2021 IEEE 30th Asian Test Symposium (ATS) | 978-1-6654-4051-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ATS52891.2021.00013

ChaoPIM: A PIM-based Protection Framework for DNN Accelerators Using Chaotic Encryption

Ning Lin[1,2], Xiaoming Chen[1,2], Chunwei Xia[1,2], Jing Ye[1,2], Xiaowei Li[1,2]
[1]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
[2]University of Chinese Academy of Sciences
{linning19b, chenxiaoming, xiachunwei, yejing, lxw}@ict.ac.cn

***Abstract*—Although deep neural networks (DNNs) have been widely used, DNN models running on ASIC- or FPGA-based accelerators still lack effective and efficient protection. Once DNN models are stolen by attackers, it will not only infringe the intellectual property of model providers but also lead to security issues. The existing parameter encryption method brings greater power consumption, which is difficult to apply to resource-constrained edge devices. This paper proposes an effective and efficient framework – ChaoPIM to protect the security of DNN models by utilizing the chaotic encryption and the Processing-In-Memory (PIM) technology. Detailed experimental results show that our framework can effectively prevent attackers from using DNN models normally, as the accuracy of stolen models is quite low. Compared with the powerful Cortex-A53, Kryo-280, Intel-i5-8265U CPUs and TITAN V GPU, ChaoPIM achieves considerable performance improvements on various DNN models.**

***Index Terms*—DNN, Security, PIM, Chaotic Encryption**

I. Introduction

With the widespread application of deep neural networks (DNNs), the security protection of DNN models has become a critical concern. After DNN models are well trained on data cloud servers and deployed on edge DNN accelerators, security problems come from three contexts: cloud transmission, the storage of DNNs and the prediction procedure of DNN models. For instance, for ASIC-based accelerators, when the weights of DNN models are being loaded from the main memory (i.e., DRAM) to the processor, attackers can steal the weights from the processor-memory interface via side-channel attacks (e.g., [1]). However, well-trained DNN models require a large amount of well-labeled private data, sufficient training resources and rich experience of experts. Thus valuable DNN models can be sold as intellectual property. More importantly, once DNN models are leaked, it will incur serious security problems. Attackers can generate adversarial examples [2], which in turn makes DNN models output wrong prediction results. This brings challenges to the application of DNN models in significant fields, such as autonomous driving and medical treatment. Therefore, it is necessary to protect the security of DNN models.

The existing methods for protecting the security of DNN models can be divided into two categories — physical unclonable function (PUF) based encryption [3], [4] and parameter encryption [5]–[9]. PUF-based encryption usually utilizes the responses of PUF to protect the security of the weights of DNN models for FPGA- or ASIC-based accelerators. For instance, Guo *et al.* [3] use the responses of PUF to change the sign of weights of DNN models to protect the security of FPGA-based DNN accelerators. P^{3}M [4] utilizes the responses of PUF as secret keys, and then combines the process-in-memory (PIM) technology to implement the XOR encryption for protection of ASIC-based DNN accelerators. However, due to environmental variations (e.g., supply voltage), the responses of PUF are unstable and the Error Correcting Code is required to produce reliable responses, inevitably producing additional power consumption [4]. Besides, PUF brings additional latency overhead and power consumption, which are not illustrated in detail in previous studies [3], [4].

Parameter encryption is intended to directly encrypt the weights of DNN models by using a certain data encryption algorithm. For instance, the Advanced Encryption Standard (AES) is adopted to encrypt the weights of DNN models in study [5]. However, the power consumption of the AES module is too high (e.g., 29.8 mW per module) to be applied to resource-constrained edge devices. To reduce encryption overhead, study [6] utilizes the fast gradient sign method [2] to encrypt a small proportion of the weights of DNN models. However, the encryption effect depends on the size of adversarial perturbations added to the weights. The greater the values of adversarial perturbations are, the better the encryption effect is. As a result, the probability distribution of the weights is changed, which is easily detected by attackers. In addition, some researchers [8], [9] use obfuscation encryption methods along the rows of memristors to protect the security of DNN models for PIM-based DNN accelerators. However, the energy-consuming redirection modules are required to correctly accumulate the intermediate results of DNN models [9]. Recently, study [10] utilizes chaotic encryption method to protect the weights of DNN models running on GPUs. However, the procedure of transferring plaintext weights decrypted on CPUs to GPUs is not secure, and the power consumption for decrypting one layer of DNN models is very high (e.g., about 14 W in our observations). As a result, there is still a lack of efficient method to protect the security of FPGA- or ASIC-based DNN accelerators.

For the purpose of tackling the above problems, we propose a PIM-based chaotic encryption framework to protect the weights of DNN models for FPGA- or ASIC-based accelerators. The primary contributions of this work include:

- We introduce a novel PIM-based chaotic encryption

2377-5386/21/$31.00 ©2021 IEEE
DOI 10.1109/ATS52891.2021.00013

framework to protect the weights of DNN models. Specifically, our framework can effectively protect the weights of various DNN models running on ASIC- or FPGA-based accelerators, and the prediction accuracy after encryption is close to that of random guessing.
- We point out that it is not secure to decrypt the entire weights of DNN models at one time during the prediction procedure, and introduce a fine-grained protection scheme that can protect the security of weights during the whole procedure.
- We design a pipelined and reconfigurable dataflow, which can further reduce latency overhead and energy consumption required for the decryption of the weights.
- We compare latency overhead and energy consumption of our method with the state-of-the-art CPUs and GPU. For example, the proposed ChaoPIM achieve 6.85× speedup and 45.27× energy reduction than a Cortex-A53 CPU for the decryption of VGG16.

II. Preliminary and Motivation

A. Anold's Cat Map

Arnold's Cat Map (ACM) [11], [12] is a kind of chaotic encryption theory, which achieves encryption or decryption by rearranging the positions of the pixels of the image as shown in Fig. 1. Although the pixel values of the encrypted image have not changed, it can effectively realize the encryption. The encrypted image has no visual information at all. Based on this idea, we expand the weights of a certain convolution layer l of DNN models from four dimensions $C \times N \times K \times K$ to two dimensions $(C \times K) \times (N \times K)$. Here, C is the number of channels, and N is the number of convolutional kernels, and K is kernel size. When $K = 1$, ACM can be applied to the fully connected layer of DNN models.

1) Encryption: The formulation of ACM's two-dimensional matrix encryption is

$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = A^{\tau} \begin{bmatrix} x \\ y \end{bmatrix} (\mathrm{mod}\ S),\ A = \begin{bmatrix} 1 & p \\ q & p \cdot q + 1 \end{bmatrix}, \quad (1)$$

where (x, y) and (x_e, y_e) are the original position and encrypted position respectively. The p, q, and τ are integers, and S is the encryption range ($0 \leq S \leq \min\{C, N\}$). Different layer l of DNN models can own different encryption parameters. Therefore, the secret key $\mathcal{K}$ can be formulated as

$$\mathcal{K} = [\mathbb{L}, \{(\tau_{(l)}, p_{(l)}, q_{(l)}, S_{(l)}) | l \in \mathbb{L}\}], \quad (2)$$

where $\mathbb{L}$ denotes the encrypted layers of DNN models and subscript $_{(l)}$ is the layer index.

2) Decryption: According to the secret key $\mathcal{K}$ and encrypted position (x_e, y_e), the original position (x, y) can be caculated by

$$\begin{bmatrix} x \\ y \end{bmatrix} = A^{-\tau} \begin{bmatrix} x_e \\ y_e \end{bmatrix} (\mathrm{mod}\ S). \quad (3)$$

In this work, to reduce the latency overhead of calculating A^{τ} and $A^{-\tau}$ in Eq. (1) and Eq. (3), we set $p = 1$ and $q = 1$. Then A^{τ} and $A^{-\tau}$ can be reformulated as

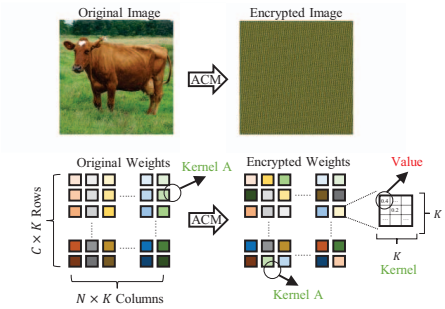

Fig. 1. ACM encryption.

$$A^{\tau} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}^{\tau} = \begin{bmatrix} f_{2\tau-1} & f_{2\tau} \\ f_{2\tau} & f_{2\tau+1} \end{bmatrix}, \quad (4)$$

$$A^{-\tau} = \begin{bmatrix} f_{2\tau+1} & -f_{2\tau} \\ -f_{2\tau} & f_{2\tau-1} \end{bmatrix}, \quad (5)$$

where f_{τ} is a fibonacci number. As the fibonacci number has a recursive formula (e.g., $f_1 = 1, f_2 = 1, f_{\tau+2} = f_{\tau+1} + f_{\tau}$), the calculation of A^{τ} and $A^{-\tau}$ has negligible latency overhead. In addition, to reduce the cost of hardware implementation of modulus operation for arbitrary encryption range, we set $S \in 2^n$ ($n = 0, 1, 2, ...$) in Eq. (1) and Eq. (3). Since the modulus of S is to take low-order n bits as output, there is almost no hardware overhead.

Each value in convolutional or fully connected layers can be encrypted by ACM algorithm. However, larger encryption range will lead to greater latency overhead. To reduce this overhead, an entire convolution kernel can be viewed as a pixel of the image for ACM encryption or decryption. In this work, for smaller encryption range (e.g., the first layer of DNN models), ACM is used to encrypt all values of layers of DNN models. For a larger encryption range, entire kernels are encrypted. After ACM encryption, the positions of original weights change, but values of weights don't. For example, the kernel A in Fig. 1 is moved from the second row of the last column to the last row of the second column after ACM encryption, which can be completely restored to the original position after decryption. Therefore, encryption and decryption of ACM are completely reversible and lossless.

B. RRAM Accelerator

Recently, resistive random access memory (RRAM) dot product accelerators with crossbars architecture have widely been investigated (e.g., [13], [14]). The digital input signal of crossbars is converted into the voltage signal by a digital-to-analog converter (DAC), and the latter is applied to each word line (WL) of crossbars. The crossbar is composed of multiple RRAM cells. These cells share a WL in the same row, and share a bit line (BL) in the same column. Each RRAM cell is used as a resistor and can be switched to two states, a high resistance state (HRS) and a low resistance state (LRS). Therefore, one cell can be used to represent logic "0" and "1", known as single-level cell (SLC). Each RRAM cell can also represent more bits information by switching to multiple

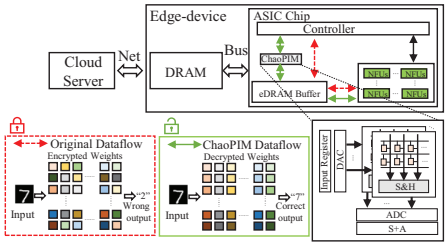

Fig. 2. Overiew of the proposed architecture.

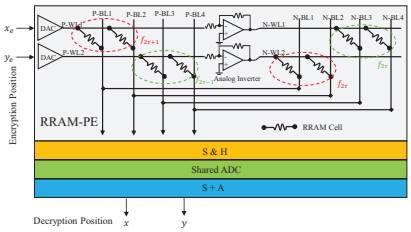

Fig. 3. The RRAM PE architecture.

levels of resistance [15], which is called the multi-level cell (MLC). Based on Ohm's Law, current generated from RRAM cells flows every BL and is converted into a digital signal by an analog-to-digital converter (ADC).

The RRAM-based crossbar can not only store the values to the RRAM cells but also can accomplish a matrix-vector multiplication (MVM) in one cycle. Due to the reduction of data movement involved in the calculation, the energy consumption is extremely low. Therefore, many researchers use RRAM-based crossbars to accelerate prediction and training procedure of DNN models (e.g., [13], [14]). In this work, we utilize RRAM-based crossbars to achieve the acceleration of ACM decryption.

III. ChaoPIM Dataflow

A. Overview

Fig. 2 describes the architecture of our proposed ChaoPIM framework for ASIC-based DNN accelerators. It should be noted that our method is a general protection framework and can be applied to any ASIC or FPGA-based DNN accelerators (e.g., DaDianNao [16]). Specifically, our framework is intended to implement a decryption accelerator for ACM based on existing DNN accelerators. RRAM-based crossbars natively have the characteristics of storage and calculation. To avoid frequent transmission and reduce the possibility of being stolen by side-channel attacks, we use the storage property of RRAM-based crossbars to store secret keys into RRAM cells. Because the calculation of ACM decryption can be performed on RRAM-based crossbars, the CMOS-based processing module is no longer needed.

During prediction procedure, the weights of DNN models are being loaded from the DRAM to the processor through the processor-memory interface. In this context, the weights will be easily attacked if they are all plaintexts. Therefore, the weights on the DRAM and the on-chip buffer (e.g., eDRAM) are kept encrypted, which is achieved by the ACM encryption. To obtain the correct prediction results of DNN models, firstly, the ChaoPIM accelerator requires to be invoked by a controller to calculate correct addresses (i.e., original positions) of the weights. Then, neural functional units (NFUs) take the weights from the correct addresses of the on-chip buffer to calculate the output results of DNN models. Without ChaoPIM, NFUs would directly use the encrypted weights and eventually output wrong prediction results. Specifically, in the original dataflow, the input image "7" and the encrypted weights are used to predict the output result of DNN models, and an error result "2" is produced. As a contrast, in our proposed ChaoPIM dataflow, the correct output "7" can be produced because ChaoPIM decrypts the weights. The proposed ChaoPIM accelerator is composed of multiple RRAM processing elements (RRAM-PEs) which share the ADC, Sample and Hold (S&H), and Shift and Add (S+A) components.

B. RRAM-PE Architecture

Fig. 3 shows the detailed architecture of a RRAM-PE. To represent positive and negative values, an analog inverter is used to connect two crossbars according to study [17]. One of the crossbars is used to represent positive values, and the other is used to represent negative values. The word line (WL) and the bit line (BL) of the positive crossbar are P-WL and P-BL, respectively. The WL and the BL of the negative crossbar are N-WL and N-BL, respectively. In order to reduce the resistance noise, SLC RRAM cells are adopted for RRAM-PEs. The RRAM cells connected between the WL and the BL are used to store the decryption parameters $f_{2\tau+1}, f_{2\tau-1}$ and $-f_{2\tau}$ in the $A^{-\tau}$ matrix in Eq. (5) and Eq. (3). The width of $f_{2\tau+1}, f_{2\tau-1}$ and $-f_{2\tau}$ depend on the size of τ. For example, if $\tau = 5$, the fibonacci number $f_{2\tau+1} = 89, f_{2\tau-1} = 34$ and $-f_{2\tau} = -55$, which can be encoded with 7-bit width. Therefore, 7 SLC RRAM cells are used to store each value in the $A^{-\tau}$ matrix. When the input of RRAM-PE is the encrypted position $\{x_e, y_e\}$, the original position $\{x, y\}$ can be calculated by $x = f_{2\tau+1} \times x_e + (-f_{2\tau}) \times y_e, y = f_{2\tau-1} \times y_e + (-f_{2\tau}) \times x_e$, which is achieved by RRAM crossbars of RRAM-PE. Because the maximum encryption range in this work is 512, i.e., the largest number of convolution kernels of DNN models used in our experiment, the encryption position $\{x_e, y_e\}$ can be encoded with 9-bit width. Furthermore, if 1-bit DACs are used, a decrypted position can be calculated in 9 cycles.

After the analog signal is processed by S&H, shared ADC and S+A components, the final digital original position $\{x, y\}$ is generated after taking the correct bit (e.g., $mod\, S$). Similarly, if RRAM cells store the encryption parameters of A^{τ} matrix in Eq. (4) and Eq. (1), RRAM-PEs can also be adopted to achieve ACM encryption.

C. Fine-grained Protection Scheme

The existing encryption method usually invokes the decryption procedure at one time on the ASIC chip. However, the weights which are decrypted at one time are plaintexts during the prediction procedure of DNN models, and these weights can be stolen by side-channel attacks. Therefore, one-time decryption of weights is not secure. To protect the security of

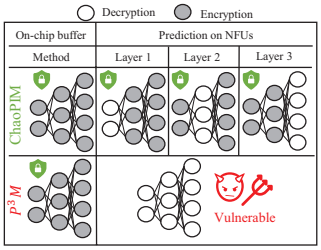

Fig. 4. Fine-grained protection.

weights during the entire prediction procedure, we propose a fine-grained protection scheme which can decrypt a part of the weights of DNN models required for prediction on the NFUs. Specifically, the weights of one layer are decrypted each time. Even if malicious attackers steal a part of the weights of DNN models, they cannot normally use its functions. Fig. 4 shows the difference between our method and previous P^{3}M method [4]. Our method can protect the weights of DNN models throughout the prediction procedure. However, the weights in the P^{3}M method are all plaintexts during the prediction procedure, which are vulnerable to side-channel attacks.

D. Pipeline

During the prediction procedure of DNN models, the decryption between different layers of DNN models are independent of each other, and the latency overhead can be reduced by utilizing the inter-layer parallelism. Therefore, we design an inter-layer pipeline to achieve high throughput as shown in Fig. 5. Assuming that a DNN model consists of N layers, the latency overhead to finish the decryption procedure is $t_i^{ChaoPIM}$, and the prediction time of one layer is t_i^{NFUs}. Then, $\sum_{i=1}^{N}(t_i^{ChaoPIM}+t_i^{NFUs})$ is required to finish the entire procedure without inter-layer pipeline. On the contrary, the latency overhead is $t_{i=1}^{ChaoPIM}+\sum_{i=2}^{N}\max\{t_i^{ChaoPIM},t_i^{NFUs}\}$ with the pipeline.

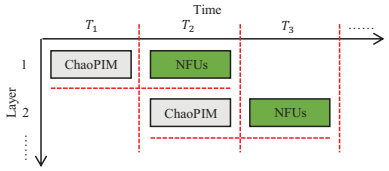

Fig. 5. ChaoPIM pipeline.

E. Reconfigurable RRAM-PEs

When the prediction time of the (i-1)-th layer of DNN models on NFUs is shorter than the decryption time of the i-th layer for ACM on a single RRAM-PE (i.e., $t_{i-1}^{NFUs}<t_i^{ChaoPIM}$), the original throughput rate of DNN accelerators will be reduced. Ideally, we can improve the throughput by paralleling multiple RRAM-PEs to reduce the latency overhead, but this inevitably brings higher power consumption. Therefore, the decryption time $t_i^{ChaoPIM}$ should be shorter than t_{i-1}^{NFUs}. Moreover, since t_{i-1}^{NFUs} may be extremely small, lots of RRAM-PEs are needed to achieve a smaller $t_i^{ChaoPIM}$. Hence, $t_i^{ChaoPIM}$ can be shorter than the decryption time t_i^{R} on a reference hardware (e.g., a powerful CPU), which may be longer than t_{i-1}^{NFUs}. To obtain the optimal number of RRAM-PEs, we formulate the following optimization problem,

$$\begin{aligned}&\min_{\alpha_i}\ \alpha_i \times Power^{RRAM-PE}\\ &\text{s.t.}\ \ t_i^{ChaoPIM}\le \max\{t_{i-1}^{NFUs},t_i^{R}\}\end{aligned}, \tag{6}$$

where $Power^{RRAM-PE}$ is the power consumption of a RRAM-PE component, and α_i is the number of RRAM-PEs for the i-th layer of DNN models. For a given encryption range $S_{(i)}$, the decryption latency overhead $t_i^{ChaoPIM}$ of ChaoPIM is formulated as,

$$t_i^{ChaoPIM}=\frac{S_{(i)}\times S_{(i)}}{\alpha_i}\times t_i^{RRAM-PE}, \tag{7}$$

where $t_i^{RRAM-PE}$ is the latency overhead required for a single RRAM-PE to decrypt one encrypted position. The optimal number of RRAM-PEs of the i-th layer can be calculated by,

$$\alpha_i^{opt}=\lceil\frac{t_i^{RRAM-PE}}{\max\{t_{i-1}^{NFUs},t_i^{R}\}}\rceil\times(S_{(i)}\times S_{(i)}). \tag{8}$$

Eq. (8) can be used to obtain the optimal number of RRAM-PEs (i.e., α_i^{opt}) needed for each layer of DNN models. Finally, according to Eq. (8) and Eq. (6), the minimum power consumption can be obtained.

IV. Evaluation

A. Experimental Setup

Benchmarks and Baselines. We evaluate the proposed ChaoPIM framework on four popular DNN models, including AlexNet, VGG11, VGG16 and ResNet18. These DNN models are well trained on ImageNet dataset with 16-bit fixed-point weights by Pytorch. We use DaDianNao-like [16] NFUs with eDRAM weights buffers to evaluate the hardware performance of ChaoPIM. For comparison, we implement ACM encryption on a powerful Cortex-A53 CPU (2.0 GHz), Kryo-280 CPU (2.45 GHz) and Intel-i5-8265U CPU (1.6 GHz) using C++, and implement it on a TITAN V GPU (1.2 GHz) using CUDA.

Configurations. The power consumption of crossbars and the analog inverter are evaluated with HSPICE simulations in the 45nm process node. We develop a cycle-accurate simulator to evaluate the latency overhead of ChaoPIM. RRAM cells have a resistance range of 5kΩ $\sim$ 1MΩ and the read voltage of 0.5V. ADC used in this work is based on the successive approximation register (SAR) design, and the power consumption of SAR ADC increases linearly as the bit resolution increases [18], i.e., $P_{SAR}\approx\beta N_b$, where β is a constant term and N_b is the resolution of SAR ADCs. The resolution of ADCs and DACs in our experiment is 2-bit and 1-bit, respectively. Parameters of ADCs, DACs, S&H, S+A and clock (i.e., 10 MHz) are consistent with those of ISAAC [14]. The accuracy loss is set to 20% [8] to meet the encryption requirement. The latency overhead of a powerful Intel-i5-8265U CPU is choosed as t_i^R in Eq. (8). The secret keys encoded with 7-bit width are randomly generated to encrypt DNN models in this work.

Fig. 6. Accuracy influence.

B. Accuracy Influence

To illustrate the effect of ACM encryption, Fig. 6 shows accuracy after the single-layer ACM encryption and the layer-by-layer ACM encryption. Two conclusions can be drawn. First, the single-layer encryption can make DNN models lose the recognition ability. The accuracy loss of single-layer encryption is larger than 20% for any layers of DNN models. For instance, the accuracy after encrypting the 8th layer of VGG16 is 46.77%, which is 24.82% smaller than the original accuracy – 71.59%. The accuracy after encrypting the 7th layer is 0.108%, which is equivalent to that of random guessing in the classification task of ImageNet. The similar conclusion also appears in ResNet18. Second, the layer-by-layer encryption is more secure than the single-layer encryption, and the accuracy loss of the layer-by-layer encryption is much larger than that of the single-layer encryption. For instance, the accuracy of ResNet18 after the layer-by-layer encryption is only 0.076%, which is more lower than that of the single-layer encryption.

TABLE I
THE ACCURACY INFLUENCE WITH CHAOPIM.

DNN Model	Original ACC (16-bit)	Without ChaoPIM	ChaoPIM
AlexNet	56.52%	0.09%	56.52%
VGG11	69.02%	0.064%	69.02%
VGG16	71.59%	0.1%	71.59%
ResNet18	69.75%	0.076%	69.75%

To achieve ideal protection effect, ACM encryption can be performed on all layers of DNN models. Table I shows the accuracy after the layer-by-layer encryption. Without ChaoPIM, DNN accelerators will output completely wrong prediction results. The accuracy of DNN models is close to that of random guessing, achieving the purpose of protection of weights. When ChaoPIM is adopted for decryption, the accuracy can be completely restored.

C. Hardware Performance

Latency overhead. To illustrate the efficiency of our method, we compare the latency overhead of ChaoPIM with that of three kinds of CPUs and a GPU for ACM decryption. As shown in Fig. 7, ChaoPIM can achieve faster decryption speed. For instance, when decrypting the 6th layer of VGG16, our method outperforms the Cortex-A53, Kryo-280, Intel-i5-8265U CPUs and TITAN V GPU by about 9.59×, 3.49×, 2.09× and 2.28×, respectively. In addition, thanks to our proposed pipeline, ChaoPIM has no additional latency overhead on some layers. For example, the latency overhead of the 3rd, 4th, 5th, 7th, and 8th layers of VGG16 is 0 us. This is because the decryption time of these layers is shorter than

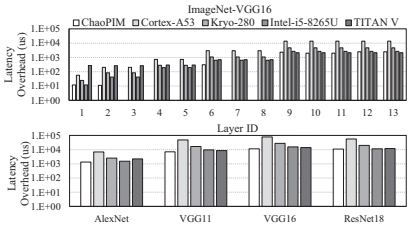

Fig. 7. Latency comparison.

the prediction time of the previous layers of DNN models. For decrypting the entire VGG16, our method can achieve about 6.85×, 2.38×, 1.35× and 1.22× speedup compared with the Cortex-A53, Kryo-280, Intel-i5-8265U CPUs and TITAN V GPU. Similar conclusions also appear in AlexNet, VGG11 and ResNet18.

Power and Energy consumption. Fig. 8 details the power consumption of different components of ChaoPIM, and Fig. 9 illustrates the power and energy consumption on different hardware platforms. As shown in Fig. 8, at first, as multiple crossbars share a SAR ADC, the power consumption of crossbars is higher than that of ADCs. The power consumption of DACs, S&H and S+A is relatively lower. In addition, because the latter layer of VGG16 has a larger encryption range (i.e., 512), more RRAM-PEs are needed to achieve fast decryption. For instance, the number of RRAM-PEs is 89 for the 9th, 10th, 11th, 12th and 13th layer, so the energy consumption is higher than that of the previous layer. Last but not least, the 3rd layer of VGG16 requires the least number (i.e., 4) of RRAM-PEs. This is because the prediction time of the second layer of VGG16 on the DaDianNao-like accelerator is longer (i.e., about 662.97 us), and the encryption range of the third layer is smaller (i.e., 64), so fewer RRAM-PEs are needed to achieve the decryption procedure.

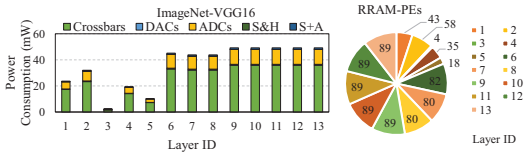

Fig. 8. Power breakdown of RRAM-PEs.

Fig. 9 illustrates that ChaoPIM requires lower power consumption to complete the decryption procedure. For instance, when decrypting the 5th layer of VGG16, ChaoPIM is 33.3×, 92.86×, 1417.08× and 3171.46× more power-efficient than the Cortex-A53, Kryo-280, Intel-i5-8265U CPUs and TITAN V GPU, respectively. Besides, the number of RRAM-PEs for decrypting the 5th layer of VGG16 is 18, and the average power consumption of each RRAM-PE is 0.56 mW, which is 53.21× more power-efficient than that of the AES module [5].

Due to lower power consumption and faster decryption speed, ChaoPIM consumes very little energy. For example, when decrypting the 9th layer of VGG16, ChaoPIM is 46.32×, 45.58×, 301.48× and 520.96× more energy-efficient than the Cortex-A53, Kryo-280, Intel-i5-8265U CPUs and TITAN V GPU, respectively. For decrypting the entire VGG16, ChaoPIM achieves 45.27×, 44.36×, 308.27× and 603.19× energy reduction compared with the Cortex-A53, Kryo-280,

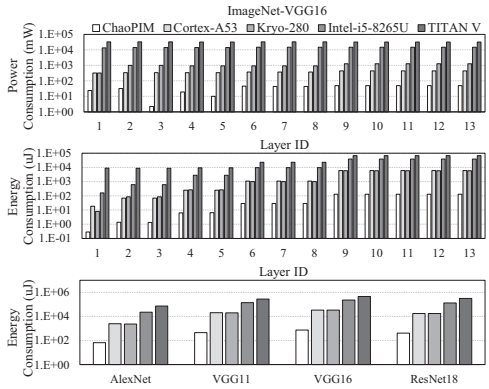

Fig. 9. Power and Energy comparison.

and Intel-i5-8265U CPUs and TITAN V GPU, respectively. Therefore, ChaoPIM can greatly reduce energy consumption by utilizing the PIM technology.

D. Security Analysis

The weights of DNN models are in the chaotic state after ACM encryption on the DRAM or the on-chip buffer. Even if attackers obtain the weights of DNN models by side-channel attacks, the function of DNN models cannot be used normally. Therefore, one possible attack method is to verify the prediction results of DNN models by using the validation dataset. However, the attack complexity is very larger, which approaches $O\left(2^{|\mathbb{L}|}\cdot\prod_{l=1}^{|\mathbb{L}|}P_{(l)}(\tau)S_{(l)}\right)\cdot O(V)$, where $|\mathbb{L}|$ is the number of layers of DNN models, $O(V)$ is the complexity of a validation procedure, and $P(\tau)$ is the period of τ for ACM encryption (generally $P(\tau)\geq 10$ [19]). It takes long time to attack one layer of VGG16 on the ImageNet validation dataset. For example, assuming that the shape of an encryption layer of VGG16 is $512\times512\times3\times3$, and that a validation procedure requires 250 seconds [6] on a hardware platform, the complexity of this layer approaches $10\times\sum_{n=1}^{9}(512-2^n)^2\approx16M$. This attack consumes about $250\times16M=4000$ million seconds (i.e., 126.8 years). Considering that more layers can be encrypted, the complexity will further increase.

TABLE II
THE RELIABILITY OF CHAOPIM.

Leakage	ACC (%)			
	AlexNet	VGG11	VGG16	ResNet18
25%	0.1	0.094	0.112	0.086
50%	0.162	0.168	0.12	0.092
75%	23.85	0.126	0.09	0.076

According to Kerckhoff's Principle and Shannon's Maxim [20], attackers can understand the encryption method except for secret keys. To illustrate the reliability of our framework, we assume that the secret keys of some DNN layers are leaked. Due to extremely low accuracy, these DNN models are secure as shown in Table II. For instance, when 75% of secret keys are leaked, the accuracy of AlexNet is 23.85%, which is 32.67% lower than the original accuracy. For ResNet18 with more layers, the accuracy is only 0.076%. Therefore, our framework is reliable. Even if most of secret keys are leaked (i.e., 75%), DNN models cannot be used normally.

V. CONCLUSIONS

In this paper, we propose an effective and efficient PIM-based protection framework for ASIC or FPGA-based DNN accelerators. Even if attackers obtain the weights of DNN models by side-channel attacks, DNN models cannot be used normally by using ACM encryption. The accuracy after encryption is close to that of random guessing. By utilizing PIM technology, energy consumption of ACM decryption can be greatly reduced. The extensive experiments on four benchmarks indicate that our framework can achieve quite high security with extremely low hardware overhead.

VI. ACKNOWLEDGMENTS

This paper is supported in part by National Natural Science Foundation of China (NSFC) under grant No. (U20A20202, 62090024, 61876173, 61804155); in part by the National Key Research and Development Program of China under Grant 2020YFB1600201. The corresponding authors are Xiaoming Chen and Jing Ye.

REFERENCES

[1] W. Hua *et al.*, "Reverse engineering convolutional neural networks through side-channel information leaks," in *DAC*, 2018, pp. 1–6.
[2] I. J. Goodfellow *et al.*, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.
[3] Q. Guo *et al.*, "Puf based pay-per-device scheme for ip protection of cnn model," in *ATS*, 2018, pp. 115–120.
[4] W. Li *et al.*, "P^{3}m: a pim-based neural network model protection scheme for deep learning accelerator," in *ASPDAC*, 2019, pp. 633–638.
[5] W. Shan *et al.*, "Machine learning assisted side-channel-attack countermeasure and its application on a 28-nm aes circuit," *IEEE JSSC*, vol. 55, no. 3, pp. 794–804, 2019.
[6] Y. Cai *et al.*, "Enabling secure in-memory neural network computing by sparse fast gradient encryption." in *ICCAD*, 2019, pp. 1–8.
[7] S. Huang *et al.*, "Xor-cim: compute-in-memory sram architecture with embedded xor encryption," in *ICCAD*, 2020, pp. 1–6.
[8] M. Zou *et al.*, "Security enhancement for rram computing system through obfuscating crossbar row connections," in *DATE*, 2020, pp. 466–471.
[9] S. Huang *et al.*, "New security challenges on machine learning inference engine: Chip cloning and model reverse engineering," *arXiv preprint arXiv:2003.09739*, 2020.
[10] N. Lin *et al.*, "Chaotic weights: A novel approach to protect intellectual property of deep neural networks," *IEEE TCAD*, 2021.
[11] V. I. Arnol'd *et al.*, *Ergodic problems of classical mechanics*. WA Benjamin, 1968.
[12] G. Peterson, "Arnold's cat map," *Math Linear Algebra*, vol. 45, pp. 1–7, 1997.
[13] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, 2016, pp. 27–39.
[14] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ISCA*, 2016, pp. 14–26.
[15] F. Alibart *et al.*, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, jan 2012.
[16] Y. Chen *et al.*, "Dadiannao: A machine-learning supercomputer," in *MICRO*, 2014, pp. 609–622.
[17] B. Li *et al.*, "Rram-based analog approximate computing," *IEEE TCAD*, vol. 34, no. 12, pp. 1905–1917, 2015.
[18] Q. Wang *et al.*, "Neuromorphic processors with memristive synapses: Synaptic interface and architectural exploration," *J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 4, May 2016.
[19] J. Bao *et al.*, "Period of the discrete arnold cat map and general cat map," *Nonlinear Dynamics*, vol. 70, no. 2, pp. 1365–1375, 2012.
[20] C. E. Shannon, "Communication theory of secrecy systems," *Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.