Chaotic Weights: A Novel Approach to Protect Intellectual Property of Deep Neural Networks

Ning Lin, *Graduate Student Member, IEEE*, Xiaoming Chen, *Member, IEEE*, Hang Lu, *Member, IEEE*, and Xiaowei Li, *Senior Member, IEEE*

***Abstract*—Despite the high accuracy achieved by the deep neural network (DNN) technique, there is still a lack of satisfying methodologies to protect the intellectual property (IP) of DNNs, which involves extensive valuable training data, abundant hardware training resources, and fine-tuning skills of experienced experts. Existing solutions based on watermarking cannot prevent malicious/unauthorized users from using well-trained DNNs. This paper proposes chaotic weights (ChaoWs), a novel framework based on the Chaotic Map theory, to protect the IP of DNN providers with very low overhead. Specifically, in order to alleviate the storage overhead and abridge the decryption time, our method makes convolutional or fully connected kernels chaotic by exchanging the weight positions to obtain a satisfying encryption effect, instead of using the conventional idea of encrypting the weight values. Comprehensive experimental evaluations on image classification, semantic segmentation, and name generation demonstrate that ChaoW can effectively protect the IP of DNNs without damaging the inference accuracy, and the impact on the inference speed is negligible.**

***Index Terms*—Chaotic encryption, deep neural network (DNN), intellectual property (IP) protection.**

I. Introduction

MACHINE learning (ML) techniques, especially deep neural networks (DNNs), have nowadays become a de facto standard for a wide range of application domains, such as self-driving vehicles, speech recognition, and natural language processing (NLP) tasks. State-of-the-art DNNs usually involve plenty of annotated data sets, powerful training resources (e.g., Google TPUs), and fine-tuning skills for hyper-parameters of DNNs. The value of DNNs is therefore significantly increased due to these requirements. For instance, the training service cost of $\text{BERT}_{\text{LARGE}}$ [1], an extremely powerful language representation DNN model that has 340 million parameters, is about 16 (TPUs) $\times$ 4 (days) $\times$ 24 (h) $\times$ 4.5 (USD per hour) [2] $=$ 6912 USD. Without considering the cost of collecting training data and fine-tuning skills, the training service cost is already considerably expensive. Such valuable DNNs are typically confidential for model providers. Therefore, protecting the intellectual property (IP) of DNNs when designing and deploying DNNs has become an urgent problem to be solved in the ML area.

To protect valuable DNNs, one popular solution is neural network watermarking, which embeds a mechanism into a DNN model such that the ownership can be verified by model providers. Uchida *et al.* [3] and Nagai *et al.* [4] made the first attempt to embed watermarks into DNNs, which uses a regularizer to retrain the parameters. Although this work does not impair the performance and inference speed of DNN models on the CIFAR-10 dataset [5], it is assumed that model providers can verify the parameters of a target DNN in a white-box way. However, white-box verification is not easy, and model providers can only access the input and output interface of DNN models, but not the parameters. The white-box method loses its effect as long as the parameters of DNNs are not open, which leads to valuable DNN models to be maliciously distributed and normally used free of charge.

To address the white-box limitations, Adi *et al.* [6] extended the watermarking approach to support black-box mode verification, which only requires the input and output interface to verify the ownership. Specifically, they trained a valuable DNN model on both the original dataset and a modified dataset where each image is watermarked by the model provider's signature. The labels of the modified images are elaborated and different from the original labels. To verify the watermark, the model provider issues queries using the signed images and tests whether the DNN model returns the designated labels. Similar black-box watermarking methods have been proposed in other studies [7]–[10]. However, these black-box methods also have limitations. First, the size of the watermark pattern of the input images has a paramount influence on the result. If it is too large, it is easy to be detected [11], [12], and if it is too small, the watermark effect is poor. Second, the robustness is poor. It is reported that

Manuscript received March 23, 2020; revised June 26, 2020; accepted July 31, 2020. Date of publication August 20, 2020; date of current version June 18, 2021. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFA0701500; in part by the Key Research Program of Frontier Sciences, CAS under Grant ZDBS-LY-JSC012; in part by the Strategic Priority Research Program of CAS under Grant XDB44000000; in part by the National Natural Science Foundation of China under Grant 61804155 and Grant 61532017; in part by the Youth Innovation Promotion Association CAS; in part by the Young Elite Scientists Sponsorship Program by CAST under Grant 2018QNRC001; and in part by the Beijing Academy of Artificial Intelligence. This article was recommended by Associate Editor R. S. Chakraborty. *(Corresponding author: Xiaoming Chen.)*

Ning Lin, Hang Lu, and Xiaowei Li are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China, and also with the School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: linning19b@ict.ac.cn; luhang@ict.ac.cn; lxw@ict.ac.cn).

Xiaoming Chen is with the Center for Intelligent Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China, and also with the School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: chenxiaoming@ict.ac.cn).

Digital Object Identifier 10.1109/TCAD.2020.3018403

the signature can be detected and eliminated from the input images by utilizing the auto-encoder technique [13]. Third, a lot of retraining procedures are required to restore the original accuracy of DNN models. At last, and even more crucial, watermarking-based methods do not have the ability of preventing malicious/unauthorized users from using well-trained DNNs. Therefore, existing neural network watermarking methods cannot obtain a satisfying protection effect of the DNN IP.

A promising solution to protect DNNs is parameter encryption, which makes the weights of DNNs unavailable to malicious/unauthorized users. Even though the weights are stolen or cloned (e.g., by side-channel attacks [14]–[17]), the DNNs will deliberately yield wrong predictions if the weights are not decrypted. A recent study [18] has shown that encrypting only 20 weight parameters per layer in ResNet-101 [19] is capable of thwarting the confidentiality attacks to DNNs parameters. Specifically, they make use of the fast gradient sign method (FGSM) [20] to generate the adversarial perturbations and then add the perturbations to the weights of DNNs to achieve parameter encryption. The perturbation intensity in their method has a vital influence on the encryption effect. When the perturbation intensity is too large, the distribution of weights will be changed, and the encrypted weights become outliers which increases the risk of being detected. On the contrary, the encryption effect is poor if the perturbation intensity is too small. Hence, selecting a proper perturbation intensity requires rich experience. Besides, the method is devised for computing-in-memory (CiM) systems. Therefore, there is still a lot to be improved for practical general computing platforms, such as graphics processing units (GPUs), field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs).

There are plenty of homomorphic encryption (HE)-based methods, such as Gazelle [21], XONN [22], and MiniONN [23], which enable third parties (e.g., Google's Cloud ML Engine [24]) to compute certain functions on encrypted data without decrypting it. It is worth noting that these HE-based methods are used to protect the privacy of input data for model users in the privacy-preserving area, rather than the IP of DNNs for model providers. Whereas the goal of our paper is to protect the IP of DNNs, and we primarily focus on protecting the parameters of DNNs, for which HE is incompetent.

Consequently, searching for an efficient and effective IP protection approach for DNN providers has practical significance. In this article, we propose an efficient and effective framework Chaotic Weight (ChaoW), standing for ChaoWs, which protects the IP of DNN providers with negligible overhead. Different from watermarking-based approaches, our approach is based on the scenario where users must purchase the secret key to decrypt the model; otherwise, wrong prediction results will be returned. Our framework does not need to retrain the weights of DNNs, and only a subset of the validation dataset is required to test the encryption effect using off-the-shelf deep learning software tools, such as PyTorch [25] and TensorFlow [26]. We do not modify the parameters to encrypt the DNNs like [18]. Instead, we take advantage of advanced chaotic encryption algorithms to achieve our goal. Specifically, we utilize a state-of-the-art chaotic encryption algorithm to make convolutional and/or fully connected weights chaotic by exchanging the weight positions. The weight distribution is not changed so the encrypted DNNs cannot be cracked by analyzing the weight statistics. The adopted chaotic encryption algorithm is very fast. Extensive experimental results on computer vision and NLP tasks show that our method does not impair the inference accuracy of DNNs, and the impact on the inference speed is negligible.

TABLE I
QUALITATIVE COMPARISON BETWEEN CHAOW AND TWO EXISTING METHODS

	White/Black Watermarking	Parameter Encryption	ChaoW
Retrain	Yes	No	No
Fidelity	Poor	Perfect	Perfect
Concealment	Poor	Poor	Perfect
Platform	Universal	CiM	Universal

II. RELATED WORK

In this section, we briefly summarize two existing DNN protection methods, and also present a comparison between our proposed method and the existing methods in Table I.

A. White-Box and Black-Box Watermarking

According to whether the DNN internal parameters need to be accessed during the verification process, the watermarking methods can be divided into two categories. One is white-box watermarking, represented by the work proposed by Uchida *et al.* [3] and Nagai *et al.* [4], which requires to access DNNs' internal parameters for verification. The other is black-box watermarking [6]–[13] which only requires the input and output interface of DNNs to conduct the verification. Both watermarking methods need the training data to retrain/fine-tune the DNNs' parameters. These retrained parameters are different from the previous ones, resulting in poor concealment. Currently, the watermarking methods only show fidelity on small datasets, such as MNIST [27] and CIFAR-10 [5] through retraining the DNN parameters, and there is no theory to ensure the fidelity on large-scale datasets such as ImageNet [28] in more realistic scenarios. Moreover, watermarking-based methods can only verify the IP but cannot protect the IP.

B. Parameter Encryption

Unlike watermarking-based methods, which can only verify the DNN IP, recently Cai *et al.* [18] utilized parameter encryption to prevent malicious users from using DNNs normally. Through FGSM [20], parameter encryption can theoretically ensure that the decrypted parameters are completely identical to the original ones, without causing loss of accuracy, so it has good fidelity. From the concealment point of view, although only very few parameters are selected for encryption, the parameter values will be changed as outliers which may be detected, so the concealment is poor. For a qualitative comparison, we list the characteristics of the two methods in Table I.

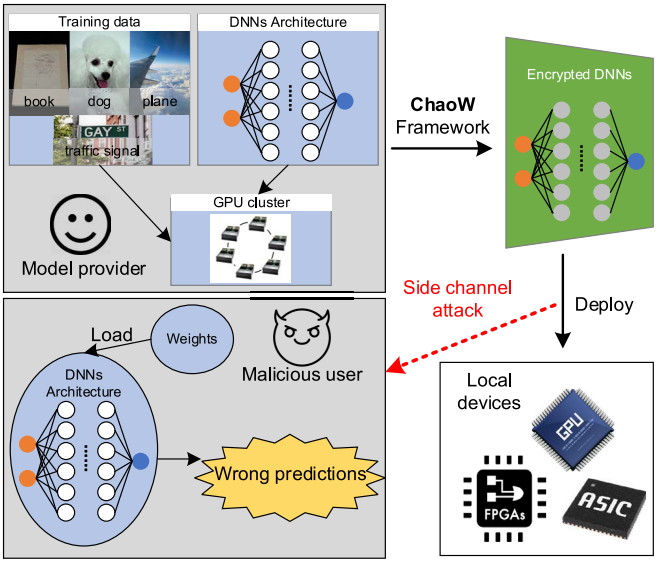


Fig. 1. General concept of ChaoW and threat model. The model provider deploys the well-trained DNNs to local devices of the user, and the malicious user may steal the DNNs. Our ChaoW framework can protect the IP of the model provider.

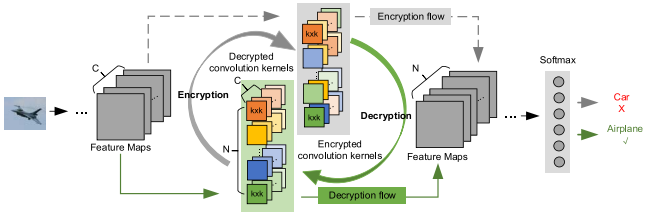


Fig. 2. Concrete framework of ChaoW. C stands for the numbers of input feature maps and N represents the numbers of convolution kernels for layer l.

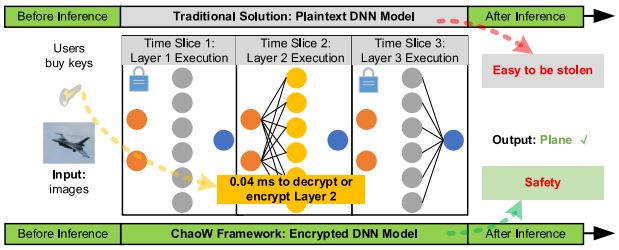


Fig. 3. Sequential unraveling of ChaoW. The speed of encryption or decryption is very fast, thus the decryption is invoked during the execution of inference procedure.

III. Chaotic Weights Framework

A. Problem Statement

We illustrate the problem statement and the general concept of our proposed ChaoW framework in Fig. 1.

Model providers spend a great deal of time and money to collect datasets, design the DNN architecture and train the weights on GPU clusters. Thus, model providers would like to monetize the DNNs on various platforms (e.g., GPUs and ASICs). Owing to the privacy policy, the well-trained DNNs are usually deployed on users' local devices rather than cloud platforms. Without any protection approach, the DNN with the weights may be maliciously distributed and normally used free of charge. Hence, the IP of model providers should be protected.

Malicious users might extract the well-trained weights and architectures of DNNs by some attack approaches (e.g., by physical side-channel attacks [14]). If the weights are in the form of plaintext, malicious users may redistribute and monetize the plaintext DNNs without permission from model providers, thus copyright infringement is caused.

ChaoW is a novel DNN IP protection framework for model providers, which encrypts the weights of DNNs by employing a state-of-the-art chaotic method. ChaoW does not change the values of the weights but the encrypted weights cause a sharp drop in the inference accuracy. Hence, even if the weights are stolen by malicious users, the function of DNNs is completely lost, achieving the goal of DNN IP protection for model providers.

B. Concrete Framework

Fig. 2 depicts the concrete encryption and decryption procedure of our proposed method. The convolution kernel dimension of layer l is $C \times N \times k \times k$, where C and N correspond to the number of input feature maps and output feature maps, respectively, and k is the size of the convolution kernel, which is usually 3, 5, or 7. When $k = 1$, it can be regarded as a fully connected layer. Assume that the feed-forward flow of a DNN model is propagated to layer l where layer l keeps encrypted. At this moment, if there is no secret key to decrypt the encrypted layer l, the forward propagation will pass the encrypted flow, and finally, the output of the DNN returns erroneous results. In contrast, if layer l is decrypted at this time, the forward propagation will pass the decrypted flow and the output of the DNN will return the correct results. It is worth noting that encryption and decryption are invoked by rapidly exchanging the positions of weights, specifically, in our implementation, by exchanging weight positions in the first two dimensions—$C \times N$.

To protect the IP of DNNs throughout the process, we describe the invoked procedure of the encryption or decryption in Fig. 3. If the encrypted DNN is decrypted before the inference procedure, the weights will remain in plaintext during all the inference time. Therefore, plaintext DNNs are easily stolen during the inference procedure. To avoid this, the decryption of ChaoW is invoked during the execution of the inference procedure at the same time when the inference flow is executed to the encrypted layer. Then, this layer restores the encryption quickly when the inference flow is finished. Specifically, our experiment demonstrates that the decryption or encryption latency of ChaoW is about 40 us for one layer, which is negligible compared with the inference time. In summary, the advantages of ChaoW are listed as follows.

1) Great Fidelity: Previous techniques [3], [4], [6] require lots of additional retraining procedures to fine-tune the weights so as to ensure that the accuracy of DNNs is not degraded. Whereas our method can completely restore the original accuracy by just changing the positions of weights.

2) Low Hardware Overhead: Only the positions of the convolutional or fully connected kernels are exchanged to encrypt

Algorithm 1 Key Search in ChaoW Framework

Input:
Set the initial random partial key: $\mathcal{K} = [\mathbb{L}, \{\tau^{(l)} | l \in \mathbb{L}\}]$
Pre-trained DNN model: F_θ
Validation dataset: D_{valid}
Expected accuracy loss: σ_e
Output:
Encrypted DNN model: $F_{E(\theta)}$
Fine-tuned secret key: $\mathcal{K}_{ft} = [\mathbb{L}, \{\tau^{(l)}, S^{(l)}_{opt} | l \in \mathbb{L}\}]$
1: **function** Encryption()
2: $\sigma \leftarrow 0, D_S^{(\mathbb{L})} \leftarrow \emptyset, D_{F_{E(\theta)}} \leftarrow \emptyset$
3: $C^{(\mathbb{L})}, N^{(\mathbb{L})} \leftarrow LayerShape(\mathbb{L})[:2]$
4: $D_{tmp}^{(\mathbb{L})} \leftarrow Sample\left(C^{(\mathbb{L})}, N^{(\mathbb{L})}\right)$
5: $Y_{F_\theta}^{pred} \leftarrow Inference(F_\theta, \quad D_{valid})$
6: **for all** $S^{(\mathbb{L})} \in D_{tmp}^{(\mathbb{L})}$ **do**
7: // Eqs. (1) and (4)
8: $F_{E(\theta)} \leftarrow Arnold(F_\theta, \ k = [\mathbb{L}, \{\tau^{(l)}, S^{(l)} | l \in \mathbb{L}\}])$
9: $Y_{F_{E(\theta)}}^{pred} \leftarrow Inference(F_{E(\theta)}, \quad D_{valid})$
10: $\sigma \leftarrow Y_{F_\theta}^{pred} - Y_{F_{E(\theta)}}^{pred}$
11: **if** $\sigma \geq \sigma_e$ **do**
12: $D_S^{(\mathbb{L})} \leftarrow D_S^{(\mathbb{L})} \cup \left\{S^{(\mathbb{L})}\right\}, D_{F_{E(\theta)}} \leftarrow D_{F_{E(\theta)}} \cup F_{E(\theta)}$
13: $S_{opt}^{(\mathbb{L})}, F_{E(\theta)} \leftarrow MinDecryptionTime(D_S^{(\mathbb{L})}, D_{F_{E(\theta)}})$
14: **return** $F_{E(\theta)}, \mathcal{K}_{ft}$

the weights, and the original positions can be quickly retrieved according to the secret keys. The memory overhead to store the secret keys can be ignored compared with conventional methods, which need store all modified weights and the related indexes.

3) Perfect Concealment: Unlike previous methods [3], [18], our method is sufficiently concealed and does not change the distribution of the parameters at all. This is achieved by utilizing an advanced chaotic encryption algorithm. Hence, malicious users cannot crack encrypted DNNs according to the statistics of the parameters.

4) Needless Retraining: Conventional methods require massive training images to retrain the well-trained models in order to get the effectiveness of the encrypted DNNs. Nevertheless, fine-tuning DNNs consumes lots of hardware resources and usually takes days or even weeks. Conversely, ChaoW only uses the well-trained DNNs in the open-sourced model zoo [29], [30] and requires a subset of the validation images.

C. Solution Details

We adopt Arnold's cat map (ACM) [31], [32] to encrypt or decrypt the weights of DNNs. ACM is a chaotic map which was first introduced by Vladimir Arnold in the 1960s. Basically, ACM encrypts an image by making chaos, i.e., by exchanging the positions of the pixels in a certain way. Though the principle looks very simple, the encrypted image does not have any visual feature—it looks like a television-static of chaos and the correlation among adjacent pixels is disturbed completely, achieving the same effect of conventional data encryption algorithms. We will show that ACM is much easier and faster to use, and has only a few functions (chaotic

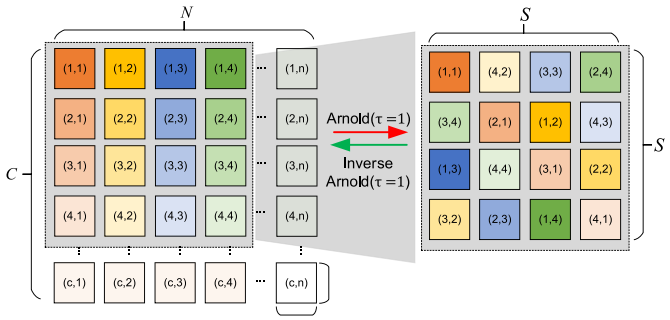


Fig. 4. Fast encryption and decryption for convolution kernels by ACM.

maps) and some parameters (secret keys) to be stored, but the attack complexity is still high.

Encryption: We use ACM to exchange the positions of convolutional or fully connected kernels of DNNs. Specifically, as shown in Figs. 2 and 4, the shape of a convolutional kernel is $C \times N \times k \times k$. Here, k is the size of the convolutional kernels, which is typically 3, 5, and 7. ChaoW can also directly be used for fully connected layers by setting $k = 1$. To adapt to both convolutional and fully connected layers, we change the positions of the weights in the first two dimensions. This 2-D convolutional kernel encryption by ACM is formulated as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A^\tau \begin{bmatrix} x \\ y \end{bmatrix} (\text{mod } S), \ A = \begin{bmatrix} 1 & p \\ q & pq+1 \end{bmatrix} \tag{1}$$

where p and q are integers, and S is the encryption size (the encryption range (ER) must be square, i.e., $S \times S$). The original position set of the convolution kernels is

$$\mathbb{I} = \{(x, y) | x = 1, 2, \ldots, S, y = 1, 2, \ldots, S\}$$

where S must satisfy both $S \leq C$ and $S \leq N$. After τ iterations, we can get a new position (x', y') for each weight parameter in the ER. Different layers have different encryption parameters (i.e., p, q, τ, S). Therefore, the encryption layer set $\mathbb{L}$ and the encryption parameters of the encrypted layers form the secret key, which is

$$\mathcal{K} = \left[\mathbb{L}, \left\{\tau^{(l)}, S^{(l)}, p^{(l)}, q^{(l)} | l \in \mathbb{L}\right\}\right] \tag{2}$$

where the superscript$^{(l)}$ is the layer index.

Decryption: When we get the secret key and the encrypted convolution kernels $\{(x', y')\}$, we can make use of the inverse ACM to decrypt the weights so as to retrieve the original positions of the convolution kernels. The decryption formula is

$$\begin{bmatrix} x \\ y \end{bmatrix} = A^{-\tau} \begin{bmatrix} x' \\ y' \end{bmatrix} (\text{mod } S). \tag{3}$$

Fast Encryption and Decryption: In order to speed up the encryption and decryption procedures, we set $p = 1$ and $q = 1$ in (2). In the following contents, p and q are removed from the secret key. As a result, the parameter A^τ of ACM in (1) can now be reformulated as

$$A^\tau = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}^\tau = \begin{bmatrix} f_{2\tau-1} & f_{2\tau} \\ f_{2\tau} & f_{2\tau+1} \end{bmatrix} \tag{4}$$

where f_τ is the Fibonacci number, i.e.,

$$f_1 = 1, \ f_2 = 1, f_{\tau+2} = f_{\tau+1} + f_\tau, \ \tau = 1, 2, \ldots \tag{5}$$

Similarly, the decryption parameter $A^{-\tau}$ of the inverse ACM in (3) can be reformulated as

$$A^{-\tau} = \begin{bmatrix} f_{2\tau-1} & -f_{2\tau} \\ -f_{2\tau} & f_{2\tau+1} \end{bmatrix}. \tag{6}$$

Finally, we can combine (4) and (6) with (1) and (3) to fast encrypt or decrypt the weights. The Fibonacci series has a general term formula so (4) and (6) can be calculated directly without any iterations. Specifically, Fig. 4 shows a fast encryption and decryption example when $\tau = 1$ and $S = 4$. We can find that the positions of are changed after encryption. For example, in this example, the first row, third column kernel—(1, 3) is moved to the third row, first column in encryption convolution kernels. The perturbed convolution kernels will affect the inference accuracy of DNNs, thus completing the encryption without modifying the weights values. Finally, the inverse ACM is used to restore the original positions of the convolutional or fully connected weights.

D. Secret Key Generation

We lay out our ChaoW framework to search for the optimal key in Algorithm 1. In the encryption phase, we first randomly set the initial partial key $\mathcal{K}$ which includes the encryption layer set $\mathbb{L}$ and the ACM parameter τ of each layer. To fast encrypt or decrypt DNNs, we set $p = 1$ and $q = 1$ in (1) and (3) so that p and q are omitted. The minimal expected encryption accuracy loss is σ_e, which equals the original accuracy minus the accuracy after encryption. To test the encryption accuracy, only a small portion of the validation dataset is used. Next, we sample the $S \times S$ dimensions from the $C \times N$ dimensions of each layer in $\mathbb{L}$, which are the first two dimensions of the weights of the layers in the layer set $\mathbb{L}$. Then, we use the fast encryption method in (4) to encrypt the DNN and save the desired accuracy loss of the encrypted DNN. Finally, we choose the encrypted model with the shortest decryption time, and then the optimal key is obtained.

During the key search process, we use a validation dataset to check if a candidate key satisfies the expected accuracy loss. The time to find a key is related to the time of accomplishing a validation procedure $T(Test)$. The larger the validation dataset, the longer the validation time required. For example, the ImageNet dataset contains 50 000 images with a 224×224 resolution, and its $T(Test)$ is about 20 min on an NVIDIA TITAN Xp GPU based on our experiments. $T(Test)$ of the smaller CIFAR-10 dataset is just about 3 min. In our experiments, usually no more than 5 validations, or even with only 1 validation, can already yield a satisfying secret key. This stems from that the encrypted layers and the ERs of the layers can be arbitrary in our solution, and more layers and larger ERs can produce a satisfying encryption effect.

E. Attack Analysis

Brute-Force Attack: There are only two known attack methods to ACM, which are known-plaintext and chosen-plaintext attacks [33], [34]. However, the two attack methods require either the corresponding plaintext weights or access to the encryption system to encrypt self-chosen weights, thus the chosen-plaintext attack and known-plaintext cannot be realized in the scenario of DNN IP protection for model providers, as the attacker only has the encrypted weights and the DNN architecture. As a result, the only possible algorithm-level attack method is exhaustive search (i.e., brute-force attack). According to the above analysis, since we can encrypt any layer and any range of a layer, it is expected that the complexity of a brute-force attack is extremely high. Therefore, a brute-force attack to crack the secret key in (2) tends to be impractical. For simplicity, we use $p = 1$ and $q = 1$ in (2). Then, the attack complexity will approach at least $O(2^{|\mathbb{L}|}) * O(C^2) * O(N^2) * O(\tau)$. Assume that τ is very small (say, 10). We further assume that the validation procedure is conducted at an extremely powerful computer like IBM *Summit* supercomputer [35] and it takes 1 s to conduct a validation procedure. This is a conservative estimation because in practice a validation procedure typically spends at least a few minutes. A simple estimation shows that cracking a 13-layer DNN encrypted by ChaoW takes about ten million years. Taking VGG-16 as an example, where the number of convolution layers is 13, and the average number of input and output channels of all convolution layers is 305, cracking our proposed ChaoW will consume at least $O(2^{13}) \times O(305^2) \times O(305^2) \times O(10) \cong 6.8 * 10^{14}$ s, which is about 21 million years.

Model Retraining or Fine-Tuning: According to the above analysis, it is almost impossible to obtain the key by a brute-force attack. Another possible attack method is to slightly retrain or fine-tune the encrypted model to obtain acceptable accuracy. A question is whether retraining or fine-tuning costs a lot. Our answer is that retraining or fine-tuning cannot be performed by attackers. Obviously, the three fundamental elements of DNNs are model architectures, computing resources (such as GPUs and TPUs), and training data, which are indispensable. Even if a malicious user has a model architecture and computing resources, an acceptable accurate model cannot be obtained without training data. There are three reasons for this. First, training data is very valuable as it costs a great deal of labor to collect and process. Second, privacy and confidentiality concerns legally prevent sharing training data (e.g., General Data Protection Regulation [36]). Last and most importantly, it is unnecessary to crack the encrypted model if a malicious user has the training data, since a malicious user already has the three fundamental elements of DNNs to obtain an accurate model by himself. Therefore, it is reasonable to assume that training data is not made public so that malicious users cannot retrain or fine-tune the encrypted DNNs.

Secret Key Leak: Under this attack model, valuable DNN models can be maliciously distributed to unauthorized users. However, how to protect keys from being leaked is beyond the scope of this article. It should be emphasized that modern cryptography is built on the base that the only secret is key (while encryption algorithms are completely public), knowns as the Kerckhoffs's Principle and Shannon's Maxim [37]. If the key is leaked, any encryption is ineffective. How to protect the key is another important topic that has been studied in other literature. Here, we just briefly provide some possible solutions. For example, people can use physical

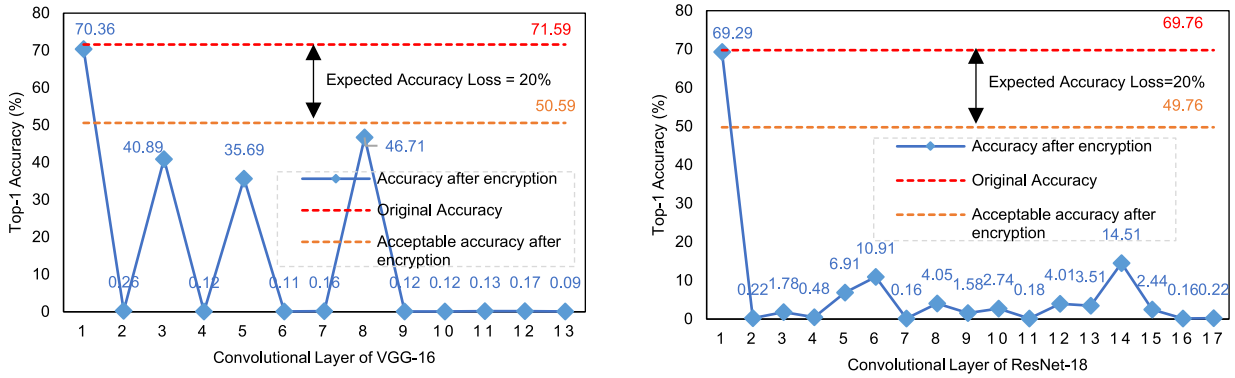


Fig. 5. Sensitivity of different layers in VGG-16 and ResNet-18 *versus* the layer indexes. Three types of validation accuracy present the original accuracy, the acceptable accuracy after encryption, and the accuracy after encryption, respectively.

unclonable functions (PUFs) [38] to generate per-chip keys, or use a secure memory [39]–[41] to store the key. In addition, like the conventional software authentication method, a model provider can prevent malicious users from using the DNN model illegally by checking users' identifiers (e.g., physical addresses).

Side-Channel Attack: Side-channel attack is an important approach to obtain confidential data (e.g., key). Recently there reports a side-channel attack approach [14] which reads data through the processor-memory interface to steal the DNN structure and weights. In DNN accelerators, typically layers are processed one by one and the weights of the current running layer are stored in the on-chip storage (e.g., buffers and registers). We only need to keep the decrypted weights on chip without writing them back to the memory, ChaoW can effectively defend against this attack [14] as the stolen weights are encrypted. This is feasible since we can read the encrypted weights from memory and then decrypt them on chip.

IV. Experimental Results

ChaoW is essentially applicable to a variety of DNN models which consist of convolutional and fully connected layers. In this section, we evaluate our framework on three challenging tasks, including two computer vision tasks: image classification and semantic segmentation, and one NLP task: name generation with a character-level recurrent neural network (RNN) on the GPU platform.

A. Experimental Setup

Our method is implemented using the PyTorch framework on a TITAN Xp GPU with CUDA9.0 and CuDNN7.1 backends. Several state-of-the-art DNNs, including VGG-16 [42], ResNet-18 and -101 [19], and GoogLeNet [43] are evaluated on the large-scale ImageNet ILSVRC-2012 dataset [28] with a 224×224 resolution for image classification; UNet [44] and LinkNet [45] are tested on the CamVid dataset [46] with a 480×360 resolution for semantic segmentation. A character-level RNN, which consists of three fully connected layers, is used to generate names. We implement the ACM using C++ to quickly encrypt or decrypt the DNN models, and all the decryption time of our method is tested on an Intel Xeon E5-2650 v4 CPUs and a TITAN Xp GPU.

B. Image Classification Application

To verify the availability of ChaoW, we first perform a sensitivity analysis on the key parameters in (2). It is critical to choose the suitable layer set $\mathbb{L}$, parameter τ, and the ER $S \times S$ of each layer to obtain a satisfying encryption effect.

1) Sensitivity Analysis of Encryption Layers: We first analyze the impact of different encryption layers on the accuracy after encryption on VGG-16 and ResNet-18 in Fig. 5. There are two conclusions can be drawn. First, the accuracy after encryption of the first layer of all the tested DNNs is the worst, which is higher than the acceptable accuracy after encryption. Specifically, the accuracy after encryption of the first layer of VGG-16 is 69.29%, which is only 1.23% smaller than the original accuracy. At the same time, 1.23% is far less than our expected accuracy loss (i.e., 20%). Therefore, if only one layer can be selected for encryption, we should not choose the first layer of VGG-16 for encryption. The reason for this phenomenon is that the first layer size of VGG-16 is $C \times N = 3 \times 64$, and the maximum range that can be encrypted is very small, which is $S \times S = 3 \times 3$. The smaller ER, the higher the accuracy after encryption. The same phenomenon also appears in ResNet-18. More details of the effect of the ER on accuracy are explained in Section IV-B3.

Second, except a few layers with higher accuracy after encryption, other layers can make the accuracy after encryption small enough to approach random guessing. For instance, when any layer of the set $\{2, 4, 6, 7, 9, 10, 11, 12, 13\}$ is encrypted, the accuracy after encryption of VGG-16 is not higher than 1%. The reason is that here we have encrypted the maximum range of a layer, that is, $S \times S = C \times N$. In contrast, when any layer in the set $\{3, 5, 8\}$ is encrypted, the accuracy after encryption of VGG-16 is relatively high but less than the acceptable accuracy after encryption (i.e., 50.59%). This high accuracy after encryption stems from that C of any layer in $\{3, 5, 8\}$ is not equal to N (C is half of N). Therefore, only half of the maximum range is encrypted, that is, $S \times S = C \times N/2$. For example, the size of the eighth layer of VGG-16 is $C \times N = 256 \times 512$, so $S \times S = 256 \times 256$.

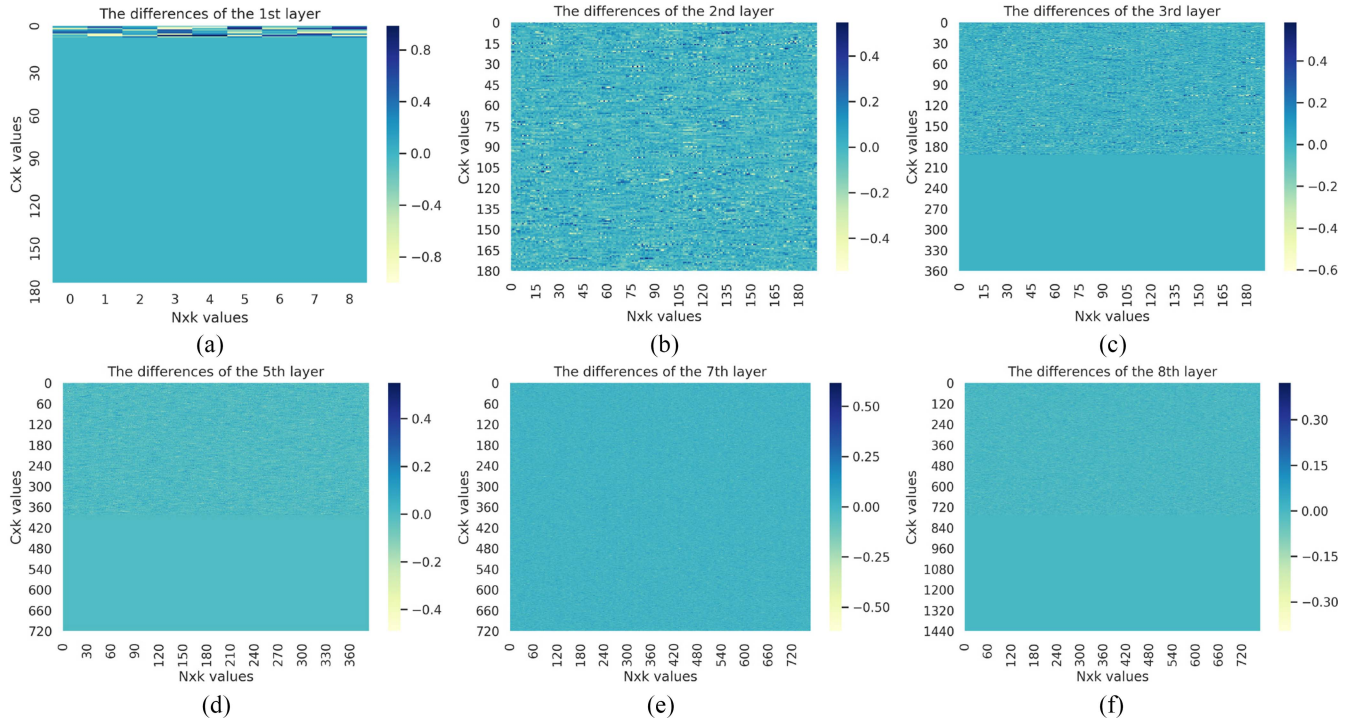


Fig. 6. Differences between original weight values and encrypted weights values of VGG-16. Subgraphs (a), (c), (d), and (f) are the differences of the first layer, the third layer, the fifth layer, and the eighth layer of VGG-16, respectively. Subgraphs (b) and (e) denote the differences of the second layer and the seventh layer of VGG-16, respectively.

To explain the reason for accuracy changes after encryption, we first expand the four-dimensional matrix $C \times N \times k \times k$ of the convolution kernel into 2-D matrix—$(C \times k, N \times k)$, and then calculate the difference at each position between the encrypted weight values and the original weight values of the first layer, the third layer, the fifth layer and the eighth layer of VGG-16 as shown in subgraphs (a), (c), (d), and (f) of Fig. 6. At the same time, to compare with those layers which lead to low accuracy loss after encryption, the second layer and the seventh layer are randomly selected as shown in subgraphs (b) and (e) in Fig. 6. It can be found through observation in Fig. 6(a) that most weight values of the first layer of VGG-16 are 0, which implies that the encryption process changes the weight values little, so the accuracy after encryption is not significantly decreased (i.e., -1.23%). In contrast, most of the weight differences of the second layer and the seventh layer are not 0 in Fig. 6(b) and (e), indicating that almost all the weight values are changed after encryption, either larger or smaller, thus causing a great loss of the original accuracy (i.e., -71.33% and -71.43%, respectively). Finally, in Fig. 6(c), (d), and (f), half of the weight differences are 0, which means that only half of the weight values are changed, thus the accuracy after encryption of these layers are between the accuracies after encryption of Fig. 6(a) and (b) and (e).

In summary, except for that the first layer of a DNN does not need to be encrypted, ChaoW can work well on other layers for encryption. In most cases, only encrypting one layer of DNNs can make the accuracy after encryption sufficiently low, which is obviously better than the recently proposed parameter encryption approach [18], which requires to encrypt more than 20 layers of a DNN.

TABLE II
EXPERIMENT SETUP

Network	Layer name[1]	$S \times S$
VGG-16	feature[21]	512×512
ResNet-18	layer4[1].conv2	512×512
ResNet-101	layer4[0].conv2	512×512
GoogLeNet	inception3a.branch1.conv	64×64

[1]. All models are imported from the *torch.vision.models* of PyTorch, and the layer name is consistent with that of *torch.vision.models*.

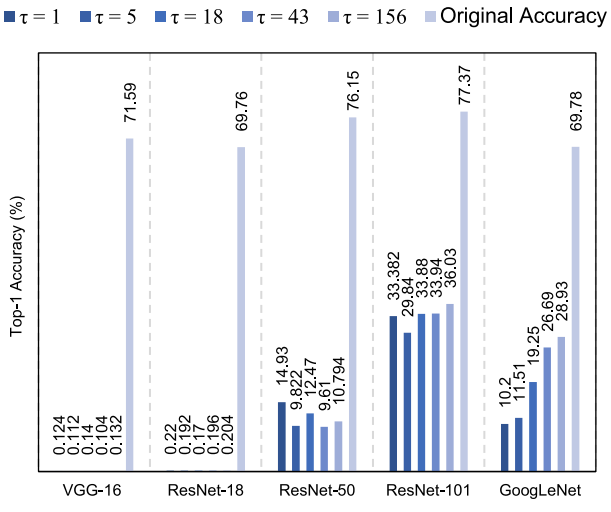


Fig. 7. Top-1 accuracy *versus* parameter τ (1, 5, 18, 43, 156) on four DNNs.

2) Sensitivity Analysis of τ: We randomly select the layers of four state-of-the-art DNN models and fix the ER $S \times S$ in Table II. To illustrate the effect of τ to the encrypted model, we randomly set τ to 1, 5, 18, 43, and 156, respectively. In Fig. 7,

TABLE III
SENSITIVITY ANALYSIS OF S FOR RESNET-18, AND WE SET τ TO 43 AND σ_e TO 20%. THE ORIGINAL TOP-1 ACCURACY OF RESNET-18 IS 69.76%. N/A MEANS THAT THE ER IS BEYOND THE LAYER SHAPE N, C. THE DECRYPTION TIME IS TESTED ON AN INTEL XEON E5-2650 V4 CPU. BOLD NUMBERS STAND FOR OPTIMAL ENCRYPTION ACCURACY ACCORDING TO ALGORITHM 1

$S \times S$	Decryption time (ms)	Top-1 accuracy after encryption (%) of ChaoW			
		layer4[1].conv2	*layer3[1].conv1*	*layer2[1].conv1*	*layer1[0].conv1*
		Layer shape: 512×512	Layer shape: 256×256	Layer shape: 128×128	Layer shape: 64×64
512x512	8.56	0.196	N/A	N/A	N/A
448x448	6.57	11.368	N/A	N/A	N/A
384x384	4.88	**35.198**	N/A	N/A	N/A
320x320	3.35	50.318	N/A	N/A	N/A
256x256	2.19	61.494	7.73	N/A	N/A
192x192	1.21	66.426	**26.668**	N/A	N/A
128x128	0.56	68.33	58.43	7.66	N/A
96x96	0.31	68.936	65.23	**33.882**	N/A
64x64	0.14	69.36	68.488	64.606	0.53
48x48	0.08	69.474	69.09	64.912	2.34
32x32	0.04	69.63	69.554	68.58	**26.2**

we can observe a very important conclusion that the parameter τ has a significant encryption effect on all DNNs, no matter how parameter τ changes. Specifically, for shallow DNN models, e.g., VGG-16 or ResNet-18, the top-1 accuracy after encryption is very low, which is the same as random guess probability from 1000 classes (i.e., 0.1%); and for deep DNN models, such as ResNet-101 and GoogLeNet, the encryption effect is also obvious, showing at least 40% accuracy drop. In addition, recall that we use (4) and (6) to quickly calculate the parameters $A^{-\tau}$ and A^{τ}, thus τ does not affect the encryption or decryption time. Therefore, the change of τ does not affect the validity of encryption and we can directly specify τ in practice.

3) Sensitivity Analysis of Encryption Range: In Table III, ResNet-18 is experimented to illustrate how the change of the ER $S \times S$ affects the encryption and decryption time. Parameter τ is randomly set to 43. An important conclusion has been made that the effect of encryption is degraded as S is decreased. For instance, *layer*4[1].*con*2 is a deep layer of ResNet-18 and the first two dimensions' shape is 512×512. If we encrypt all weights of this layer, i.e., $S \times S = 512 \times 512$, the encryption effect is the best where the top-1 accuracy of encrypted ResNet-18 is 0.196%. When we encrypt a quarter of the weights (i.e., $S \times S = 256 \times 256$), the accuracy of the encrypted model rises to 61.494%, which implies a poor encryption effect. The similar experimental phenomenon also appears in *layer*3[1].*conv*1, *layer*2[1].*conv*1, and *layer*1[0].*conv*1 of ResNet-18. For this reason, to get a satisfying encryption effect, a large ER should be selected.

Another conclusion is that the larger ER $S \times S$, the longer decryption time it takes, and a smaller ER should be selected to reduce the decryption time. Therefore, we use the proposed algorithm 1 to get a suitable ER $S \times S$ to achieve the above two goals at the same time. Specifically, if we set the expected accuracy loss σ_e to 20%, which means that the top-1 accuracy of encrypted ResNet-18 should be less than 49.76% (original: 69.76%). In Table III, if we encrypt *layer4*[1].*conv2*, the ER that can be selected is 384×384, 448×448, and 512×512 according to the top-1 accuracy. The decryption time is the shortest -4.88 ms when $S \times S$ is set to 384×384. So, according to (2), the applicable key of *layer*4[1].*conv*2 is $\mathcal{K} = [layer4[1].conv2, \{43, 384\}]$. To simplify, we express it as $\mathcal{K} = [4.1.2, \{43, 384\}]$.

4) Speed and Accuracy: Fig. 8 shows the inference speed and accuracy comparisons using different keys on four DNN models. We set the expected accuracy loss σ_e to 20%, which can generate satisfying encryption effect. As expected, all encrypted models meet the expected accuracy loss requirement regardless of how the key changes. However, different keys have a large impact on the inference speed. For instance, the inference speed of key-1 for VGG-16 is 208.33 fps [Fig. 8 (a)], which is almost equal to the original inference speed 211.86 fps. Whereas the inference of key-4 is 104.17 fps, and the speed overhead accounts for nearly half of the original speed. The fundamental reason is that the deeper the layer, the more difficulty the encryption. In Table III, when the ER $S \times S$ is set to 96×96, the accuracy after encryption of *layer*2[1].*conv*1 is 33.88%, which is 35.06% better than *layer*4[1].*conv*2. When the ER is increased to 384×384, the encryption effect of *layer*4[1].*conv*2 can be close to that of *layer*2[1].*conv*1. Thus, a large ER is needed to encrypt deep layers of DNN models, but a large ER also increases the overhead of decryption. Therefore, we can encrypt shallow layers of DNN models for fast decryption in practice. In conclusion, our method is still valid regardless of the key, as the speed of all image classification models is faster than the real-time inference speed requirement, which is typically 30 fps.

To illustrate the impact of multilayer encryption on the inference and accuracy after encryption, we encrypt two layers and three layers of GoogLeNet as shown in Fig. 8(e) and (f). Two conclusions can be drawn.

① The more encryption layers in DNNs, the lower the accuracy after encryption. Specifically, the accuracy after encryption of key-3 in Fig. 8(d) is 38.27%, which is 31.51% smaller than the original accuracy (i.e., 69.78%). When the number of encrypted layers is increased by one, as shown in Fig. 8(e), the accuracy after encryption of key-3 is decreased to 4.76% when the accuracy after encryption is low enough to meet the encryption requirements. When the number of encrypted layers is

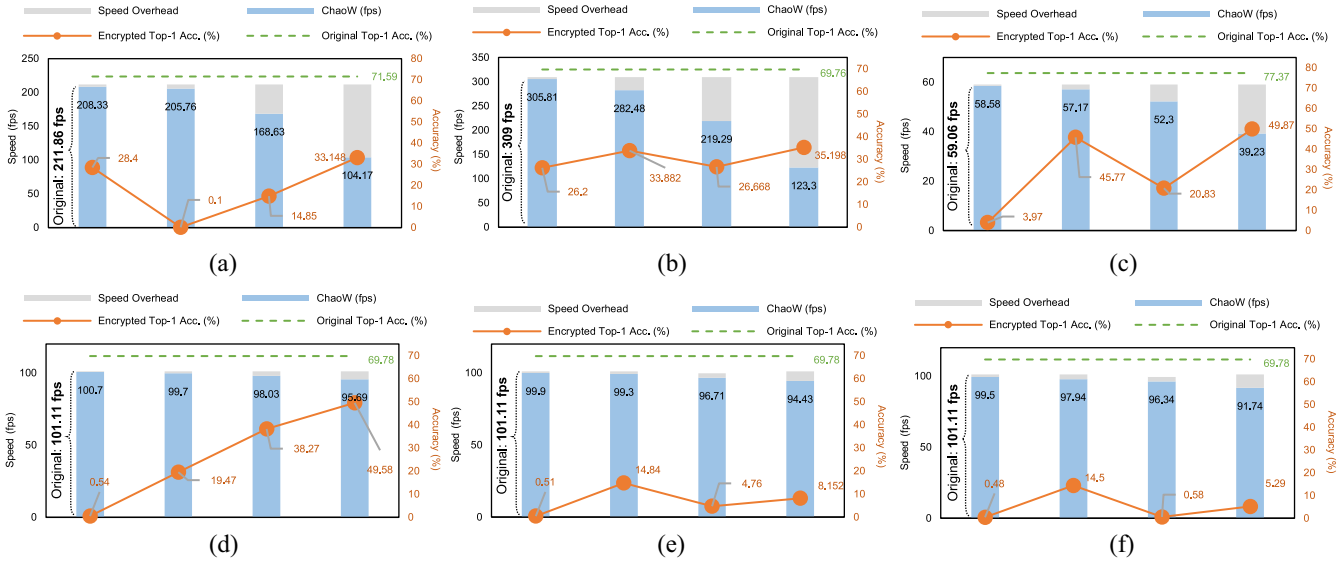


Fig. 8. Inference speed (fps) and top-1 accuracy (%) of four DNNs, and inference speed is tested on a TITAN Xp GPU. We set σ_e as 20% and the format of key is expressed according to (2) in Section II-C. Taking key-1 of VGG-16 as an example, first, the encryption layer we randomly choose is *feature*[2].*conv* and set τ to 5. Then, S is calculated as 48 according to Algorithm 1. Finally, key-1 of VGG-16 is $\mathcal{K} = [feature[2].conv, \{5, 48\}]$. We write it as $\mathcal{K} = [2, \{5, 48\}]$. (a) VGG-16 ($|\mathbb{L}| = 1$). (b) ResNet-18 ($|\mathbb{L}| = 1$). (c) ResNet-101 ($|\mathbb{L}| = 1$). (d) GoogLeNet ($|\mathbb{L}| = 1$). (e) GoogLeNet ($|\mathbb{L}| = 2$). (f) GoogLeNet ($|\mathbb{L}| = 3$).

increased to 3, the accuracy after encryption of key-3 is decreased to 0.58%, which is approximately equal to the probability of 1000-class random guess (i.e., 0.1%).

② Although the number of encrypted layers is increased, the practical inference speed of DNNs is hardly changed, and the decryption takes far less time than the inference of DNNs. Taking key-3 as an example, the inference speed of the two-layer and three-layer encryption in Fig. 8(e) and (f) is only 1.32 and 1.69 frames/s, respectively, less than inference speed of the one-layer encryption in Fig. 8(d). This low-speed overhead stems from our proposed Algorithm 1, which chooses an ER $S \times S$ corresponding to the minimum decryption time of each layer as a part as the key-3.

C. Semantic Segmentation Application

The CamVid dataset [46] is a good standard for the verification of semantic segmentation of urban streets. This dataset contains 701 images extracted from high-resolution video sequences, which is divided into 367 100 and 233 images for training, validation and testing, and each pixel is annotated to one of 11 semantic classes. We evaluate four state-of-the-art segmentation models of open-sourced cars segmentation experiments [47] to verify the effectiveness of our method. For comparison, the commonly employed metric—mean intersection-over-union (mIoU) is used to calculate the accuracy results, and all results of the inference speed are evaluated on a TITAN Xp GPU. The expected accuracy loss σ_e is 20%, which is consistent with the setup in image classification.

As Table IV implies, ChaoW still obtains satisfying encryption accuracy and inference speed for four segmentation models. When the models are decrypted, the inference speed meets the need of real-time speed (i.e., 30 fps). Specifically, when the secret key $\mathcal{K}$ is [2,{3,48}], the encrypted mIoU (E.mIoU) of ChaoW is 40.96%, which is 33.53% lower than that of UNet-VGG16. And the inference speed is 90.25 fps, which is almost the same as the original UNet-VGG16 speed of 90.9 fps. The similar experimental results also appear in UNet-ResNet34, LinkNet-VGG16, and LinkNet-ResNet34.

In addition, to compare the effect of encryption accuracy on segmentation models, we demonstrate the input images, ground truth labels, the predictions of original models, and the predictions of our proposed ChaoW models of UNet-VGG16 in Fig. 9. The comparisons are agreed with the secret keys of UNet-VGG16 in Table IV. Two conclusions can be drawn from the figure.

1) Compared with the original models, the predictions of ChaoW differ greatly from the ground truth. For instance, the E.mIoU of $\mathcal{K} = [2, \{3, 48\}]$ produces 40.96% accuracy and the prediction of ChaoW loses the far car in Fig. 9(a). To make matters worse, when $\mathcal{K} = [14, \{4, 192\}]$ and the accuracy after encryption is 4.48%, no car can be predicted in Fig. 9(c). This is what we expect—even if the segmentation model is stolen by a malicious user, it will not work normally.
2) The lower the accuracy of the encryption, the worse the prediction will be, which can be obtained by observing the predictions of ChaoW at the last column of Fig. 9. Therefore, our ChaoW framework is able to protect the IP of the semantic segmentation models.

D. Generating Names With Character-Level RNN

Because our framework can directly be applied to fully connected layers by setting $k = 1$, RNN or long short-term memory (LSTM), which is principally comprised of fully connected layers, frequently used in NLP tasks, can also be protected by ChaoW. Here, we use an open-sourced RNN consisting of three fully connected layers [48] to illustrate the

TABLE IV
SEGMENTATION EXPERIMENT. THE INFERENCE SPEED IS TESTED ON A TITAN XP GPU. THE KEY FORMAT IS SIMPLIFIED AS ONLY ONE LAYER IS ENCRYPTED IN THESE CASES

Original Models	ChaoW			Original Models	ChaoW		
	$\mathcal{K} = [l, \{\tau, S\}]$	E.mIoU(%)	Speed(fps)		$\mathcal{K} = [l, \{\tau, S\}]$	E.mIoU(%)	Speed(fps)
UNet-VGG16 mIoU: 74.49 % Speed: 90.9 fps	[2, {3,48}]	40.96	90.25	**LinkNet-VGG16** mIoU: 76.42% Speed: 83.33 fps	[2, {9,48}]	42.59	82.78
	[7, {15,96}]	13.82	88.42		[7, {3,96}]	10.52	81.23
	[14, {4,192}]	4.48	75.82		[12, {16,192}]	1.84	70.47
	[24, {6,448}]	50.52	56.91		[28, {8,448}]	38.44	53.85
UNet-ResNet34 mIoU: 78.36% Speed: 111 fps	[1.0.1, {2,32}]	32.02	110.6	**LinkNet-ResNet34** mIoU: 72.37% Speed: 111 fps	[1.1.1, {3,48}]	33.14	110.13
	[2.1.1, {7,128}]	38.03	104.6		[2.1.1, {3,128}]	32.92	104.6
	[3.0.2, {11,256}]	10.21	89.37		[2.0.2, {7,128}]	5.45	104.6
	[4.2.1, {9,448}]	44.23	56.95		[3.0.1, {11,256}]	6.05	89.37

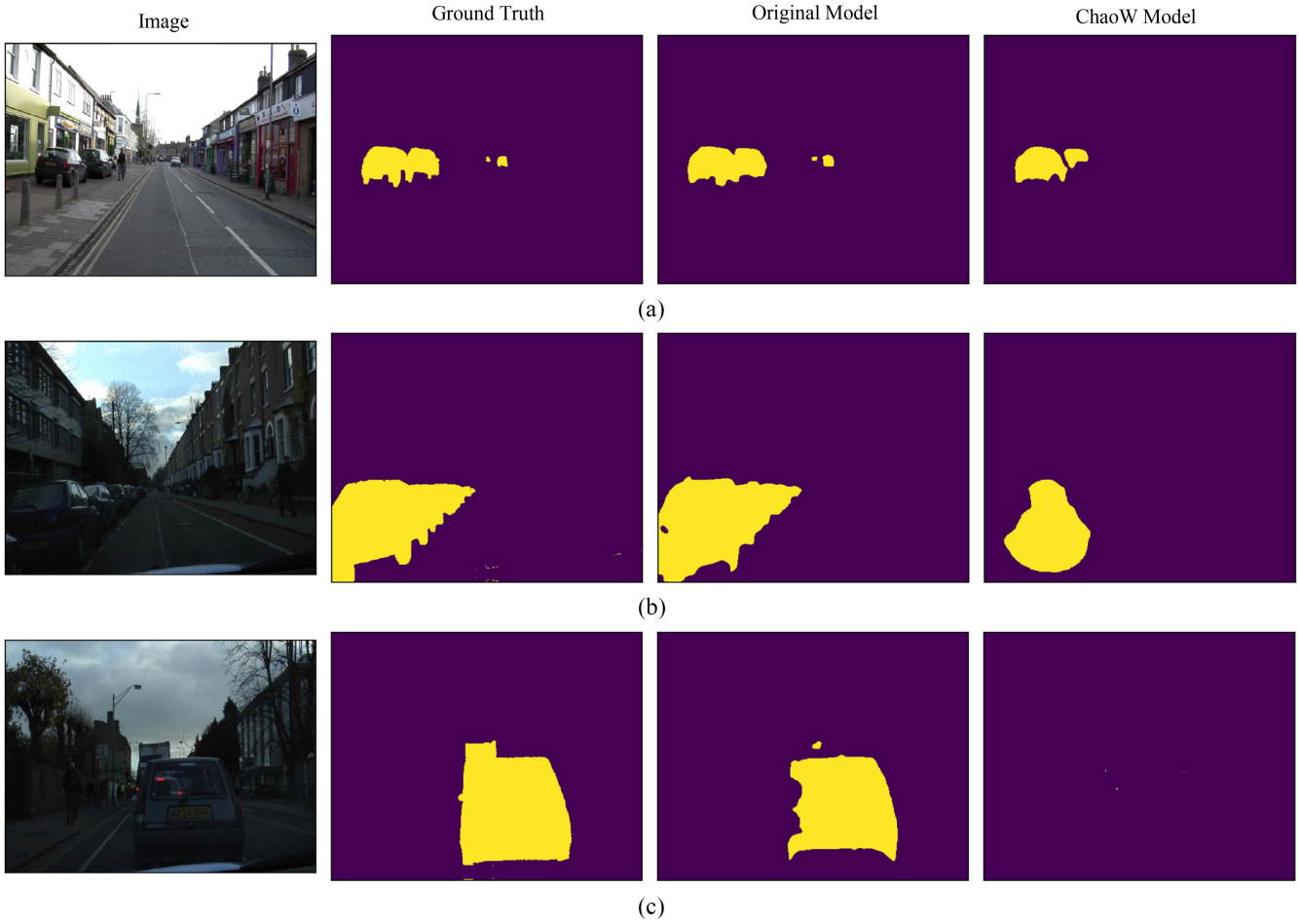


Fig. 9. Prediction comparison between ground truth, original model, and ChaoW model of UNet-VGG16. (a) Secret key of ChaoW is $\mathcal{K} = [2,\{3, 48\}]$. (b) Secret key of ChaoW is $\mathcal{K} = [7,\{15, 96\}]$. (c) Secret key of ChaoW is $\mathcal{K} = [14,\{4, 192\}]$.

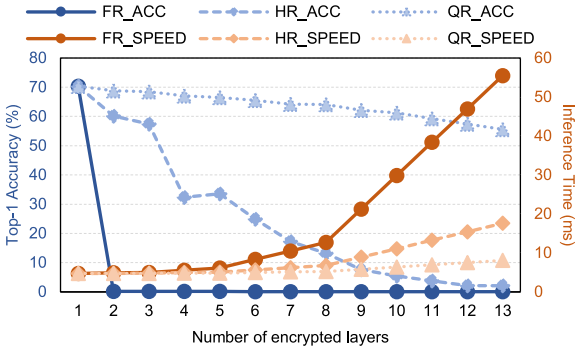
effectiveness of ChaoW in Table V. Compared with the original RNN, the encrypted RNN by ChaoW can make the names generation results far from the original ones. Specifically, when only FC-1 is encrypted, and the ER is 128×128, the generated names from the encrypted RNN have no meaning at all. This stems from that the ER is large and the accuracy of the encrypted RNN is reduced. When the ER of FC-1 gradually decreases, the number of characters predicted by the encrypted RNN tends to be consistent with the original RNN, but the correct result cannot be completely restored. In addition, since the range that can be encrypted of FC-2 or FC-3 is relatively small, the encryption effect is weakened but the results are still incorrect. As a summary, ChaoW can protect the IP of NLP tasks due to the ability to encrypt fully connected layers.

E. Tradeoff Between Speed and Accuracy

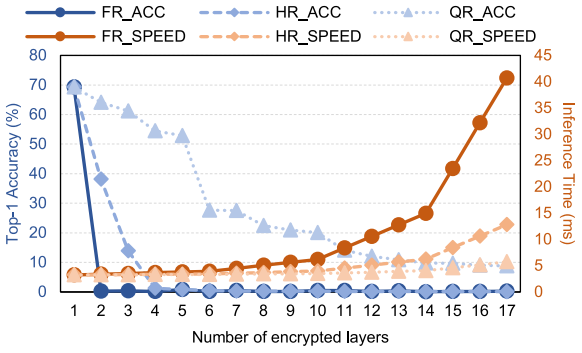
In fact, the ChaoW framework is able to obtain a satisfying accuracy after encryption if only the selected layers are encrypted, and the corresponding secret key in (2) is calculated based on Algorithm 1. Due to the satisfying accuracy after encryption, the number of layers to be encrypted is small, so the decryption time in the inference of DNNs

TABLE V
EFFECTIVENESS OF CHAOW FOR A CHARACTER-LEVEL RNN, WHICH CONSISTS OF THREE FULLY CONNECTED LAYERS, NAMED FC-1, FC-2, AND FC-3, RESPECTIVELY

Language	Correct name	Name generated from ChaoW without decryption						
		FC-1 (C×N=205×128)					FC-2 (C×N=59×205)	FC-3 (C×N=59×187)
		ER: 128×128	ER: 120×120	ER: 110×110	ER:100×100	ER:90×90	ER:59×59	ER:59×59
Russian	**R**oulov **U**akinovev **S**hanakov	Roaaaiaiaiaiaiaiaiaia Uaoaiaiaiaiaiaiaieaoo Saaaaaiaiaiaiaaaaaiai	Raiea Uaieieaititaieititiit Saaneieititititititil	Roureine Uairese Saureerene	Rointen Uonininin Sainin	Rousa Uauere Saiena	Romano Uonev Shalavak	Rouiti Uanov Santini
German	**G**anger **E**eren **R**oune	Goaieiarieieraiaraiai Eaaiieiaraiaraieraiar Raieaieieiarieiaiaai	Gaises Eaitiss Raiss	Goueirane Earerieieieereereana Raicerierieereieaner	Goone Eaeree Raiere	Ganere Eaurr Rourere	Gaen Eeren Roune	Geren Eereng Roumer
Spanish	**S**antana **P**arez **A**llan	Saaiaieiaraiaraiaiaia Paaiaiaiaiaiaiaiaiaia Aaaiaiaiaiaiaiaiaiaia	Saaiaieiaieiitiieiili Paeeaieieia Aaueieaieieiaieieiii	Saieeereereieereerane Paaiae Aaieereereereieereere	Sanere Paana Aaare	Soura Paireraie Aatea	Salas Pallas Allana	Salaza Peraza Areza
Chinese	**C**han **H**an **I**un	Caaaaeeieianaieiaiaia Haaaieiaiaiaianaier Iaaaaieianaieraaaaeea	Cauina Hainieiitiiitiitiitii Iaianieiitieiitiitii	Caina Haina Iaina	Chane Haie Iueie	Cain Haing Iuon	Chan Hang Iue	Chang Hang Iang



(a)



(b)

Fig. 10. Tradeoff between speed and accuracy after encryption. The inference time is measured on a NVIDIA TITAN Xp GPU. "ACC" represents the accuracy after encryption, and "SPEED" denotes the inference time. (a) VGG-16. (b) ResNet-18.

is little affected. Here, we analyze the tradeoff between the inference speed and the accuracy after encryption when the secret key is fixed. Fig. 10 explains this tradeoff on VGG-16 and ResNet-18 with three types of secret keys, which means the ER $S \times S$ is full range (FR) (i.e., $S \times S = C \times N$), half range (HR) (i.e., $S \times S = C/2 \times N/2$), and quarter range (QR) (i.e., $S \times S = C/4 \times N/4$) of each layer, respectively. **FR_ACC** denotes the accuracy after encryption of FR encryption, and **FR_SPEED** represents the inference time consumed by FR encryption. **HR_ACC**, **HR_SPEED**, **QR_ACC**, and **QR_SPEED** have the similar meanings for HR and QR encryptions.

TABLE VI
EFFECTIVENESS OF CHAOW FOR LENET NETWORK. THE ORIGINAL ACCURACY OF LENET ON CIFAR-10 IS 70.82%

Layer	Shape	Encryption range	Accuracy after encryption (%)
Conv-1	3×6×5×5	3×3	58.84
Conv-2	6×16×5×5	6×6	39.29
FC-1	400×120	120×120	39.97
FC-2	120×84	84×84	35.92
FC-3	84×10	10×10	66.85

As we expect, the accuracy after encryption decreases as the number of layers increases. Although the accuracy of some layers is slightly increased, such as the accuracy after encryption of **HR_ACC** when 5 layers are encrypted, as shown in Fig. 10(a), there is no rebound to the original accuracy. The inference time becomes longer as the number of encrypted layers increases, and it increases faster as it goes to the back of DNNs as shown in Fig. 10(a). This is because the number of channels in deep layers is large. For example, there are 512 channels in the eighth layer to the thirteenth layer of VGG16, so the decryption overhead will be larger.

In addition, compared with **HR_ACC** and **QR_ACC**, the accuracy after encryption of **FR_ACC** decreases faster. For instance, the **FR_ACC** of ResNet-18 is 0.23% when the number of encrypted layer is 2. And when the number of encrypted layers is 4, the accuracy after encryption of **HR_ACC** is 1.21%. When all layers are encrypted, the accuracy after encryption of **QR_ACC** drops to 8.8%. The reason for this phenomenon is that the ER of **FR_ACC** is the largest, and it encrypts all the weights of the second layer of VGG-16 and ResNet-18.

F. Effectiveness on Small Networks: Example of LeNet-5

In fact, the DNN models in the commercial field that are worth protecting IP are often deep models with a huge amount of parameters. However, for small networks, ChaoW is also effective. Here, we evaluate the effectiveness of ChaoW on LeNet-5 [27] which has only five layers, including two convolutional layers (Conv-1 and Conv-2) and three fully connected layers (FC-1, FC-2, and FC-3) on the CIFAR-10 dataset [5]. We show the encryption result of each single layer of LeNet-5 in Table VI. Regardless of whether it is a convolutional layer or

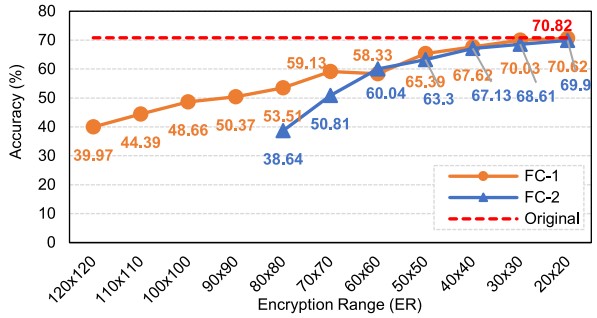


Fig. 11. Impact of different ERs for FC-1 and FC-2 of LeNet.

a fully connected layer, ChaoW can achieve a good encryption effect. For example, the accuracy after encryption of FC-1 is 39.97%, which is 30.85% lower than the original accuracy (70.82%). In addition, in Fig. 11, we show the influence of different ERs on the inference accuracy, which is consistent with the previous conclusion (see Section IV-B). The larger ER, the better the encryption effect. Therefore, ChaoW can also achieve good encryption for shallow DNN models.

V. Conclusion

In this article, we introduce a novel DNN IP protection framework ChaoW, which can economize the hardware resources. Different from traditional encryption approaches, our method encrypts or decrypts the weights by making weights chaotic, i.e., by exchanging the positions of weights via utilizing chaotic encryption algorithms. Therefore, our method is applicable to all DNNs which are made up of convolutional and fully connected layers, targeting at various tasks including but not limited to image classification, semantic segmentation, and names generation. To protect the IP of DNN models all the time, our method decrypts the encrypted parameters during the inference procedure. Extensive experimental evaluations demonstrate that the ChaoW framework can protect the IP of the model provider efficiently without damaging the original accuracy, and the impact on the inference speed is negligible.

References

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805.

[2] *Single Cloud TPU Device Pricing*. Accessed: Feb. 2, 2020. [Online]. Available: https://cloud.google.com/tpu/

[3] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proc. ACM Int. Conf. Multimedia Retrieval* 2017, pp. 269–277, doi: 10.1145/3078971.3078974.

[4] Y. Nagai, Y. Uchida, S. Sakazawa, and S. Satoh, "Digital watermarking for deep neural networks," *Int. J. Multimedia Inf. Retrieval*, vol. 7, no. 1, pp. 3–16, 2018, doi: 10.1007/s13735-018-0147-1.

[5] A. Krizhevsky, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, USA, Rep., 2009.

[6] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *Proc. 27th USENIX Security Symp. (USENIX Security)*, 2018, pp. 1615–1631.

[7] J. Zhang *et al.*, "Protecting intellectual property of deep neural networks with watermarking," in *Proc. Asia Conf. Comput. Commun. Security*, 2018, pp. 159–172.

[8] J. Guo and M. Potkonjak, "Watermarking deep neural networks for embedded systems," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Diego, CA, USA, 2018, pp. 1–8.

[9] B. Darvish Rouhani, H. Chen, and F. Koushanfar, "DeepSigns: An end-to-end watermarking framework for ownership protection of deep neural networks," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2019, pp. 485–497.

[10] E. Le Merrer, P. Perez, and G. Trédan, "Adversarial frontier stitching for remote neural network watermarking," *Neural Comput. Appl.*, vol. 32, pp. 1–12, Aug. 2019.

[11] B. Chen *et al.*, "Detecting backdoor attacks on deep neural networks by activation clustering," 2018. [Online]. Available: arXiv:1811.03728.

[12] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "STRIP: A defence against trojan attacks on deep neural networks," in *Proc. 35th Annu. Comput. Security Appl. Conf.*, 2019, pp. 113–125.

[13] R. Namba and J. Sakuma, "Robust watermarking of neural network with exponential weighting," in *Proc. ACM Asia Conf. Comput. Commun. Security*, 2019, pp. 228–240.

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

[15] X. Hu et al., "DeepSniffer: A DNN model extraction framework based on learning architectural hints," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, New York, NY, USA, 2020, pp. 385–399, doi: 10.1145/3373376.3378460.

[16] X. Hu *et al.*, "Practical attacks on deep neural networks by memory trojaning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, May 19, 2020, doi: 10.1109/TCAD.2020.2995347.

[17] Y. Zhao *et al.*, "Memory trojan attack on neural network accelerators," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Florence, Italy, Mar. 2019, pp. 1415–1420, doi: 10.23919/DATE.2019.8715027.

[18] Y. Cai, X. Chen, L. Tian, Y. Wang, and H. Yang, "Enabling secure in-memory neural network computing by sparse fast gradient encryption," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Westminster, CO, USA, 2019, pp. 1–8.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.

[20] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014. [Online]. Available: arXiv:1412.6572.

[21] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Security Symp. (USENIX Security)*, 2018, pp. 1651–1669.

[22] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Proc. 28th USENIX Security Symp. (USENIX Security)*, 2019, pp. 1501–1518.

[23] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 619–631.

[24] *AI and Machine Learning Products*. Accessed: Feb. 2, 2020. [Online]. Available: https://cloud.google.com/products/ai

[25] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances Neural Information Processing Systems*. Vancouver, BC, Canada: Curran Assoc., Inc., 2019, pp. 8024–8035.

[26] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Miami, FL, USA, 2009, pp. 248–255.

[29] *TorchVison Pretrained Models*. [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html

[30] Y. Jia and E. Shelhamer, *Caffe Model Zoo*, UC Berkeley, Berkeley, CA, USA, 2015.

[31] V. I. Arnol'd and A. Avez, *Ergodic Problems of Classical Mechanics*. New York, NY, USA: W.A. Benjamin, Inc., 1968.

[32] G. Peterson, "Arnold's cat map," in *Math Linear Algebra*, vol. 45. Upper Saddle River, NJ, USA: Prentice-Hall, 1997, pp. 1–7.

[33] I. E. Hanouti, H. E. Fadili, and K. Zenkouar, "Breaking an image encryption scheme based on Arnold map and Lucas series," 2019. [Online]. Available: arXiv:1910.11678.

[34] C. Cokal and E. Solak, "Cryptanalysis of a chaos-based image encryption algorithm," *Phys. Lett. A*, vol. 373, no. 15, pp. 1357–1360, 2009.
[35] *June 2019 TOP500 Supercomputer Sites*. Accessed: Sep. 23, 2019. [Online]. Available: https://www.top500.org/lists/2019/06/
[36] *General Data Protection Regulation*. Accessed: Feb. 25, 2020. [Online]. Available: https://gdpr-info.eu/
[37] C. E. Shannon, "Communication theory of secrecy systems," *Bell Syst. Techn. J.*, vol. 28, no. 4, pp. 656–715, Oct. 1949.
[38] C. Böhm and M. Hofer, *Physical Unclonable Functions in Theory and Practice*. New York, NY, USA: Springer, 2012.
[39] A. Seznec, "A phase change memory as a secure main memory," *IEEE Comput. Archit. Lett.*, vol. 9, no. 1, pp. 5–8, Jan. 2010.
[40] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, and M. K. Qureshi, "SYNERGY: Rethinking secure-memory design for error-correcting memories," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Vienna, Austria, 2018, pp. 454–465.
[41] R. Xin, Y. Li, Y. Liu, and Y. Zhao, "System design scheme of configurable secure memory chip," in *Proc. 9th Int. Conf. Intell. Hum. Mach. Syst. Cybern. (IHMSC)*, vol. 2. Hangzhou, China, 2017, pp. 280–283.
[42] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: arXiv:1409.1556.
[43] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1–9.
[44] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Int. Conf. Med. Image Comput. Comput. Assist. Intervention*, 2015, pp. 234–241.
[45] A. Chaurasia and E. Culurciello, "LinkNet: Exploiting encoder representations for efficient semantic segmentation," in *Proc. IEEE Vis. Commun. Image Process. (VCIP)*, St. Petersburg, FL, USA, 2017, pp. 1–4.
[46] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *Proc. Eur. Conf. Comput. Vis.*, 2008, pp. 44–57.
[47] *Segmentation Models: Car Segmentation Experiments*. Accessed: Feb. 25, 2020. [Online]. Available: https://github.com/qubvel/segmentation_models.pytorch
[48] *NLP From Scratch: Generating Names With a Character-Level RNN*. Accessed: Jun. 28, 2020. [Online]. Available: https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.htm

Ning Lin (Graduate Student Member, IEEE) received the B.S. degree in microelectronics from Xiangtan University, Xiangtan, China, in 2016. He is currently pursuing the Ph.D. degree with the State Key Laboratory. of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

His current research interests include efficient deep learning algorithms, security of DNNs, neural networks, and hardware accelerator co-design.

Xiaoming Chen (Member, IEEE) received the B.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2009 and 2014, respectively.

He is currently an Associate Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His current research interests include computer-aided design for integrated circuits and advanced computer architecture design.

Dr. Chen was a recipient of the 2015 European Design and Automation Association Outstanding Dissertation Award and the 2018 DAMO Academy Young Fellow Award. He served on the Organization Committee of Asia and South Pacific Design Automation Conference (ASP-DAC) 2020 and also served on the Technical Program Committees of Design Automation Conference (DAC) 2020, ACM Great Lakes Symposium on VLSI (GLSVLSI) 2020, International Conference On Computer-Aided Design 2019 & 2020, ASP-DAC 2019, International Conference on VLSI Design 2019 & 2020, Asian Hardware Oriented Security and Trust Symposium (AsianHOST) 2018–2020, and IEEE Computer Society Annual Symposium on VLSI (ISVLSI) 2018 & 2019.

Hang Lu (Member, IEEE) received the Ph.D. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2015.

He is currently an Associate Professor with the State Key Laboratory of Computer Architecture (CARCH), Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His research interests include high performance and power efficient networks-on-chip architectures, many-core processors, and domain-specific accelerators.

Xiaowei Li (Senior Member, IEEE) received the B.Eng. and M.Eng. degrees in computer science from the Hefei University of Technology, Hefei, China, in 1985 and 1988, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China, in 1991.

He was an Associate Professor with the Department of Computer Science and Technology, Peking University, Beijing, from 1991 to 2000. In 2000, he joined ICT, CAS, as a Professor, where he is currently the Deputy Director of the State Key Laboratory of Computer Architecture. He has coauthored over 280 papers in journals and international conferences, and he holds 60 patents and 30 software copyrights. His current research interests include VLSI testing, design for testability, design verification, dependable computing, and wireless sensor networks.

Dr. Li has been the Vice Chair of the IEEE Asia & Pacific Regional Test Technology Technical Council (TTTC) since 2004. He is currently the Vice Chair of IEEE TTTC. He was the Chair of the Technical Committee on Fault-Tolerant Computing, China Computer Federation from 2008 to 2012, and the Steering Committee Chair of the IEEE Asian Test Symposium from 2011 to 2013. He was the Steering Committee Chair of IEEE Workshop on RTL and High Level Testing 2007 to 2010. He services as Associate Editor of the *Journal of Computer Science and Technology*, the *Journal of Low Power Electronics*, the *Journal of Electronic Testing: Theory and Applications*, and the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.