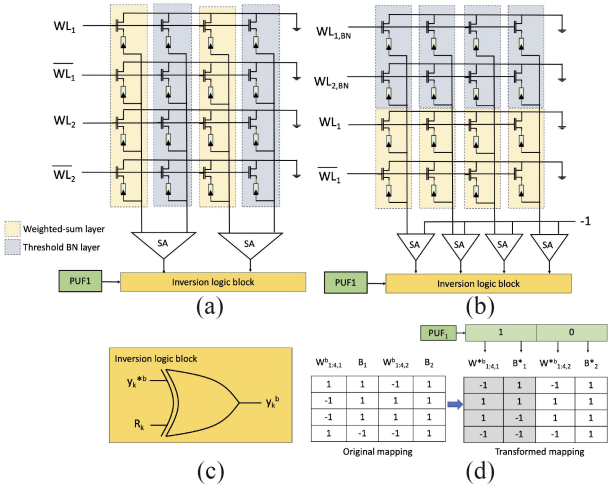

Fig. 3. Layout of PUF-protected column inversion of (a) double-column and (b) single-column BN architectures, (c) inversion logic to recover the inference output and (d) illustration of transformation, including BN layer.

1) Row Inversion: This transformation can be completed using (3), where R_j and $W^b_{j,1:l_k}$ refer to the response bit value and weight matrix elements at row index j, respectively. Only the transformed matrix $W^{*b}_{j,k}$ is mapped to the crossbar. During inference runtime, the same PUF response used for transformation is used to transform input signals X^b_j applied to the crossbar as given in (4). Both double-column and single-column BN architectures are implemented with the same transformation flow as detailed in Fig. 2. The inversion logic required at the input is the XOR gate that takes X^b_j and R_j as inputs and outputs the transformed input signal X^{*b}_j to the crossbar, as detailed in Fig. 2

$$W^{*b}_{j,1:l_k} = \begin{cases} -W^b_{j,1:l_k}, & \text{if } R_j = 1 \\ W^b_{j,1:l_k}, & \text{otherwise} \end{cases} \tag{3}$$

$$X^{*b}_j = \begin{cases} -X^b_j, & \text{if } R_j = 1 \\ X^b_j, & \text{otherwise.} \end{cases} \tag{4}$$

2) Column Inversion: The response bit value R_k of PUF is directly coupled to the column k of the weight matrix. In row-inversion, we transformed only the weight matrix $W^b_{j,k}$. However, both $W^b_{j,k}$ and B_k are transformed in column inversion. For double-column BN architectures, $W^b_{j,k}$ and B_k are transformed using the mappings in (5) and (6), respectively. Similarly, the mappings in (5) and (7) transform $W^b_{j,k}$ and B_k of single-column BN architectures, respectively. The comparator output y^{*b}_k is checked for sign and inverted during inference runtime with the PUF response as given in (8) for both the architectures before passing it to the next inference stage. The required inversion logic is the XOR gate that takes the y^{*b}_k and R_k as inputs and gives y^b_k as output, as detailed in Fig. 3(c)

$$W^{*b}_{1:l_j,k} = \begin{cases} -W^b_{1:l_j,k}, & \text{if } R_k = 1 \\ W^b_{1:l_j,k}, & \text{otherwise} \end{cases} \tag{5}$$

$$B^*_k = \begin{cases} -B_k + 2, & \text{if } R_k = 1 \\ B_k, & \text{otherwise} \end{cases} \tag{6}$$

$$B^*_k = \begin{cases} B_k - 2, & \text{if } R_k = 1 \\ B_k, & \text{otherwise} \end{cases} \tag{7}$$

$$y^b_k = \begin{cases} -y^{*b}_k, & \text{if } R_k = 1 \\ y^{*b}_k, & \text{otherwise.} \end{cases} \tag{8}$$

Different PUF responses are used to transform each layer. Hence, recovering the transferred layer without the actual response is tricky, improving security. We used 10 different PUF responses for each layer to analyze the impact of weight transformation and calculated the average inference accuracy. In consideration of our simulation time constraints, we have empirically chosen the value of 10 to ensure a realistic analysis by utilizing multiple secret keys rather than relying solely on a single key. Tables I and II show the classification accuracy loss with the MNIST data set when individual layers are transformed and not recovered with PUF response during inference. We have also presented the response key length required for each transformation case. The original accuracy of the model without transformation is 96.74%. When all the layers are transformed using the inversion scheme, the accuracy is reduced to 11.13% and 10.2% for row and column inversion, respectively. The impact of the transformation of the individual layers is also presented in Tables I and II. We performed a power analysis for the overhead incurred because

TABLE I
INFERENCE ACCURACY WITH ROW INVERSION

Transformed layers	PUF key length	Inference accuracy without right PUF key
None (without PUF protection)	0	96.74%
FC 1	784	9.83%
FC 2	512	8.28%
FC 3	512	13.49%
FC 1,2,3	$784 + 2 \times 512$	11.13%

TABLE II
INFERENCE ACCURACY WITH COLUMN INVERSION

Transformed layers	PUF key length	Inference accuracy without right PUF key
None (without PUF protection)	0	96.74%
FC +BN 1	512	10.71%
FC +BN 2	512	7.35%
FC +BN 3	512	13.8%
FC + BN 1,2,3	3×512	10.2%

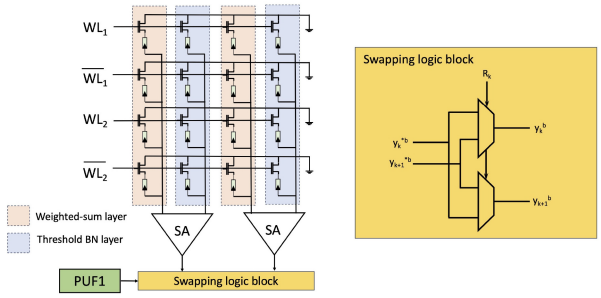

Fig. 4. Layout of PUF-protected swapping scheme with its logic block.

TABLE III
INFERENCE ACCURACY WITH SWAPPED COLUMNS

Transformed layers	PUF key length	Inference accuracy without right PUF key
None (without PUF protection)	0	96.74%
FC +BN 1	256	79.79%
FC +BN 2	256	88.4%
FC +BN 3	256	94.57%
FC + BN 1,2,3	3×256	52.96%

of the XOR logic. We found that the percentage increase in power is negligible and significantly less than 1%.

B. Transformation by Swapping

A trained BNN model is valid only when all the individual weight values are used for inference in trained order. We propose transforming the weight matrix by locally swapping the values along the rows or columns based on the PUF response. We discuss transforming with column swapping here; the same procedure applies to row swapping. To accomplish swapping with BN-integrated architectures, we need to swap the column values of the weight matrix together with the BN layer values as given in (9) and (10). The new $W^{*b}_{j,k}$ and B^*_k are mapped to the crossbar. The layout of such implementation is given in Fig. 4. Based on the PUF response R_k, the output y^{*b}_k is swapped during inference runtime. The required swapping logic block is also given in Fig. 4. Similar to the previous section, we analysed the impact of swapping transformation on inference accuracy with PUF responses. With this method, transformation on individual layers alone does not significantly affect the overall classification accuracy. However, when all the layers are transformed, the accuracy drops to 52.96%. The key point is that this accuracy does not reflect the security strength, and the PUF response still determines the security. The accuracy drops only show the performance without the key. Three PUF keys of 256-bit length are required in this transformation, which is half the column size of the weight matrix. Hence, the security against brute force attack is 2^{256} for the individual layer, with the total security being $2^{3\times256}$. Similar to the inversion scheme, we estimated power for the overhead incurred by the MUX. We found that the percentage increase in power is insignificant and well below 1%

$$R_k = \begin{cases} 1, \text{ then } W^{*b}_{j,k} = W^b_{j,k+1} \text{ and } W^{*b}_{j,k+1} = W^b_{j,k} \\ 0, \text{ then } W^{*b}_{j,k} = W^b_{j,k} \text{ and } W^{*b}_{j,k+1} = W^b_{j,k+1} \end{cases} \quad (9)$$

$$R_k = \begin{cases} 1, \text{ then } B^*_k = B_{k+1} \text{ and } B^*_{k+1} = B_k \\ 0, \text{ then } B^*_k = B_k \text{ and } B^*_{k+1} = B_{k+1}. \end{cases} \quad (10)$$

IV. CONCLUSION

We presented two methods with their hardware design and analysis to protect BNN in RRAM crossbars using PUF. The security strength of the stolen transformed weight matrix $W^{*b}_{j,k}$ against brute force attacks for both schemes is directly related to the length of the response key. The response key length required in the swapping scheme is half that required in the inversion scheme for the same weight matrix. Hence, the inversion scheme is more secure in terms of key length. However, this difference in key length in the two schemes may not be significant if we transform the larger matrix. For example, the key length of (3×256) discussed in this letter for column swapping can provide increased security against brute force attacks. Hence, the swapping scheme is also robust. We conclude that the choice of the scheme is based on the required security strength and the availability of the PUF response length. In situations where the secret key generated through PUF is not of adequate length for transforming the weight matrix by the inversion scheme, the designer may choose to use the swapping scheme. Furthermore, to achieve a better tradeoff between security strength, the inversion scheme may be preferred to transform $W^b_{j,k}$ with a smaller dimension, and a swapping scheme may be preferred to transform $W^b_{j,k}$ with a larger dimension. The proposed row inversion and swapping techniques can also be extended to RRAM-based multibit DNN implementations, which can be future work.

REFERENCES

[1] X. Sun et al., "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," in *Proc. Design, Autom. Test Europe Conf. Exhibit. (DATE)*, 2018, pp. 1423–1428.
[2] H. Kim, Y. Kim, and J.-J. Kim, "In-memory batch-normalization for resistive memory based binary neural network hardware," in *Proc. 24th Asia South Pac. Design Autom. Conf.*, 2019, pp. 645–650.
[3] G. Rajendran et al., "Application of resistive random access memory in hardware security: A review," *Adv. Electron. Mater.*, vol. 7, no. 12, 2021, Art. no. 2100536.
[4] M. E. Galicia et al., "'S3cure': Scramble, shuffle and shambles-secure deployment of weight matrices in memristor crossbar arrays," in *Proc. Int. Conf. Neuromorph. Syst.*, 2023, pp. 1–8.
[5] M. Zou et al., "Security enhancement for RRAM computing system through obfuscating crossbar row connections," in *Proc. DATE*, 2020, pp. 466–471.
[6] Y. Wang et al., "A low cost weight obfuscation scheme for security enhancement of ReRAM based neural network accelerators," in *Proc. Asia South Pac. Design Autom. Conf.*, 2021, pp. 499–504.
[7] M. Zou et al., "Enhancing security of memristor computing system through secure weight mapping," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, 2022, pp. 182–187.
[8] M. Zou, N. Du, and S. Kvatinsky, "Review of security techniques for memristor computing systems," *Front. Electron. Materials*, vol. 2, Dec. 2022, Art. no. 1010613.
[9] C. Chavda et al., "Vulnerability analysis of on-chip access-control memory," in *Proc. 9th USENIX Workshop Hot Topics Storage File Syst. (HotStorage 17)*, 2017, pp. 1–6.
[10] N. Huynh et al., "Hardware security of emerging non-volatile memory devices under imaging attacks," in *Proc. Int. Conf. Appl. Electron. (AE)*, 2021, pp. 1–4.
[11] G. Rajendran et al., "Harnessing entropy: RRAM crossbar-based unified PUF and RNG," in *Proc. Int. Conf. VLSI Design (VLSID)*, 2024, pp. 560–564.
[12] C. Bengel et al., "Variability-aware modeling of filamentary oxide-based bipolar resistive switching cells using SPICE level compact models," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 12, pp. 4618–4630, Dec. 2020.