# Securing On-device Transformer with Hardware Binding and Reversible Obfuscation

Peichun Hua[1], Hanxiu Zhang[2], Tuo Li[3], and Yue Zheng[2]
*[1]School of Data Science, The Chinese University of Hong Kong, Shenzhen, Guangdong, 518172, P.R. China*
*[2]School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, Guangdong, 518172, P.R. China*
*[3]Shandong Yunhai Guochuang Cloud Computing Equipment Industry Innovation Co., Ltd., China*
*{peichunhua, hanxiuzhang}@link.cuhk.edu.cn, lituo@inspur.com, zhengyue@cuhk.edu.cn*

***Abstract*—Building a production-level deep learning (DL) model requires substantial data, computational resources, and expert knowledge. Therefore, a well-trained DL model is a desirable asset that can be subject to intellectual property (IP) theft. Apart from violating commercial interests, the stolen model facilitates various adversarial attacks and reduces the cost of building malicious applications. Thus, it is more urgent today to protect the DL models from leakage and theft. Existing popular solutions, such as watermarking-based schemes, provide only passive ownership verification after model theft. Instead, we aim to explore an active model IP protection method, especially for transformer models, which binds a model to specific hardware using physical unclonable functions (PUF). By obfuscating model weights with a reversible obfuscation scheme, Arnold's Cat Map (ACM), only authorized users (or hardware) can run the model on authorized machines, thereby actively preventing model misuse, even if the model is intercepted or leaked. We show that it not only protects model IP by incurring high accuracy loss without the presence of authorized hardware, but is also robust to retraining attacks, random key-guessing attacks, and encrypted-layer leakage attacks. In addition, our proposed method does not require additional retraining and incurs little inference overhead in practice. We believe that our proposed method can be useful and low-cost for the safe deployment of DL models at the edge. Our code for the software part is publicly available on GitHub [1].**

***Index Terms*—Intellectual Property, Transformer, Model Stealing Attacks, Physical Unclonable Functions**

## 1. Introduction

Modern deep neural networks (DNNs) have become indispensable in many applications, driving remarkable achievements in computer vision, natural language processing, and more. Among them, *transformer* models have emerged as a predominant architecture across various tasks due to their superior performance and scalability. These advanced models typically demand extensive data collection and massive computational resources for training, making well-tuned weights a valuable *intellectual property* (IP) asset.

Deploying transformers at the edge (e.g., in smartphones, IoT devices, or embedded systems) can bring substantial benefits, such as low-latency inference, reduced cloud dependence, powerful functionality without internet connectivity, and enhanced user privacy. However, it also introduces critical security challenges. Once a transformer model is locally stored on a device, malicious actors may seek to *steal* and *misuse* its parameters. With access to these parameters, adversaries could create harmful applications with minimal effort or launch adversarial attacks targeting the original model.

Although various IP protection strategies [1] have been proposed for neural networks—ranging from watermarking schemes to trusted execution environments (TEEs) [2]—they are often ill-suited for transformers deployed at the edge. Watermarking techniques require significant retraining efforts to embed ownership information and provide only a *passive* ownership verification mechanism rather than preventing model theft outright. They do not provide any formal security guarantee and are fragile against various attacks, such as removal attacks and spoofing attacks [3]–[11], or when the model is repurposed for other tasks (e.g., many black box methods embed and detect "trigger set" for classification models in the form of pairs of input and output, which can only be verified when the model is still used for the same purpose). Meanwhile, TEE-based approaches [2], [12]–[15] introduce substantial runtime overhead and memory constraints, which are particularly problematic for large transformer models. Consequently, there is a need for a practical and *proactive* IP protection method that can safeguard on-device transformers without requiring retraining or complicated hardware support.

Against this backdrop, our goal is to actively *bind* the transformer model's execution to a specific hardware device in order to prevent unauthorized usage—even if an attacker gains white-box access to the stored parameters. Specifically, we seek a solution that: (1) ensures only authorized hardware can correctly run the model; (2) incurs minimal inference overhead; (3) does not require retraining the original model; (4) does not negatively impact model performance as previous watermark methods did; (5) maintains platform

1. https://github.com/sarendis56/PUF-Transformer-IP-Protection

independence by operating solely on model weights without using framework-specific features; and (6) remains robust even if attackers have partial access to the training data and full knowledge of the network architecture.

To address these requirements, we propose a hardware-bound encryption framework for IP protection of transformers. Our approach leverages (1) Physical Unclonable Function (PUF) to bind the model to the target device. Each target device integrates a PUF that reliably derives a unique cryptographic key based on uncontrollable silicon variations. This key is never stored on the device, preventing attackers from extracting it from the memory. (2) Reversible Weight Obfuscation to effectively scramble the weights in a reversible yet highly disruptive manner. We selectively encrypt the attention and feedforward layers of the transformer using a lightweight chaotic map (e.g., Arnold's Cat Map (ACM)) or secure permutations. The correct PUF-based key is indispensable for reconstructing the original layer weights on the authorized device.

- We propose a novel IP protection framework that locks transformer models so they can execute correctly only on authorized devices with valid PUF-derived keys. Hardware-bound ownership verification can also be achieved, as the correct derivation of PUF keys requires the input of a correct owner ID.
- Our reversible obfuscation does not require retraining and operates in a layer-by-layer manner for enhanced security. Once decrypted on authorized hardware, the model achieves its original performance.
- We show that attackers with white-box access to encrypted parameters, partial training data, and knowledge of protected layers cannot feasibly recover or reuse the scrambled model.
- We demonstrate that inference overhead is minimal and that our obfuscation process applies to various transformer architectures, making it practical for real-world edge deployments.

The rest of the paper is organized as follows. Section 2 introduces background on transformer architectures, Arnold's Cat Map, and PUF. Section 3 details our threat model. Our IP protection framework is presented in Section 4. Section 5 describes our experimental evaluation. We compare our methods with existing techniques in Section 6 and conclude in Section 7.

## 2. Preliminaries

### 2.1. Transformer

Transformers are DL models with strong capabilities in language modeling [16], computer vision [17], time series [18], code generation [19], and software security [20]. A typical transformer [21] consists of a series of transformer layers stacked together. Each layer has two main components: *multi-head self-attention* and a *feedforward network*. Fig. 1 illustrates a standard transformer architecture, with our targets highlighted in deep blue.

**Multi-Head Self-Attention.** Given input embeddings $X \in \mathbb{R}^{n\times d}$, compute the Query (Q), Key (K) and Value (V) vectors by

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V,$$

and then the attention scores are computed with

$$\text{Attention}(Q, K, V) = \text{softmax}\big(QK^\top/\sqrt{d_k}\big)\, V.$$

where $W^Q, W^K, W^V$ are three learnable weight matrices for $Q, K, V$, and $d_k$ denote the dimensionality of $Q$ and $K$ vectors. To better capture multiple types of relationships (e.g., short-range and long-range) in the sequence at the same time, we typically split the attention mechanism into $h$ heads, compute attention individually, and aggregate with a $W^O$ matrix:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\, W^O,$$

where $\text{head}_i = \text{Attention}(XW_i^Q,\ XW_i^K,\ XW_i^V)$.

**Feedforward Network.** The FFN applies two linear layers with GELU:

$$\text{FFN}(X) = \text{GELU}(XW_1 + b_1)\, W_2 + b_2.$$

Residual connections and layer normalization follow each sub-layer. However, we focus on encrypting multi-head attention and FFN weights, the bulk of the weights in a transformer model.

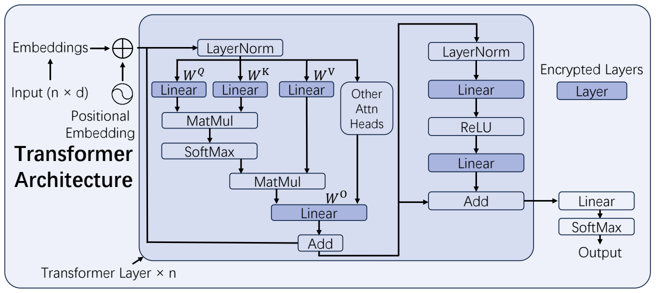


Figure 1. The block diagram of a standard transformer layer. The components targeted by our scheme are denoted in deep blue (see later in Sec. 4.2), which account for most of the parameters in the transformer layer.

### 2.2. Arnold's Cat Map

Arnold's Cat Map (ACM) is a chaotic transposition technique originally designed to scramble images as a more efficient alternative to block ciphers, as image data has a large capacity and high redundancy. ACM, along with other chaotic maps, including the Baker Map and Standard Map, was introduced into image encryption by Fridrich [22] and later applied to many other uses, such as medical image protection [23] and voice communication encryption [24].

The general form of Arnold's Cat Map is:

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = A^n \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \pmod{S} \tag{1}$$

Here, $(x_n, y_n)$ denotes the position of the element $(x_0, y_0)$ after $n$ iterations. In each iteration, all elements of the input matrix are shuffled into new positions. $S$ denotes the size of the input matrix. $A$ is a square $2 \times 2$ matrix, and the determinant of $A$ must be $\pm 1 \pmod{S}$ to ensure the decryption is reversible:

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = A^{-n} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \pmod{S} \tag{2}$$

There are several forms of ACM applied in practice [25]–[28]. In particular, Lin et al. [25] adopt a restricted form with 2 degrees of freedom in $A$, where

$$A = \begin{bmatrix} 1 & p \\ q & pq+1 \end{bmatrix} \tag{3}$$

We adopt this specific formulation due to its balance of sufficient chaotic properties for effective obfuscation and its relative simplicity in implementation and analysis, aligning with our goal of a lightweight protection scheme. We prove that this scheme is chaotic in the continuous domain, following the definition in [29] in the Appendix.

Since ACM encryption merely shuffles the input elements with a specific key, it does not affect the input distributions. This characteristic prevents the formation of outliers in the weights, as seen in [30]–[33], thereby reducing the likelihood that an adversary could infer the encrypted layers and keys through the analysis of these weights [34].

## 2.3. Physical Unclonable Functions (PUF)

PUF is a hardware security primitive designed based on uncontrollable and unpredictable manufacturing process variations [35] in circuit design. A PUF can be mathematically viewed as an irreversible probabilistic function that produces a random bit string (the *response*) given an external *challenge* to the circuit:

$$PUF : Ch \rightarrow R.$$

$$PUF(ch) : \ r \quad (ch \in Ch, r \in R).$$

where $Ch$ is the set of all external *challenges*, and $R$ describes the *responses* produced by the PUF. The challenge-to-response mapping of a specific PUF is unique and irreproducible, just as no two people have identical fingerprints due to natural biological variations, making PUF a promising technique for generating device-specific keys and hardware identifiers.

In practice, for better security and robustness against attacks and random errors, a technique called controlled PUF (CPUF) is often adopted such that PUFs can only be accessed via an algorithm that is physically bound to the PUF in an inseparable way [36], [37]. In particular, the hardware designer can create the package such that its existence has a significant influence on the PUF measurement, and any efforts to tamper with the PUF or probe PUF responses will change the behavior and destroy the PUF immediately. Meanwhile, the user or application engages with the PUF through a controlled and non-bypassable interface to reach the shared secret. In this work, it is assumed that the hardware provider incorporates such a control mechanism within the PUF design, ensuring that a physical adversary is unable to directly probe the PUF responses.

## 3. Threat model

**Assumptions and Guarantees** In our framework, we assume a deployment environment where transformer models are executed on edge devices equipped with a robust PUF design. In particular, 1) Any physical tampering made to the PUF circuits would destroy the variation property of the underlying circuit, thus producing the wrong PUF response to a given challenge. 2) Fuzzy extractor [35], [38], [39] is implemented to ensure the reliable generation of PUF response-based cryptographic keys under different environmental conditions. 3) We assume the PUF readout interface will be removed after completing the PUF CRP registration, which ensures that both the users and attackers do not have access to the PUF keys.

TABLE 1. Summary of the Threat Model

| Attacker Information | Accessible |
|---|---|
| Model architecture | ✓ |
| Offline Storage Access | ✓ |
| Partial Training Data / Downstream Dataset | ✓ |
| Runtime Memory Access | × |
| PUF-derived Keys (Per-device Secrets) | × |

Attacker knowledge and capabilities: A checkmark (✓) indicates that the attacker has access to the corresponding information or resource. A cross (×) indicates that access is restricted or explicitly denied under our threat model.

**Adversarial Capabilities** Table 1 demonstrates the adversarial capabilities. Attackers can examine the encrypted model parameters and architecture at rest (e.g., from a file or during transit). The adversaries may hold a *subset* of training data or other downstream datasets. And they have sufficient computing power to attempt to recover the model accuracy. We acknowledge that timing side-channel attacks may reveal information about which specific layers are encrypted. Our security analysis accounts for this possibility, and the robustness of the system under this attack is evaluated in Sec. 5.2.2. Additionally, we assume that the PUF-derived keys are inaccessible by disabling the PUF readout interface. We further assume that memory access during model runtime is unavailable. While advanced attacks [40]–[42] may potentially retrieve plaintext model weights from memory (e.g., with cold boot attacks [43]–[46]), our approach can be configured to dynamically decrypt each layer during computation and then re-encrypt them immediately afterward, significantly raising the bar for such attacks. In addition, existing attacks primarily focus on smaller Convolutional Neural Networks (CNNs), while none of them consider the much larger transformer models. We encourage future research to develop more sophisticated attacks on transformer models and against our defense.

## 4. Proposed Approach

This section presents our proposed approach. As shown in Fig. 2, the overall framework can be divided into three stages. In the first stage, the model provider trains the model using its training expertise, computing resources, and proprietary datasets. In the second stage, users or companies request to purchase and deploy the protected model. The provider interacts with them using a specific protocol, detailed in Sec. 4.1, encrypts the model weights based on the identity of the purchaser and the designated device characteristics derived from its PUF. After the purchase, the model can be deployed only on the specific machine. Eavesdropping or stealing weights from storage no longer makes them usable.

### 4.1. Secure Deployment Protocol

Securely deploying a transformer model on PUF-equipped hardware involves a coordinated protocol among three main entities: `Hardware Vendor`, `Model Provider`, and `User`. Fig. 3 depicts the protocol, with numbers denoting the different stages and letters denoting actions taken by different parties in a stage. Next, we introduce each stage as follows:

**(1) Hardware Acquisition and Registration:** (a) The `Hardware Vendor` is assumed to be trusted, responsible for characterizing or enrolling the device PUF, and storing the PUF CRPs in a secure database $DB$. After registration, the PUF readout interface will be removed completely. (b) Next, the `User` acquires PUF-enabled hardware (e.g., an FPGA or other domain-specific accelerator) from the `Hardware Vendor`, and (c) The `Hardware Vendor` will send the device and the unique device identifier *DID* to the user.

**(2) Model Purchase Request:** The `User`, intending to deploy a specific DL model, contacts the `Model Provider`. In the request, the `User` provides (or is designated) their user identifier *UID* and device identifier *DID*.

**(3) Device-Specific Key Material Retrieval:** (a) The `Model Provider`, upon receiving a legitimate purchase request, needs to obtain device-specific key material derived from the PUF on the `User`'s designated hardware. The `Model Provider` sends *DID* and the owner identifier *OID* to `Hardware Vendor`. (b) Once the request is received, the `Hardware Vendor` hashes *OID* to compute the challenge index $C$ and uses it to search for the corresponding response $R$ in the database. Then, the `Hardware Vendor` computes $K = h(R \parallel DID)$ (where $h$ is a hash function) and shares it with the `Model Provider` through a secure interface, which can be achieved with established communication protocols [47].

**(4) Model Encryption and Configuration:** Upon receiving the device-specific key material $K$ via a secure channel, the `Model Provider` treats it as an ephemeral secret. In a secure environment, the key is used a single time to obfuscate the plaintext transformer model, as illustrated in Sec. 4.3. Immediately after the encryption, $K$ is securely discarded and is never stored by the provider. This one-time process effectively binds the model to the unique characteristics of the `User`'s PUF. The `Model Provider` also prepares the necessary decryption configuration that will be used by the model at runtime. The `Model Provider` delivers the obfuscated model (with decryption configurations) to the `User`. The `User` can then deploy this model on their designated PUF-enabled hardware.

**(5) PUF-Driven In-Situ Decryption:** At runtime, the model queries the on-device PUF to regenerate $K$. This key exists only ephemerally in volatile memory and is used to de-obfuscate the model, typically on a layer-by-layer basis as needed for computation. The `User` does not have direct access to the PUF's raw responses or the intermediate cryptographic keys; these are managed internally by the secure model execution flow.

This protocol ensures that the obfuscated model remains non-functional if stolen or copied to a different hardware device, as the correct PUF responses required for decryption can only be generated by the original, authorized hardware.

### 4.2. PUF-based Single-layer Obfuscation

Fig. 4 depicts how we obfuscate one single layer with hardware support. Firstly, the hardware takes the `Model Provider`'s identity *OID* as input, hashes it, and applies it to the PUF to generate a corresponding response $R$, which is then integrated with the device identifier *DID* through a hash for the master key $K$ generation:

$$R = \mathrm{PUF}\big(h(\mathrm{OID})\big) \tag{4}$$

$$K = h\big(R \parallel \mathrm{DID}\big) \tag{5}$$

where $\parallel$ denotes concatenation. This master key integrates information from both the `Model Provider` and the devices, ensuring that any incorrect application of the ownership identifier or device mismatches will render the model inoperable, therefore achieving hardware-bound ownership verification and preventing unauthorized model redistribution. This verification can be demonstrated to a third party by having them provide a claimed Owner ID (OID) to the authorized device. The party can then observe whether the model's high accuracy is restored, which confirms ownership without requiring privileged access to the vendor's database or direct PUF queries.

We obfuscate multiple layers to improve security. Specifically, we expand $K$ using a cryptographic key derivation function (KDF) to generate layer-specific keys:

$$K_i = KDF(K) = K_i^{attn} \parallel K_i^{FFN} \tag{6}$$

where $K_i$ denotes the generated key for layer $i$. Common key expansion methods, such as the AES key expansion

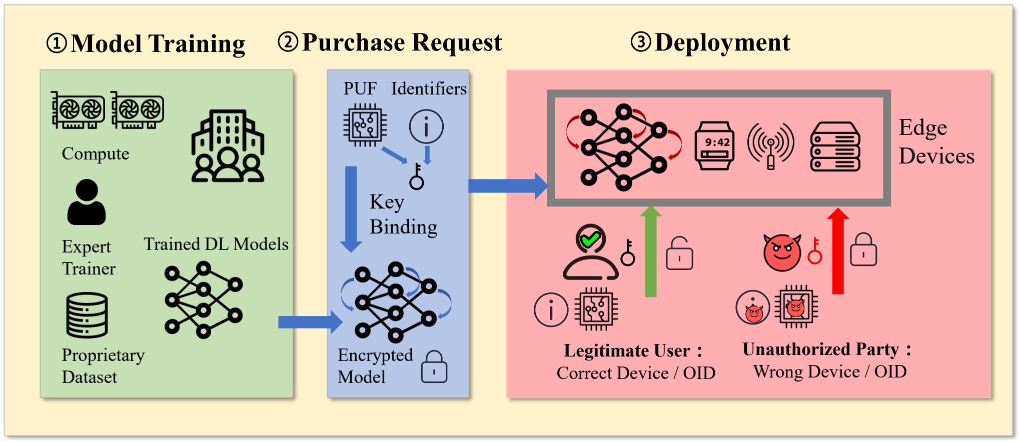
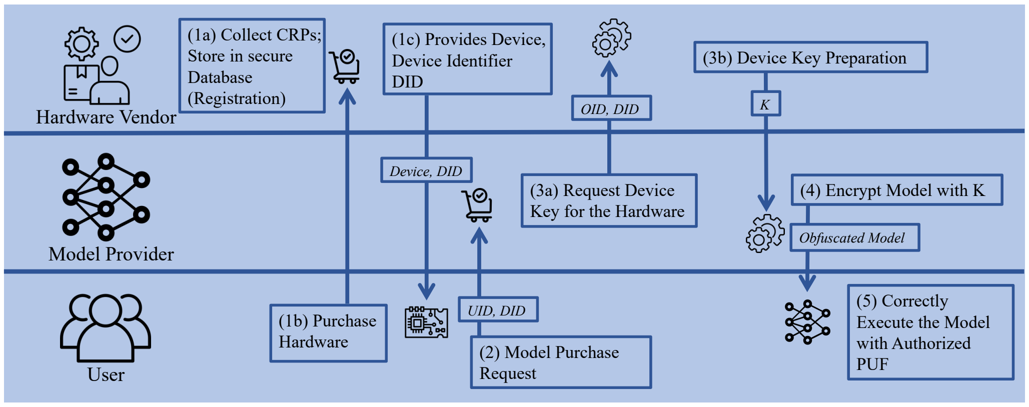


Figure 2. Overview of our proposed framework, which consists of three stages. (1) Model Training: The trainer trains the model using proprietary datasets and compute resources. (2) Purchase: The model is encrypted and bound to the purchaser's device using PUF-derived keys, ensuring that the model can only be deployed on authorized hardware. (3) Deployment: The model remains secure against on-device extraction and server breaches, as unauthorized decryption is infeasible without the device-specific PUF keys.



Hardware Vendor
(1a) Collect CRPs; Store in secure Database (Registration)
(1c) Provides Device, Device Identifier DID
OID, DID
(3b) Device Key Preparation
K
Model Provider
Device, DID
(3a) Request Device Key for the Hardware
(4) Encrypt Model with K
Obfuscated Model
User
(1b) Purchase Hardware
UID, DID
(2) Model Purchase Request
(5) Correctly Execute the Model with Authorized PUF

Figure 3. The protocol of the interaction between the `Hardware Vendor` (e.g., FPGA vendor), `Model Provider`, and `User`. The protocol is used to ensure secure model deployment.

scheme or dedicated KDF [48], can be adopted. Each $K_i$ is partitioned into two parts, $K_i^{attn}$ and $K_i^{FFN}$, used to obfuscate the self-attention weights $(W_i^Q, W_i^K, W_i^V, W_i^O)$ and the FFN weights $(W_i^1, W_i^2)$ respectively, ensuring that only authorized hardware can correctly execute the model while maintaining efficiency in edge deployment scenarios. We observe that the attention matrix is typically square for most transformer models; therefore, we can use the ACM method to encrypt these matrices. Let $ACM(message, key)$ denote the ACM encryption, the matrix $A$ determined by $p, q$ in Eq. 3 is used as the ACM key in our proposed method:

$$W_i^{j\prime} = ACM(W_i^j, (p_i^j, q_i^j)), \text{ for } j \in \{Q, K, V, O\} \tag{7}$$

where $(p_i^j, q_i^j)$ are subkeys derived from $K_i^{attn}$ by partitioning. To mitigate potential timing side-channel attacks, where an attacker infers secret data by measuring variations in execution time, we ensure that our de-obfuscation process runs in constant time. The execution time of the ACM operation is determined by the number of iterations, $n$, independent of the secret key values $(p_i^j, q_i^j)$. By fixing the number of iterations to a constant value, we make the de-obfuscation time for any given layer constant. This transforms the dynamic chaotic map into a specific key-dependent permutation. The security of the obfuscation no longer relies on the process being dynamically chaotic, but on the properties of the permutation selected by the true secret key: the parameters $p_i^j$ and $q_i^j$. The underlying chaotic nature of the ACM system ensures that this key space is vast and exhibits a strong avalanche effect, where a small change in the key $(p, q)$ results in a completely different permutation.

Unlike attention weights, the FFN weights in transformers are typically not square. The dimension of the FFN layer is often four times the dimension of the model's hidden states/embeddings, which makes ACM encryption targeting square matrices directly applicable. Instead, we suggest permuting the FFN weights $W_i^f$ $(f = 1, 2)$:

$$W_i^{f\prime} = P(W_i^f, h(K_i^{\text{FFN}} \parallel f)), \text{ for } f \in \{1, 2\} \tag{8}$$

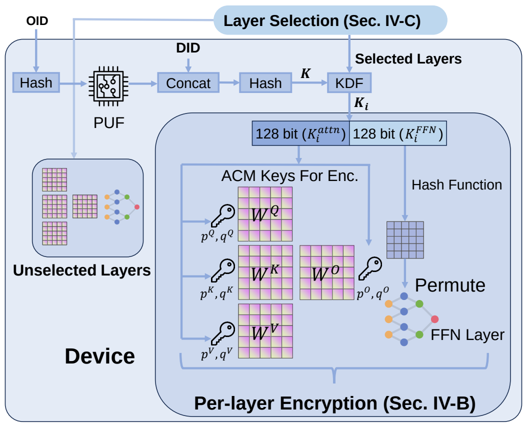


Figure 4. Our obfuscation scheme detailed in Sec. 4.

where $P(message, key)$ denotes the permutation, such as Knuth Shuffle [49], and $h(K_i^{\text{FFN}}, f)$ is used to calculate the key for each FFN matrix. We found that this permutation scheme is very effective in degrading model performance if incorrect keys are used and is generally much faster than ACM encryption. However, it is important to note that relying exclusively on permutation may still present a security vulnerability, as evidenced in [50], since permutation alone does not meet the criteria for cryptographic security.

In our proposed scheme, we do not impose constraints on the specific type of PUF utilized. Nonetheless, it is important that the PUF consistently generates responses that can serve as the basis for the required key material. This can be ensured using the fuzzy extractor mechanism with error correction incorporated [35]. Thanks to the PUF, the obfuscation keys are effectively generated only during model inference on the specific device and are concealed when the model is in transit or at rest.

## 4.3. Layer Selection and Model Obfuscation

Obfuscating all transformation layers for model performance locking is straightforward but entails a great deal of computational overhead during inference. To reduce overhead while maintaining strong robustness, we propose selectively identifying a minimal subset of transformation layers for encryption. Algorithm 1 outlines the heuristic approach for the "layer selection" block in Fig. 4, which only needs to be executed once on the model owner's side.

The main idea of Algorithm 1 is to identify the layers that are most accuracy-degrading when encrypted. The algorithm takes as input a pre-trained transformer model with an initial accuracy $acc_0$ and the target accuracy threshold $\epsilon$ as the stop criterion. This algorithm is run before deploying the model to specific users, so we designate randomly generated keys for analysis. Encrypting with distinct random keys introduces independent perturbations in each layer. When multiple layers are encrypted, the perturbations propagate through the network and compound across layers. By the central limit theorem, the aggregated effect of these perturbations—despite originating from different layers—converges to a noise distribution with predictable statistical properties, provided the perturbations are independent, bounded in magnitude, and sufficiently numerous. This universality in noise distribution ensures that the model's output deviates from its original behavior in a consistent manner, regardless of the specific random keys used. This aligns with our empirical observations, where different keys yield statistically indistinguishable performance degradation under the same encryption configuration (cf. Fig. 6). This key-agnostic property has favorable implications for our analysis. While Algorithm 1 is presented with freshly generated random keys for cryptographic rigor and generality, our **empirical implementation** of this analysis uses a single fixed (but randomly generated) default key. This approach ensures both efficiency and reproducibility without compromising the validity of the layer impact analysis.

**Algorithm 1** Selection of impactful layers

```
Require: Initial model accuracy $acc_0$ and target accuracy threshold $\epsilon$.
Ensure: Layers to be encrypted that will incur sufficient accuracy drop
 1: $\mathcal{L}_{encrypted} \leftarrow \emptyset$
 2: $acc_{current} \leftarrow acc_0$
 3: while $acc_{current} > \epsilon$ do
 4:     $impacts \leftarrow \emptyset$
 5:     for layer $l \notin \mathcal{L}_{encrypted}$ do
 6:         $W_l \leftarrow \{W^Q, W^K, W^V, W^O, W^{FFN}\}$
 7:         $K_l \sim \{0,1\}^{256}$          ▷ Generate random layer key
 8:         $W'_l \leftarrow$ Obfuscate $W_l$ with $K_l$
 9:         Replace $W_l$ with $W'_l$ in model
10:         $acc_l \leftarrow$ Evaluate model accuracy
11:         $impacts \leftarrow impacts \cup \{(l, acc_0 - acc_l)\}$
12:         Restore original weights $W_l$
13:     end for
14:     $l_{best} \leftarrow \arg\max_l impacts$
15:     Obfuscate the layer $l_{best}$ with random $K_l$ again
16:     $acc_{current} \leftarrow$ Evaluate model accuracy
17:     $\mathcal{L}_{encrypted} \leftarrow \mathcal{L}_{encrypted} \cup \{l_{best}\}$
18: end while
19: return $\mathcal{L}_{encrypted}$
```

We systematically evaluate each layer's contribution to the model's accuracy with random keys by 1) temporarily encrypting individual layers (Lines 6-9), 2) measuring the resulting accuracy drop (Lines 10-11), 3) restoring the original weights (Line 12), and 4) comparing the impact across layers (Line 14), encrypting the one with the highest impact (Line 15). Finally, the set of layers with sufficient performance degradation is returned. Let this set of layers be $L_\epsilon$. To further enhance security against sophisticated attacks, such as those attempting to infer keys or reverse the obfuscation (as discussed in Sec. 5.2.3 and Sec. 5.5.2), we then augment $L_\epsilon$ by encrypting a small, fixed number of additional layers. These extra security layers are critical: they increase the search space for layer-guessing attacks and

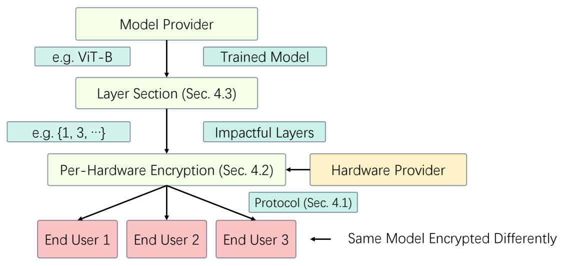


Figure 5. Complete model protection workflow. The model provider identifies impactful layers using Algorithm 1, then encrypts them using hardware-specific PUF-derived keys, allowing the same model to be securely distributed to multiple end users with different device-specific encryptions.

disrupt the feedback loop for an attacker. If only the most impactful layers were encrypted, an attacker who correctly guesses a single key might see a measurable accuracy gain, confirming their guess. By adding more encrypted layers, such single-key verifications become unreliable. We choose these additional layers randomly, as our investigation showed no significant benefits from more complex selection heuristics. This results in the final set of encrypted layers, $L_{encrypted}$, which includes both the layers critical for accuracy degradation and these extra security layers. The total number of layers in $L_{encrypted}$ then determines the required keys generated by KDF.

As shown in Fig. 5, our complete protection workflow begins with the model provider's trained transformer model (e.g., ViT-B). Using Algorithm 1, we first identify the most impactful layers for encryption—those that cause the greatest degradation in accuracy when obfuscated. We then augment this set with a few additional, strategically chosen security layers (e.g., 4 in our evaluation in Sec. 5) to harden the defense. These selected layers are then encrypted using device-specific keys derived from each end user's hardware PUF, which is facilitated through interaction with the hardware provider. This approach enables a single model to be distributed to multiple end users, with each instance protected by a unique encryption that binds it exclusively to the authorized hardware. Since each encryption uses different hardware-specific keys, the model remains secure against unauthorized access, even if multiple encrypted versions are analyzed together.

## 5. Experiments

In this section, we conduct extensive experiments. We validate our approach on vision transformers including ViT [51], Swin Transformer [52], DeiT [53], BeiT [54], and their multiple variants with different model sizes, input resolutions, or corresponding quantized versions. Table 2 provides the details of these models and their original model accuracy ($Acc_0$) trained on the ImageNet dataset. We report top-1 accuracy in all evaluations. The base-sized and large-sized models are denoted as 'B' and 'L', with 12 and 24 transformer layers, respectively. For example, ViT-B denotes the base-sized ViT model. The input resolution is $224 \times 224$ for most models while it is $384 \times 384$ for models ViT-B-384, ViT-L-384, and Swin-B-384. The proposed method is evaluated for both effectiveness (Sec. 5.1) and robustness (Sec. 5.2). The overhead and ablation studies are also presented in Sec. 5.3 and Sec. 5.5, respectively. Our solution is framework-agnostic, so we choose PyTorch and make use of the Transformer library by Hugging Face [2]. When running Algorithm 1, $\epsilon$ is chosen to be 0.15%, as random guesses of 1000 classes have only 8.3% probability to be correct more than 0.15% of the time. The experiments are all carried out on a single NVIDIA GeForce RTX 4090.

### 5.1. Effectiveness Test

Our proposed scheme is fully reversible and lossless by construction. When the model is decrypted on an authorized device using the correct PUF-derived key, it suffers no accuracy loss. We empirically verify this in Table 2, which shows that the decrypted accuracy ($Acc_{dec}$) is equal to the original accuracy ($Acc_0$) across all tested models. The effectiveness of our protection is therefore measured by the drastic accuracy drop ($Acc'$) that occurs when the model is used without the correct key. Table 2 presents $Acc'$ of the 10 models, and as anticipated, our scheme significantly diminishes the model's accuracy even if only a small number of layers are encrypted. In Table 2, $Acc_1(i)$ presents the model accuracy after Algorithm 1 has iteratively encrypted the specific layers (indicated in parentheses by their indices) required to reduce performance to near random-guess levels (i.e., meeting the $\epsilon$ threshold). For instance, for ViT-B, encrypting layers 0 and then 10 dropped the accuracy to 0.08%. For ViT-L, encrypting only layer 1 is sufficient to achieve 0.1% accuracy. Following this, a fixed number of additional layers (typically 4 for base-sized models and 6 for large-sized models) are encrypted to bolster security. We show in the ablation study (Sec. 5.5.2) that encrypting fewer layers might lead to higher attack accuracy; however, our framework still prevents fully recovering the model's capability. The $Acc'$ row shows the final model accuracy after all selected and additional security layers are encrypted, along with the total count of encrypted layers. Empirically, we found no significant difference between the different variants of the same model in terms of resolution and weight precision.

### 5.2. Robustness Test

**5.2.1. Retraining Attack.** The attacker may attempt to reconstruct the model's capability through retraining, either in the same application or a different one. To evaluate the robustness of the proposed approach against retraining attacks, we assume that the attacker possesses *a subset of* the training dataset along with a smaller downstream dataset. Accordingly, we consider two cases:

**Same-application retraining attack.** In this scenario, the attacker retrains the obfuscated model with up to 20%

2. https://huggingface.co/docs/transformers/en/index

TABLE 2. Models, parameter counts, and baseline vs. locked accuracies.

| **Model** | **ViT-B** | **ViT-B-384** | **ViT-L** | **ViT-L-384** | **Swin-B** | **Swin-B-384** | **BeiT-B** | **DeiT-B** | **ViT-B-8b** | **ViT-B-4b** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Params(M)** | 86 | 86 | 307 | 307 | 88 | 88 | 86 | 86 | 86 | 86 |
| $Acc_0$(%) | 80.33 | 83.91 | 81.98 | 84.96 | 85.14 | 86.44 | 84.54 | 80.20 | 80.24 | 80.20 |
| $Acc_{dec}$(%) | 80.33 | 83.91 | 81.98 | 84.96 | 85.14 | 86.44 | 84.54 | 80.20 | 80.24 | 80.20 |
| $Acc_1(i)$(%) | 0.08 (0, 10) | 0.09 (0, 11) | 0.1 (1) | 0.1 (1) | 0.10 (2-9, 2-8)* | 0.10 (2-9, 3-1) | 0.11 (2, 9) | 0.11 (8, 0) | 0.07 (6) | 0.08 (7) |
| $Acc'$(%) | 0.10 \| 6 | 0.09 \| 6 | 0.10 \| 7 | 0.07 \| 7 | 0.10 \| 6 | 0.08 \| 6 | 0.14 \| 6 | 0.10 \| 6 | 0.11 \| 5 | 0.10 \| 5 |

$Acc_0$ is the original accuracy. $Acc_{dec}$ is the accuracy after decryption with the correct key, demonstrating full performance restoration. $Acc_1(i)$ denotes accuracy after Algorithm 1 has encrypted the layers listed in parentheses. The notation x|y in $Acc'$ indicates final accuracy $x$% when $y$ layers are encrypted. *We treat two blocks in Swin Transformer as one layer and use (x-y) to denote the "y-th" layer in the "x-th" stage.

of the original training dataset (i.e., ImageNet in our experiments). Table 3 presents the *best* final result that the attacker can obtain in five random retraining experiments. The results show that the model accuracy stops increasing after only a few epochs, indicating that even with 20% of the ImageNet dataset, an adversary is *unable* to retrain and achieve the full accuracy of the original model from its encrypted counterpart.

TABLE 3. Same-application retraining attack results.

| Epoch | ViT-B [10%] | ViT-B | ViT-B-384 | Swin-B-224 | Swin-B-384 |
|---|---|---|---|---|---|
| 1 | 13.5 | 32.4 | 31.0 | 67.5 | 66.9 |
| 3 | 41.7 | 53.9 | 56.5 | 71.4 | 70.5 |
| 5 | 47.1 | 56.4 | 59.3 | 71.4 | 70.8 |
| 10 | 47.2 | 56.4 | 59.3 | 71.4 | 70.8 |
| 20 | 47.9 | \ | \ | \ | \ |

ViT-B[10%] is retrained with 10% of the original dataset. Others are retrained with up to 20% of the original dataset.

**Domain-transfer retraining attack.** After retraining with the partial original dataset, the attacker may further fine-tune the model with a downstream dataset for alternative purposes. Table 4 presents the model accuracy (%) by fine-tuning the ViT-B model with downstream datasets CIFAR-10, CIFAR-100, and Caltech 101. The column "Original" denotes the model accuracy by fine-tuning the unencrypted ViT-B model directly, while the columns "ViT-B[x%]" present the model accuracy the adversary can obtain by fine-tuning the encrypted model, which is first retrained with $x$% of the original training dataset. Compared with the column "Original", Table 4 shows that it is difficult for the attacker to restore full accuracy even under their very favorable conditions: possessing 20% of the original training dataset and fine-tuning for a very simple downstream task. This evidence substantiates the practicality of our framework in real-world applications and its potential to deter IP theft of models.

TABLE 4. Domain-transfer retraining attack results on ViT-B.

| **Downstream** | Original | ViT-B[0%] | ViT-B[10%] | ViT-B[20%] |
|---|---|---|---|---|
| **CIFAR-10** | 97.25 | 76.62 | 94.01 | 94.20 |
| **CIFAR-100** | 91.71 | 46.93 | 77.93 | 81.10 |
| **Caltech 101** | 98.16 | 58.35 | 91.42 | 94.47 |

ViT-B is fine-tuned on CIFAR-10, CIFAR-100, and Caltech 101 with varying amounts of the original dataset. ViT-B[0%] uses no original data.

**5.2.2. Leakage of Encrypted Layers.** The proposed scheme secures the model through the encryption of specific layers using a PUF-derived key. Existing literature indicates that schemes which modify the distribution of weights may present vulnerabilities that invite statistical attacks, thereby compromising security [34], [55]. Theoretically, both the ACM encryption and permutation do not alter the distribution of weights. It would be difficult for the attacker to identify the obfuscated layers without knowing the PUF response. However, we further discuss the scenario where the attacker has learned which specific layers of the deployed model are encrypted. This situation can arise in centralized deployment and a remote attacker utilizing side-channels. Prior side-channel studies [56]–[58] show that an adversary with sufficient device access (e.g., co-located on the same device or granted limited sharing of memory/cache resources) can measure fine-grained execution patterns and deduce sensitive information. Such methods can reveal the moments at which certain weights are decrypted and loaded, thereby indicating which layers of the model are encrypted in our scheme.

Once the attacker knows which subset of layers is encrypted, one might expect that directly fine-tuning *only the encrypted layer* using a smaller dataset could yield a near-restored model. To examine this threat, we retrained *only* the encrypted layers while keeping the rest of the model's layers fixed. Table 5 provides the accuracy obtained at various training epochs. Surprisingly, the accuracy remains substantially worse than the baseline (in contrast to retraining *all* parameters, cf. Table 3).

TABLE 5. Fine-tuning only encrypted layers with a subset of the original dataset.

| Epoch | ViT-B [20%] | ViT-B [10%] |
|---|---|---|
| 1 | 7.7 | 4.7 |
| 3 | 17.3 | 11.0 |
| 5 | 23.7 | 14.9 |
| 10 | 31.2 | 17.9 |
| 20 | 31.2 | 17.9 |

Five random experiments are conducted with the best final result reported. The results are worse than retraining all parameters, as presented in Table 3.

Conceptually, our encryption imposes sufficient entropy to break the loss landscape of the original weights. When the attacker focuses only on the encrypted layers, the attainable performance after retraining is almost as poor as training from scratch on a randomly initialized model. We observe minimal gains from allowing more epochs or a larger subset of data. As interpreting this phenomenon in more depth may require an investigation into the geometry of the altered weight space, we leave detailed theoretical exploration for future work.

**5.2.3. Key Recovery Attacks.** Even if an attacker identifies the encrypted layers, as discussed in Section 5.2.2, a direct attack would involve attempting to recover the secret keys. We argue that such key-recovery attacks are computationally infeasible due to several compounding factors: 1) The key space is too large for a brute-force search. The combination of the ACM parameters $(p, q)$ for attention layers and the seeds for the FFN permutations presents a vast search space that makes exhaustive guessing impractical. 2) More efficient, gradient-based search methods are inapplicable. The permutation operations central to our obfuscation—both Arnold's Cat Map and the Knuth Shuffle for FFNs—are non-differentiable. This prevents an attacker from using gradient descent or similar optimization techniques to iteratively find the correct key. 3) Heuristic-based searches are thwarted by a strong avalanche effect. A robust encryption scheme ensures that an incorrect key, even one numerically "close" to the correct key, produces an output statistically indistinguishable from random noise. Our experiments, illustrated in Fig. 6, confirm this property. This lack of a "closeness" metric provides no feedback to guide an intelligent search, rendering it no more effective than random guessing.

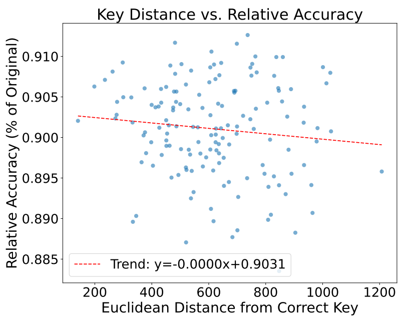


Figure 6. Sensitivity of model accuracy to key changes. We encrypt the first layer weights of ViT-B with a random key and test decryption using different incorrect keys at different distance from the true key of various distance to the correct one. The y-axis represents **relative accuracy**, defined as the ratio of the model's accuracy with an incorrect key to its original model accuracy. The plots show that incorrect keys consistently result in severely degraded performance, demonstrating a strong avalanche effect.

In addition, we consider an extreme scenario where the attacker is fortuitously able to correctly guess the key for a specific encryption layer. We examine whether the decryption of an entire layer yields significant improvements in accuracy, which could inadvertently reveal information about the correctness of a random guess to an attacker. To evaluate this, we conduct experiments using three distinct models (all with the base version, 224 resolution, and 6 layers encrypted according to the proposed scheme) to determine if the accurate decryption of *any* encrypted layer, including attention and FFN weights, results in notable enhancements in model accuracy. As demonstrated in Table 6, this action leads to only minor variations in model accuracy due to the additional encrypted layers that enhance security.

TABLE 6. Model accuracy change after any layer is totally decrypted.

| Layer | ViT-B | DeiT-B | BeiT-B |
|---|---|---|---|
| 1 | +0.01% (0) | +0.04% (8) | -0.04% (9) |
| 2 | +0.01% (2) | +0.01% (2) | -0.04% (0) |
| 3 | 0% (10) | +0.01% (0) | +0.03% (5) |
| 4 | 0% (4) | +0.01% (3) | +0.01% (4) |
| 5 | 0% (11) | +0.01% (7) | 0% (2) |
| 6 | 0% (5) | 0% (1) | 0% (6) |

For example, +0.01%(0) means the model accuracy increased 0.01% after decrypting the 0th layer of the ViT-B model.

## 5.3. Inference Overhead

Managing the inference overhead associated with our encryption scheme is crucial for practical deployment. Our experiments involve a sequential layer-by-layer de-obfuscation process: for each encrypted layer, the model initially de-obfuscates the weights, proceeds with the original layer operation, and subsequently re-encrypts them before moving to the next layer. This dynamic approach enhances security by minimizing the time plaintext weights reside in memory. Figures 7 and 8 systematically evaluate this overhead on both CPU and GPU platforms for a ViT-B model on ImageNet dataset with a batch size of 64. The analysis examines two critical factors: the number of ACM iterations $(n)$ and the number of encrypted layers. By fixing one factor while varying the other [3], we can isolate their individual impacts on total inference time. The evaluation duration for PUF key generation typically occurs at the microsecond level, which is negligible compared to the millisecond-scale forward pass of transformers, and is thus omitted from this analysis.

The results reveal a significant difference in relative overhead between CPU and GPU execution. On the CPU (Fig. 7), the baseline forward pass is computationally intensive, making the additional overhead from our scheme relatively small. For instance, encrypting 12 layers with 3 ACM iterations results in a total overhead factor of just 1.119x. In contrast, the GPU's highly parallel architecture accelerates the forward pass more dramatically, causing the sequential decryption/encryption operations to constitute a larger portion of the total time. On the GPU (Fig. 8), the same configuration results in a more substantial 1.671x overhead when iterating 12 times and 1.580x when encrypting 12 layers.

Across both platforms, the stacked bars clearly show that the overhead is dominated by ACM operations, while the FFN permutations add minimal latency. For efficiency, it is therefore advisable to use a small number of ACM iterations $(n)$. Our security analysis in Sec. 5.2 confirms that a small value such as $n = 3$ is sufficient to provide robust protection. The number of encrypted layers is the primary lever for balancing security and performance; increasing the

3. When not varied, the constant factors are: $n = 3$ ACM iterations and 6 encrypted layers.

layer count provides stronger security at the cost of higher, but predictable, latency.

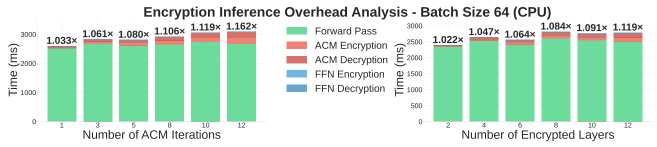


Figure 7. Inference Overhead on CPU (Batch Size = 64). The stacked bars show the time breakdown in milliseconds. (a) Performance vs. Number of ACM Iterations. (b) Performance vs. Number of Encrypted Layers. The numbers above each bar indicate the total overhead factor compared to the baseline forward pass.

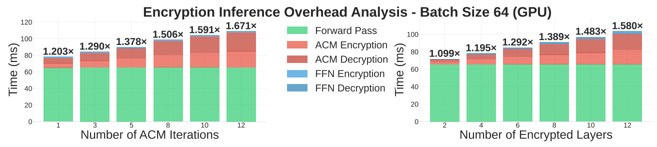


Figure 8. Inference Overhead on GPU (Batch Size = 64). (a) Performance vs. Number of ACM Iterations. (b) Performance vs. Number of Encrypted Layers. Note the significantly lower absolute inference time compared to CPU, which makes the relative overhead of decryption/encryption more pronounced.

TABLE 7. Encryption/Decryption Runtimes.

| Method | Time (seconds) on ViT-B | Time on ViT-L | Device |
|---|---|---|---|
| AES-CBC | 2.30 | 9.02 | CPU |
| AES-CTR | 1.34 | 5.17 | CPU |
| AES-GCM | 1.45 | 5.18 | CPU |
| TEA | 19.63 | 69.51 | CPU |
| XOR Cipher | 3.72 | 13.47 | CPU |
| Our Method w. CPU | **1.19** | **2.58** | CPU |
| XOR Cipher w. GPU | **0.043** | 0.22 | GPU |
| **Our Method** | **0.044** | **0.051** | GPU |

Evaluations are carried out on 330MB ViT-B model and 1161MB ViT-L. Each experiment is carried out 5 times and the average is taken. For ACM, $n$ is set to 3, and the encrypted layers are 6 for ViT-B and 7 for ViT-L, aligning our previous experiment.

## 5.4. Comparison

To highlight the advantages of our proposed method, this section conducts a comparative analysis with existing works, evaluating both encryption module overhead (Sec. 5.4.1) and the qualitative IP protection features (Sec. 5.4.2).

**5.4.1. Comparison with common encryption methods.** Conventional methods for protecting on-device models often encrypt the entire model file using schemes like AES, Tiny Encryption Algorithm (TEA), and basic bitwise operations (XOR), then decrypt it fully into memory before inference, as observed in many Android apps [41]. While this secures the model at rest, it leaves the full plaintext model exposed during execution, which has been exploited in prior attacks [40]–[42], [66]. Moreover, block ciphers such as AES are optimized for general data and CPU execution, not for the structure or redundancy of neural network weights. In contrast, our ACM-based obfuscation can decrypt weights one layer at a time during inference and re-encrypt them immediately after use. This ensures that the full plaintext model is never simultaneously in memory, reducing vulnerability to runtime memory attacks. ACM operations are also suited for GPU acceleration.

We benchmarked our method against AES (with AES-NI support), TEA, and XOR (without and with GPU acceleration), evaluating the additional wall-clock time [4] required for decrypting and re-encrypting a ViT-B model (330 MB) and a ViT-L model (1161 MB). CPU methods are tested with an `Intel Xeon Platinum 8255C @ 2.50GHz` with 32 cores. For GPU-based methods, only the time spent on the GPU (excluding CPU-GPU communication time) is reported. Table 7 shows that our proposed method achieves the lowest time overhead, even without GPU support, and is faster than different modes of AES with special hardware acceleration. In the meantime, it is highly parallelizable and amenable to GPU acceleration. In particular, our previous experiments in Sec. 5.1 show that in larger models, our method requires even fewer layers to achieve satisfactory accuracy reduction; therefore, it scales more gracefully than methods like the naive XOR cipher while providing better security.

**5.4.2. Comparison with other IP protection approaches.** Table 8 compares our proposed work with representative IP protection approaches in the literature. In particular, the “Retrain” column denotes whether the model owner needs to retrain the models with additional regularizers or customized datasets, etc., which often impede convergence and prolong the training process, while also lacking scalability for a substantial number of users. Although passive strategies typically do not introduce inference-time overhead (“Inference Overhead” column), they cannot offer protection but only provide ownership tracing. Our previous discussions in this section demonstrate that our proposed scheme achieves sufficient security with a tolerable level of inference overhead. Our scheme serves as a pilot study to explore transformer protection. Li et al. [13] address on-device transformer deployment by securing layers with weight matrix permutations within a TEE. However, it suffers from high inference overhead due to frequent TEE–GPU interactions. Meanwhile, given the cost of training transformers, models are often adapted and fine-tuned (e.g., using LoRA [67]). Attackers with access to permuted weights and the original public model might align columns to recover the permutations. Like all encryption-based schemes, our approach does not hurt model performance (“Acc. Loss” column).

## 5.5. Ablation Study

**5.5.1. Impact of Encrypting FFN.** We first performed an ablation study on the encryption of the FFN layer in the

4. We report the wall-clock time for a single round of decryption and encryption for fair comparison; in practice, inputs might be evaluated in minibatches such that this time can be amortized to each input sample in the batch.

TABLE 8. Comparison of prior IP protection methods.

| Work | Type | Mechanism | Retrain? | Target Models | Inference Overhead | Acc. Loss |
|---|---|---|---|---|---|---|
| Uchida et al. [59] | Passive | White-box Watermarks | Yes | ConvNet | None | ✓ |
| Adi et al. [60] | Passive | Backdoor Training | Yes | ConvNet | None | ✓ |
| Chen and Wu [61] | Active | Active Control Module | Yes | ConvNet | Low (Anti-piracy module) | ✓ |
| M-LOCK [62] | Active | Data Poisoning | Yes | ConvNet | None | ✓ |
| DeepIPR [63] | Active | Passport-based Verification | Yes | ConvNet | Low (Around 10%) | ✓ |
| NNLock [64] | Active | Weight Encryption | No | ConvNet | Moderate (1.5-2x) | × |
| Phantom [65] | Active | Architecture Obfuscation with TEE | Yes$^\sharp$ | ConvNet | Moderate (3.5x$^\natural$) | ×$^\natural$ |
| ChaoW [25] | Active | Weight Encryption | No | ConvNet, RNN | Moderate | × |
| TransLinkGuard [13] | Active | Weight Permutation | No | Transformers | High (TEE$^\dagger$) | × |
| Ours | **Active** | Weight Encryption | **No** | **Transformers** | Low ($\approx 0.2x^\ddagger$) | × |

Comparison of DL Model IP Protection Methods. $^\sharp$Include neural architecture search for architecture obfuscation, which is about 26 hours for ResNet-18 on CIFAR-10 with a single A6000 GPU. $^\natural$Include the overhead for additional obfuscation layer, TEE latency, and TEE-GPU communication; results on ResNet 18, SGX2, Top-3 Layer strategy. Though their scheme doesn't introduce accuracy loss per se, to improve efficiency, they employ 8-bit quantization when transmitting data between TEE and GPU. $^\dagger$The authors [13] provide a rough estimate of the additional FLOPs incurred by the permutation operations but lack a comprehensive evaluation accounting for the TEE communication overhead. $^\ddagger$Assume mini-batch size of 64, ACM iterations set to 3, and extra encrypted layers set to 4 for a ViT-B.

transformer layers. Table 9 demonstrated the effectiveness of implementing solely the ACM encryption to attention weights. The experimental results show that it requires a considerably larger number of layers to reach a similar model degradation effect without FFN obfuscation. As demonstrated in Sec. 5.3, encrypting more layers would introduce significant overhead, while not obfuscating FFN has little benefit in inference time. Therefore, our proposed method of encrypting both attention weights and FFN weights is necessary.

**5.5.2. Impact of Extra Security Layers.** To empirically validate our claim that adding more extra layers enhances security, we conducted a controlled experiment isolating this variable. We performed retraining attacks on ViT-B models obfuscated with a varying number of additional security layers, from zero to four. For each case, the attacker's best attempt at recovering performance by retraining the model's full parameters with 20% of the original training dataset was recorded.

The results, presented in Table 10, demonstrate a clear trend: increasing the number of randomly selected extra security layers significantly degrades the model's post-retraining performance. This enhances the robustness of the IP protection, as it becomes progressively harder for an attacker to restore the model's functionality. We observe that after adding three extra layers, the attack accuracy begins to plateau, indicating that a small number of additional layers is sufficient to provide a strong defense. In practice, the number of extra layers can be chosen to balance this enhanced security against a minor increase in inference overhead.

## 6. Related Works

This section reviews IP protection methods for deep learning models. We hereby categorize previous efforts into **passive protection** and **proactive protection**.

### 6.1. Passive Protection

Passive protection (also called reactive protection) involves extracting a unique identifier (fingerprint) from the target model [68]–[73], embedding watermarks into models by training with additional objectives [6], [9], [59], [60], [74]–[79], or utilizing training-free backdoors [9], [80]. These schemes deter theft by enabling model owners to prove prior possession in disputes. However, they do not prevent unauthorized usage or distribution by adversaries who run the stolen model privately; additionally, these techniques are often prone to various attacks, such as removal attacks and spoofing attacks [3]–[5], [7], [8], [10], [11], [81], [82]. Watermarking techniques often require additional training or fine-tuning, which may be infeasible for large, ever-evolving models. Despite their limitations, such methods can be complementary to the proactive protection described below.

### 6.2. Proactive Protection

Proactive schemes aim to block unauthorized inference from the outset, typically by requiring a correct key or license. They can be classified based on whether they rely on weight encryption, trusted execution environment, or retraining-based mechanisms.

**Encryption-based methods.** Several works use cryptographic keys to lock neural networks, focusing on minimal overhead and no additional training. For instance, Alam et al. [64], Lin et al. [25], and Mukherjee et al. [83] encrypt weights to prevent unauthorized inference. Goldstein et al. [84] introduce an obfuscation pipeline secured by a hardware security module. CoreLocker [33] and NNSplitter [32] specifically focus on identifying a small portion of key weights to lock the network, which has been shown to be vulnerable to adaptive attacks [34]. Others leverage PUFs for hardware-specific encryption (e.g., [85]–[89]), similar to ours. However, these works primarily focus on CNNs or MLPs and do not specifically explore transformers, leaving it unclear how well their techniques scale or adapt to the

TABLE 9. Model accuracy (%) when encrypting only attention weights.

| Steps | ViT-B | ViT-B-384 | ViT-B-8b | BeiT-B | DeiT-B | Swin-B |
|---|---|---|---|---|---|---|
| $Acc_0$(%) | 80.32 | 83.91 | 80.24 | 84.54 | 80.20 | 85.14 |
| 1 | 56.46 (11) | 60.65 (11) | 53.64 (11) | 62.67 (1) | 75.69 (10) | 64.36 (11) |
| 2 | 13.87 (10) | 14.37 (10) | 11.35 (10) | 8.23 (2) | 31.33 (11) | 42.16 (10) |
| 3 | 4.87 (9) | 4.83 (9) | 3.93 (9) | 0.87 (3) | 1.04 (9) | 12.01 (9) |
| 4 | 2.01 (8) | 2.11 (8) | 1.61 (8) | 0.50 (6) | 0.15 (8) | 2.09 (8) |
| 5 | 0.97 (7) | 1.01 (0) | 0.74 (7) | 0.30 (8) | 0.10 (7) | 0.63 (7) |
| 6 | 0.46 (0) | 0.45 (7) | 0.30 (0) | 0.13 (5) | 0.09 (5) | 0.30 (0) |
| 7 | 0.21 (6) | 0.22 (6) | 0.20 (6) | – | – | 0.12 (1) |
| 8 | 0.17 (5) | 0.13 (5) | 0.14 (5) | – | – | 0.11 (6) |

The number in parenthesis indicates the layer index. We found that encrypting only attention weights requires more layers to achieve satisfactory performance degradation.

TABLE 10. Impact of extra security layers on retraining ViT-B.

| Num. of Extra Security Layers | Best Top-1 Acc. after Retraining (%) |
|---|---|
| 0 | 69.7 |
| 1 | 68.1 |
| 2 | 58.4 |
| 3 | 54.5 |
| 4 | 56.4 |

The baseline model required two layers to fall below the $\epsilon$ threshold. "Number of Extra Security Layers" refers to the layers added on top of these initial two.

unique architectural components of transformers, such as the attention mechanism.

**TEE-based methods.** TEE (Trusted Execution Environment) can protect models during execution [2], [12]–[15], [31], [65], [90]–[96], permit remote license revocation [14], and even support federated learning [90], [91]. However, their limited execution capability and amount of secure memory allowed (e.g., 128 MB in Intel SGX1 and up to 16 MB for most TrustZone-enabled devices) is insufficient for large models, leading to significant performance overhead due to memory swapping [97]. Partitioning models across TEE and untrusted memory introduces additional attack surfaces (e.g., side-channels [98]–[103]) and complex implementation requirements. Notably, recent proposals also explore multiple GPU TEE designs. However, they either require hardware modification, which is expensive and slow for deployment [104], [105], or significant software adaptation for application-level isolation [106], [107]. In contrast, the PUF design is able to utilize the existing hardware on the device, such as SRAM and DRAM, and it is relatively easy to integrate into the software pipeline.

**Model-level access control.** This line of work incorporates access control or licensing through additional training and grants access to model functionality only to users who possess certain confidential information. Chen and Wu [61] propose adding an additional transform module and using adversarial perturbations for access control; Fan et al. [63] add passport layers, where scale factors and bias terms are functional only when a "passport" (e.g., a fixed personal identification picture) is provided. Data manipulation and backdoor injection [62], [108]–[111] are also common strategies. While these methods are effective, they demand retraining with extra objectives, potentially hindering the model's ability to converge and possibly leading to increased complexity or compromises in accuracy. Furthermore, training separate models for individual users lacks scalability.

**Our focus.** Most of the solutions above have been demonstrated on small networks or do not consider attackers with partial data or fine-tuning capabilities. In contrast, our approach encrypts transformer models using PUF-driven keys without retraining. We adopt a lightweight design that avoids TEE constraints while still providing robust defense under the white-box settings described in Section 3.

## 7. Conclusion and Future Work

We propose an active, hardware-bound IP protection scheme that integrates PUF-based cryptographic key derivation and ACM-based weight obfuscation. The resulting model cannot be executed correctly outside the authorized device, effectively thwarting IP theft and mitigating attacks that rely on stolen models, even under white-box settings. Unlike watermarking or retraining-based schemes, our method imposes no changes to the original training process and adds less than 30% runtime overhead. In our experiments across popular vision transformer models, locking only a subset of layers drove the accuracy down to random levels—and even with 20% of the training data, fine-tuning the encrypted model could not restore its original performance.

Looking ahead, there are three clear paths to strengthen and scale this protection: first, while our current work focuses on Transformers, the core framework is adaptable to other architectures, such as CNNs. For the 4D convolutional kernels, which are typically not square, one could apply a rectangular chaotic map such as the Baker's Map [22] in place of ACM; second, extend and adapt our layer-selection heuristic to billion-parameter models (e.g., Large Language models) so that overhead remains manageable even at that scale; third, while our method only protects the weights and assumes full knowledge of the architecture, we can further integrate architectural obfuscation like [65], [112] to achieve deeper protection [113]–[115]. By offering a low-cost, framework-agnostic lock on valuable model IP, our approach takes a practical step toward securing on-device deep learning, from edge applications to large-scale deployments.

## 8. Acknowledgement

This work was supported in part by the National Natural Science Foundation of China, under grant number 62404192, and in part by Shenzhen Stability Science Program 2023. The authors thank the anonymous reviewers for their constructive feedback on improving our work. P. H. was supported by the Undergraduate Research Award program at The Chinese University of Hong Kong, Shenzhen.

## References

[1] Y. Zheng, C.-H. Chang, S.-H. Huang, P.-Y. Chen, and S. Picek, "An overview of trustworthy ai: Advances in ip protection, privacy-preserving federated learning, security verification, and gai safety alignment," *IEEE J. Emerging and Selected Topics in Circuits and Systems*, 2024.

[2] Z. Zhang, C. Gong, Y. Cai, Y. Yuan, B. Liu, D. Li, Y. Guo, and X. Chen, "No privacy left outside: On the (in-)security of tee-shielded dnn partition for on-device ml," in *2024 IEEE Symposium on Security and Privacy (SP)*. Washington, DC, USA: IEEE Computer Society, 2024, pp. 3327–3345.

[3] M. Shafieinejad, N. Lukas, J. Wang, X. Li, and F. Kerschbaum, "On the robustness of backdoor-based watermarking in deep neural networks," in *Proc. 2021 ACM Workshop on Information Hiding and Multimedia Security*, ser. IH&MMSec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 177–188. [Online]. Available: https://doi.org/10.1145/3437880.3460401

[4] X. Chen, W. Wang, C. Bender, Y. Ding, R. Jia, B. Li, and D. Song, "Refit: A unified watermark removal framework for deep learning systems with limited data," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 321–335. [Online]. Available: https://doi.org/10.1145/3433210.3453079

[5] S. Sun, H. Wang, M. Xue, Y. Zhang, J. Wang, and W. Liu, "Detect and remove watermark in deep neural networks via generative adversarial networks," in *Information Security: 24th International Conference, ISC 2021, Virtual Event, November 10–12, 2021, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 341–357. [Online]. Available: https://doi.org/10.1007/978-3-030-91356-4_18

[6] Z. Jiang, M. Fang, and N. Z. Gong, "Ipcert: Provably robust intellectual property protection for machine learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3612–3621.

[7] Y. Yan, X. Pan, M. Zhang, and M. Yang, "Rethinking White-Box watermarks on deep learning models under neural structural obfuscation," in *Proc. the 32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA, USA: USENIX Association, Aug. 2023, pp. 2347–2364. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/yan

[8] H. Zhang, B. L. Edelman, D. Francati, D. Venturi, G. Ateniese, and B. Barak, "Watermarks in the sand: Impossibility of strong watermarking for generative models," *arXiv preprint arXiv:2311.04378*, 2023.

[9] K. Dang, P. Lai, N. Phan, Y. Shen, R. Jin, A. Khreishah, and M. Thai, "Sok: Are watermarks in llms ready for deployment?" *arXiv preprint arXiv:2506.05594*, 2025.

[10] Q. Pang, S. Hu, W. Zheng, and V. Smith, "Attacking llm watermarks by exploiting their strengths," in *ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*, 2024.

[11] N. Jovanović, R. Staab, and M. Vechev, "Watermark stealing in large language models," *arXiv preprint arXiv:2402.19361*, 2024.

[12] Z. Sun, R. Sun, C. Liu, A. R. Chowdhury, L. Lu, and S. Jha, " ShadowNet: A Secure and Efficient On-device Model Inference System for Convolutional Neural Networks ," in *2023 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 1596–1612. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.10179382

[13] Q. Li, Z. Shen, Z. Qin, Y. Xie, X. Zhang, T. Du, and J. Yin, "TransLinkGuard: Safeguarding Transformer Models Against Model Stealing in Edge Deployment," Apr. 2024.

[14] X. Zhuang, L. Zhang, C. Tang, H. Liu, B. Wang, Y. Zheng, and B. Ren, "Deepcontract: Controllable authorization of deep learning models," in *Proceedings of the 39th Annual Computer Security Applications Conference*, ser. ACSAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 609–620. [Online]. Available: https://doi.org/10.1145/3627106.3627107

[15] B. Hu, Y. Wang, J. Cheng, T. Zhao, Y. Xie, X. Guo, and Y. Chen, "Secure and efficient mobile dnn using trusted execution environments," in *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, 2023, pp. 274–285.

[16] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM transactions on intelligent systems and technology*, vol. 15, no. 3, pp. 1–45, 2024.

[17] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu *et al.*, "A survey on vision transformer," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 1, pp. 87–110, 2022.

[18] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun, "Transformers in time series: A survey," *arXiv preprint arXiv:2202.07125*, 2022.

[19] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, "A survey on large language models for code generation," *arXiv preprint arXiv:2406.00515*, 2024.

[20] J. Zhang, K. Huang, Y. Huang, B. Chen, R. Wang, C. Wang, and X. Peng, "Killing two birds with one stone: Malicious package detection in npm and pypi using a single model of malicious behavior sequence," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 4, pp. 1–28, 2025.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] J. Fridrich, "Symmetric Ciphers Based on Two-Dimensional Chaotic Maps," *Int. J. Bifurcation and Chaos*, vol. 08, no. 06, pp. 1259–1284, Jun. 1998.

[23] L. Chen and S. Wang, "Differential cryptanalysis of a medical image cryptosystem with multiple rounds," *Computers in Biology and Medicine*, vol. 65, pp. 69–75, Oct. 2015.

[24] M. F. A. Elzaher, M. Shalaby, and S. H. El Ramly, "An Arnold Cat Map-Based Chaotic Approach for Securing Voice Communication," in *Proceedings of the 10th International Conference on Informatics and Systems*. Giza Egypt: ACM, May 2016, pp. 329–331.

[25] N. Lin, X. Chen, H. Lu, and X. Li, "Chaotic Weights: A Novel Approach to Protect Intellectual Property of Deep Neural Networks," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1327–1339, Jul. 2021.

[26] H. Tora, E. Gokcay, M. Turan, and M. Buker, "A generalized arnold's cat map transformation for image scrambling," *Multimedia Tools and Applications*, vol. 81, no. 22, pp. 31 349–31 362, September 2022. [Online]. Available: https://doi.org/10.1007/s11042-022-11985-2

[27] Y. Zhang, Z. Hua, H. Bao, H. Huang, and Y. Zhou, "An $n$-dimensional chaotic system generation method using parametric pascal matrix," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 8434–8444, 2022.

[28] Y. Wu, Z. Hua, and Y. Zhou, "$n$-dimensional discrete cat map generation using laplace expansions," *IEEE transactions on cybernetics*, vol. 46, no. 11, pp. 2622–2633, 2015.

[29] Y. Zhang, Z. Hua, H. Bao, H. Huang, and Y. Zhou, "Generation of n-dimensional hyperchaotic maps using gershgorin-type theorem and its application," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 53, no. 10, pp. 6516–6529, 2023.

[30] Y. Cai, X. Chen, L. Tian, Y. Wang, and H. Yang, "Enabling secure in-memory neural network computing by sparse fast gradient encryption," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[31] J. Hou, H. Liu, Y. Liu, Y. Wang, P.-J. Wan, and X.-Y. Li, "Model protection: Real-time privacy-preserving inference service for model privacy at the edge," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4270–4284, 2021.

[32] T. Zhou, Y. Luo, S. Ren, and X. Xu, "Nnsplitter: an active defense solution for dnn model via automated weight obfuscation," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23. Honolulu, Hawaii, USA: JMLR.org, 2023.

[33] Z. Wang, Z. Ma, X. Feng, R. Sun, H. Wang, M. Xue, and G. Bai, " CORELOCKER: Neuron-level Usage Control ," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2024, pp. 2497–2514. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00233

[34] Y. Sun, V. Bonde, L. Duan, and Y. Li, "Obfuscation for deep neural networks against model extraction: Attack taxonomy and defense optimization," in *International Conference on Applied Cryptography and Network Security*. Springer, 2025, pp. 391–414.

[35] C.-H. Chang, Y. Zheng, and L. Zhang, "A retrospective and a look forward: Fifteen years of physical unclonable function advancement," *IEEE Circuits and Systems Magazine*, vol. 17, no. 3, pp. 32–62, 2017.

[36] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Controlled physical random functions," in *18th Annual Computer Security Applications Conference, 2002. Proceedings.* Los Alamitos, CA, USA: IEEE Computer Society, 2002, pp. 149–160.

[37] B. Gassend, M. V. Dijk, D. Clarke, E. Torlak, S. Devadas, and P. Tuyls, "Controlled physical random functions and applications," *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 4, Jan. 2008. [Online]. Available: https://doi.org/10.1145/1284680.1284683

[38] H. Kang, Y. Hori, T. Katashita, M. Hagiwara, and K. Iwamura, "Cryptographie key generation from puf data using efficient fuzzy extractors," in *16th International conference on advanced communication technology*. IEEE, 2014, pp. 23–26.

[39] J. Delvaux, D. Gu, I. Verbauwhede, M. Hiller, and M.-D. Yu, "Efficient fuzzy extraction of puf-induced secrets: Theory and applications," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 412–431.

[40] Z. Sun, R. Sun, L. Lu, and A. Mislove, "Mind your weight(s): A large-scale study on insufficient machine learning model protection in mobile apps," in *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*. Vancouver, BC, Canada: USENIX Association, August 2021, pp. 1955–1972. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/sun-zhichuang

[41] Z. Deng, K. Chen, G. Meng, X. Zhang, K. Xu, and Y. Cheng, "Understanding real-world threats to deep learning models in android apps," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 785–799. [Online]. Available: https://doi.org/10.1145/3548606.3559388

[42] D. Oygenblik, C. Yagemann, J. Zhang, A. Mastali, J. Park, and B. Saltaformaggio, "AI psychiatry: Forensic investigation of deep learning networks in memory images," in *Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA, USA: USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/oygenblik

[43] S. Lindenlauf, H. Höfken, and M. Schuba, "Cold boot attacks on ddr2 and ddr3 sdram," in *2015 10th International Conference on Availability, Reliability and Security (ARES)*. Washington, DC, USA: IEEE Computer Society, 2015, pp. 287–292.

[44] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, p. 91–98, May 2009. [Online]. Available: https://doi.org/10.1145/1506409.1506429

[45] T. Müller and M. Spreitzenbarth, "Frost," in *Applied Cryptography and Network Security*, M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 373–388.

[46] M. Gruhn and T. Müller, "On the practicability of cold boot attacks," in *2013 International Conference on Availability, Reliability and Security*. Washington, DC, USA: IEEE Computer Society, 2013, pp. 390–397.

[47] M. K. Hasan, Z. Weichen, N. Safie, F. R. A. Ahmed, and T. M. Ghazal, "A survey on key agreement and authentication protocol for internet of things application," *IEEE access*, 2024.

[48] H. Krawczyk, "Cryptographic extraction and key derivation: The hkdf scheme," in *Annual Cryptology Conference*. Springer, 2010, pp. 631–648.

[49] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Addison-Wesley, 1997.

[50] B. Basak, N. Patil, K. Polachan, and S. Vivek, "Attack on a puf-based secure binary neural network," *arXiv preprint arXiv:2510.24422*, 2025.

[51] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[52] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. Washington, DC, USA: IEEE Computer Society, 2021, pp. 9992–10 002.

[53] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," 2021. [Online]. Available: https://arxiv.org/abs/2012.12877

[54] H. Bao, L. Dong, S. Piao, and F. Wei, "Beit: Bert pre-training of image transformers," 2022. [Online]. Available: https://arxiv.org/abs/2106.08254

[55] Z. Zhang, N. Wang, Z. Zhang, Y. Zhang, T. Zhang, J. Liu, and Y. Wu, "Groupcover: a secure, efficient and scalable inference framework for on-device model protection based on tees," in *Forty-first international conference on machine learning*, 2024.

[56] Y. Yarom and K. Falkner, "Flush+Reload: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[57] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-Level Cache Side-Channel Attacks are Practical," in *2015 IEEE Symposium on Security and Privacy*. San Jose, CA: IEEE, May 2015, pp. 605–622.

[58] A. S. Rakin, M. H. I. Chowdhury, F. Yao, and D. Fan, "Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories," in *2022 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, 2022, pp. 1157–1174.

[59] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding Watermarks into Deep Neural Networks," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*. Bucharest, Romania: ACM, 2017, pp. 269–277. [Online]. Available: https://dl.acm.org/doi/10.1145/3078971.3078974

[60] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: watermarking deep neural networks by backdooring," in *Proceedings of the 27th USENIX Conference on Security Symposium*, ser. SEC'18. USA: USENIX Association, 2018, p. 1615–1631.

[61] M. Chen and M. Wu, "Protect your deep neural networks from piracy," in *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. Hong Kong, China: IEEE, 2018, pp. 1–7.

[62] G. Ren, J. Wu, G. Li, S. Li, and M. Guizani, "Protecting Intellectual Property With Reliable Availability of Learning Models in AI-Based Cybersecurity Services," *IEEE Trans. Dependable and Secure Computing*, vol. 21, no. 2, pp. 600–617, Mar. 2024.

[63] L. Fan, K. W. Ng, C. S. Chan, and Q. Yang, "DeepIPR: Deep Neural Network Ownership Verification With Passports," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 6122–6139, Oct. 2022.

[64] M. Alam, S. Saha, D. Mukhopadhyay, and S. Kundu, "*NN-Lock* : A Lightweight Authorization to Prevent IP Threats of Deep Learning Models," *ACM J. Emerging Technologies in Computing Systems*, vol. 18, no. 3, pp. 1–19, Jul. 2022.

[65] J. Bai, M. H. I. Chowdhuryy, J. Li, F. Yao, C. Chakrabarti, and D. Fan, "Phantom: Privacy-preserving deep neural network model obfuscation in heterogeneous tee and gpu system," in *25th USENIX Security Symposium (USENIX Security 16)*, 2025.

[66] P. Ren, C. Zuo, X. Liu, W. Diao, Q. Zhao, and S. Guo, "Demistify: Identifying on-device machine learning models stealing and reuse vulnerabilities in mobile apps," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[67] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[68] K. Patwari, S. M. Hafiz, H. Wang, H. Homayoun, Z. Shafiq, and C.-N. Chuah, "Dnn model architecture fingerprinting attack on cpu-gpu edge devices," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2022, pp. 337–355.

[69] X. Cao, J. Jia, and N. Z. Gong, "Ipguard: Protecting intellectual property of deep neural networks via fingerprinting the classification boundary," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 14–25.

[70] J. Zhao, Q. Hu, G. Liu, X. Ma, F. Chen, and M. M. Hassan, "Afa: Adversarial fingerprinting authentication for deep neural networks," *Computer Communications*, vol. 150, pp. 488–497, 2020.

[71] S. Wang, X. Wang, P.-Y. Chen, P. Zhao, and X. Lin, "Characteristic examples: High-robustness, low-transferability fingerprinting of neural networks," in *International Joint Conferences on Artificial Intelligence Organization (IJCAI)*, 2021.

[72] Y. Zheng, S. Wang, and C.-H. Chang, "A dnn fingerprint for non-repudiable model ownership identification and piracy detection," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2977–2989, 2022.

[73] A. Yan, H. Ren, K. Mo, Z. Zhang, S. Wang, and J. Li, "Enhancing model intellectual property protection with robustness fingerprint technology," *IEEE Transactions on Information Forensics and Security*, 2025.

[74] H. Liu, Z. Weng, and Y. Zhu, "Watermarking deep neural networks with greedy residuals." in *ICML*, vol. 139, 2021, pp. 6978–6988.

[75] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *Proceedings of the 2018 on Asia conference on computer and communications security*, 2018, pp. 159–172.

[76] X. Fan, D. Fu, H. Gui, X. Zhang, and X. Zhou, "Pcpt and acpt: Copyright protection and traceability scheme for dnn models," 2023. [Online]. Available: https://arxiv.org/abs/2206.02541

[77] B. Darvish Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 485–497.

[78] H. Li, E. Wenger, S. Shan, B. Y. Zhao, and H. Zheng, "Piracy resistant watermarks for deep neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1910.01226

[79] S. Shao, Y. Li, H. Yao, Y. He, Z. Qin, and K. Ren, "Explanation as a watermark: Towards harmless and multi-bit model ownership verification via watermarking feature attribution," *arXiv preprint arXiv:2405.04825*, 2024.

[80] Y. Huang, Z. Zhang, Q. Zhao, X. Yuan, and C. Chen, "Themis: Towards practical intellectual property protection for post-deployment on-device deep learning models," in *34th USENIX security symposium (USENIX Security 25)*, 2025.

[81] A. Pegoraro, C. Segna, K. Kumari, and A.-R. Sadeghi, "{DeepEclipse}: How to break {White-Box}{DNN-Watermarking} schemes," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5287–5304.

[82] Z. Fei, B. Yi, J. Geng, R. He, L. Nie, and Z. Liu, "Your fixed watermark is fragile: Towards semantic-aware watermark for eaas copyright protection," *arXiv preprint arXiv:2411.09359*, 2024.

[83] R. Mukherjee and R. S. Chakraborty, "Attacks on Recent DNN IP Protection Techniques and Their Mitigation," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 3642–3650, Nov. 2023.

[84] B. F. Goldstein, V. C. Patil, V. C. Ferreira, A. S. Nery, F. M. G. França, and S. Kundu, "Preventing dnn model ip theft via hardware obfuscation," *IEEE J. Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 267–277, 2021.

[85] Q. Guo, J. Ye, Y. Gong, Y. Hu, and X. Li, "Puf based pay-per-device scheme for ip protection of cnn model," in *Proc. 2018 IEEE 27th Asian Test Symposium (ATS)*. Hefei, China: IEEE, October 2018, pp. 115–120, presented at the 2018 IEEE 27th Asian Test Symposium (ATS), October 15–18, Hefei, China. [Online]. Available: https://ieeexplore.ieee.org/document/8567421

[86] D. Li, Y. Ren, D. Liu, Y. Guo, Z. Guan, and J. Liu, "Pipp: A practical puf-based intellectual property protection scheme for dnn model on fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 2, pp. 912–916, 2024.

[87] Q. Pan, M. Dong, K. Ota, and J. Wu, "Device-bind key-storageless hardware ai model ip protection: A puf and permute-diffusion encryption-enabled approach," 2022. [Online]. Available: https://arxiv.org/abs/2212.11133

[88] J. Jiang, Y. Zheng, and C.-H. Chang, "Puf-based edge dnn model ip protection with self-obfuscation and publicly verifiable ownership," in *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2025, pp. 1–5.

[89] G. Rajendran, D. Basak, S. Deb, and A. Chattopadhyay, "Securing binarized neural networks via puf-based key management in memristive crossbar arrays," *IEEE Embedded Systems Letters*, vol. 17, no. 1, pp. 30–33, 2024.

[90] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious {Multi-Party} machine learning on trusted processors," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 619–636.

[91] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, "Ppfl: Privacy-preserving federated learning with trusted execution environments," in *Proceedings of the 19th annual international conference on mobile systems, applications, and services*, 2021, pp. 94–108.

[92] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "Darknetz: towards model privacy at the edge using trusted execution environments," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 161–174.

[93] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang, and J. Song, "Occlumency: Privacy-preserving remote deep-learning inference using sgx," in *The 25th Annual international conference on mobile computing and networking*, 2019, pp. 1–17.

[94] T. Shen, J. Qi, J. Jiang, X. Wang, S. Wen, X. Chen, S. Zhao, S. Wang, L. Chen, X. Luo *et al.*, "{SOTER}: Guarding black-box inference for general neural networks at the edge," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 723–738.

[95] M. Moon, M. Kim, J. Jung, and D. Song, "Asgard: Protecting on-device deep neural networks with virtualization-based trusted execution environments," in *Proceedings 2025 Network and Distributed System Security Symposium*, 2025.

[96] Z. Liu, Y. Luo, S. Duan, T. Zhou, and X. Xu, "Mirrornet: A tee-friendly framework for secure on-device dnn inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[97] J. Wang, Y. Wu, H. Liu, B. Yuan, R. Chamberlain, and N. Zhang, "Ip protection in tinyml," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, 2023, pp. 1–6.

[98] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems," in *2020 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, 2020, pp. 1416–1432.

[99] X. Zhang, J. Wang, Y. Cheng, Q. Li, K. Sun, Y. Zheng, N. Zhang, and X. Li, "Interface-based side channel in tee-assisted networked services," *IEEE/ACM Transactions on Networking*, vol. 32, no. 1, pp. 613–626, 2024.

[100] Y. Yuan, Z. Liu, S. Deng, Y. Chen, S. Wang, Y. Zhang, and Z. Su, "Hypertheft: Thieving model weights from tee-shielded neural networks via ciphertext side channels," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4346–4360.

[101] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[102] Z. Kou, S. Sinha, W. He, and W. Zhang, "Cache side-channel attacks and defenses of the sliding window algorithm in tees," in *Proceedings of the 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Antwerp, Belgium: IEEE, 2023, pp. 1–6.

[103] Y. Yuan, Z. Liu, S. Deng, Y. Chen, S. Wang, Y. Zhang, and Z. Su, "Ciphersteal: Stealing input data from tee-shielded neural networks with ciphertext side channels," in *Proceedings of the 2025 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE Computer Society, 2024, pp. 79–79. [Online]. Available: https://www.ieee-security.org/TC/SP2025/

[104] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity gpus," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 455–468.

[105] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on {GPUs}," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 681–696.

[106] X. Wu, D. J. Tian, and C. H. Kim, "Building GPU TEEs using CPU Secure Enclaves with GEVisor," in *Proceedings of the 14th ACM Symposium on Cloud Computing (SOCC 2023)*, Santa Cruz, CA, Oct. 2023. [Online]. Available: https://doi.org/10.1145/3620678.3624659

[107] H. Mai, J. Zhao, H. Zheng, Y. Zhao, Z. Liu, M. Gao, C. Wang, H. Cui, X. Feng, and C. Kozyrakis, "Honeycomb: Secure and efficient {GPU} executions via static validation," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 155–172.

[108] P. Li, J. Huang, H. Wu, Z. Zhang, and C. Qi, "SecureNet: Proactive intellectual property protection and model security defense for DNNs based on backdoor learning," *Neural Networks*, vol. 174, p. 106199, Jun. 2024.

[109] J. Tian, J. Zhou, and J. Duan, " Probabilistic Selective Encryption of Convolutional Neural Networks for Hierarchical Services ," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2021, pp. 2205–2214. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.00224

[110] M. Xue, Z. Wu, Y. Zhang, J. Wang, and W. Liu, "AdvParams: An Active DNN Intellectual Property Protection Technique via Adversarial Perturbation Based Parameter Encryption," *IEEE Trans. Emerging Topics in Computing*, vol. 11, no. 3, pp. 664–678, Jul. 2023.

[111] J. Chen, H. Zheng, T. Liu, J. Liu, Y. Cheng, X. Zhang, and S. Ji, " EdgePro: Edge Deep Learning Model Protection via Neuron Authorization ," *IEEE Trans. Dependable and Secure Computing*, vol. 21, no. 05, pp. 4967–4981, Sep. 2024. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/TDSC.2024.3365730

[112] J. Li, Z. He, A. S. Rakin, D. Fan, and C. Chakrabarti, "Neurobfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 248–258.

[113] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood *et al.*, "Deepsniffer: A dnn model extraction framework based on learning architectural hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 385–399.

[114] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. Al Faruque, "Stealing neural network structure through remote fpga side-channel analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4377–4388, 2021.

[115] Y. Gao, H. Qiu, Z. Zhang, B. Wang, H. Ma, A. Abuadbba, M. Xue, A. Fu, and S. Nepal, "Deeptheft: Stealing dnn model architectures through power side channel," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3311–3326.

[116] Z. Hua and Y. Zhou, "Exponential chaotic model for generating robust chaos," *IEEE transactions on systems, man, and cybernetics: systems*, vol. 51, no. 6, pp. 3713–3724, 2019.

## Appendix

This appendix provides a formal proof that Arnold's Cat Map (ACM), as utilized in our weight obfuscation scheme, is a chaotic system from a mathematical standpoint, disregarding the limitations of finite-precision digital domains.

### 1. Fundamental Definitions

We first establish the mathematical framework for our proof by defining a chaotic system and the key metric used to identify it, the Lyapunov exponent, following [27], [29].

**Definition 1** (Chaotic System [27])**.** *A discrete dynamical system is defined as* ***chaotic*** *if it satisfies two fundamental conditions:*

1) *Its dynamics are confined to a* ***globally bounded phase space****.*
2) *It exhibits sensitivity to initial conditions, which is mathematically characterized by the presence of at least one* ***positive Lyapunov exponent****.*

The practical meaning of the second condition is profound for cryptography and obfuscation. A positive Lyapunov exponent implies that two trajectories starting arbitrarily close to each other will diverge exponentially.

This property is known as the **avalanche effect**, where a minuscule change in the system's initial state (or, in our case, the secret key) results in a completely different and unpredictable output trajectory (the obfuscated weights).

**Definition 2** (Lyapunov Exponent [29]). *For a discrete n-dimensional map $F$ given by $\mathbf{x}_{i+1} = F(\mathbf{x}_i)$, the Lyapunov exponents, $\lambda_k$, are calculated as:*

$$\lambda_k = \lim_{t\to\infty} \frac{1}{t} \ln(|\mu_k|) \tag{9}$$

*where $\mu_k$ is the $k_{th}$ eigenvalue of the matrix $J_t = J(\mathbf{x}_{t-1})\dots J(\mathbf{x}_1)J(\mathbf{x}_0)$, and $J(\mathbf{x}_i)$ is the Jacobian matrix of the map $F$ evaluated at state $\mathbf{x}_i$. For a linear map where the Jacobian is a constant matrix $A$, this simplifies to $\lambda_k = \ln(|\mu_k|)$, where $\mu_k$ is the $k_{th}$ eigenvalue of $A$.*

## 2. Proof

We now formally prove that the ACM satisfies both conditions for chaos.

**Theorem 1.** *The general Arnold's Cat Map, defined by the transformation matrix $A = \begin{bmatrix} 1 & p \\ q & pq+1 \end{bmatrix}$, for any positive integers $p, q$, is a chaotic system.*

*Proof.* We will prove the theorem by demonstrating that both conditions from Definition 1 are met.

**2.1. Condition 1: Globally Bounded Phase Space.** The ACM transformation is defined on a two-dimensional torus, which can be represented by the unit square $[0, S) \times [0, S)$. The governing equation is:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & p \\ q & pq+1 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \pmod S \tag{10}$$

The modulo $S$ operation ensures that for any state vector $(x_n, y_n)$ within this square, the subsequent state $(x_{n+1}, y_{n+1})$ is also mapped back into the same finite region. This space is, by definition, bounded. Thus, the first condition is satisfied.

**2.2. Condition 2: Positive Lyapunov Exponent.** To satisfy the second condition, we must show that the transformation matrix $A$ has at least one eigenvalue with a magnitude greater than 1.

The map is linear, so its Jacobian matrix is the constant matrix $A$. According to Definition 2, the Lyapunov exponents are determined by the natural logarithm of the magnitudes of the eigenvalues of $A$. The eigenvalues, $\mu$, are the roots of the characteristic equation $\det(A - \mu I) = 0$.

$$\det\left(\begin{bmatrix} 1-\mu & p \\ q & pq+1-\mu \end{bmatrix}\right) = 0 \tag{11}$$

Expanding the determinant yields:

$$(1-\mu)(pq+1-\mu) - pq = 0 \tag{12}$$

$$pq + 1 - \mu - pq\mu - \mu + \mu^2 - pq = 0 \tag{13}$$

$$\mu^2 - (2+pq)\mu + 1 = 0 \tag{14}$$

We solve for the eigenvalues $\mu$ using the quadratic formula:

$$\mu = \frac{(2+pq) \pm \sqrt{(2+pq)^2 - 4}}{2} \tag{15}$$

$$= \frac{(2+pq) \pm \sqrt{4pq + (pq)^2}}{2} \tag{16}$$

Let us analyze the larger eigenvalue, $\mu_1$:

$$\mu_1 = \frac{(2+pq) + \sqrt{4pq + (pq)^2}}{2} \tag{17}$$

For any positive integers $p, q \geq 1$, the product $pq \geq 1$. Consequently, the term under the square root, $4pq + (pq)^2$, is strictly positive. This allows us to establish a lower bound for $\mu_1$:

$$\mu_1 > \frac{2+pq}{2} = 1 + \frac{pq}{2} \tag{18}$$

Since $pq \geq 1$, it follows that $\frac{pq}{2} \geq 0.5$, and therefore:

$$\mu_1 > 1.5 \tag{19}$$

We have proven that the magnitude of the larger eigenvalue, $|\mu_1|$, is strictly greater than 1. The corresponding Lyapunov exponent, $\lambda_1$, is:

$$\lambda_1 = \ln(|\mu_1|) > \ln(1.5) > 0 \tag{20}$$

Since we have found at least one positive Lyapunov exponent, the second condition for chaos is satisfied.

□

For completeness, the product of the eigenvalues is $\mu_1\mu_2 = 1$. Since $\mu_1 > 1$, the second eigenvalue $0 < \mu_2 < 1$ yields a negative Lyapunov exponent $\lambda_2 = \ln(|\mu_2|) < 0$, which corresponds to a contracting direction necessary to keep the system bounded. Therefore, our method is *not* hyper-chaotic [29], where more than one Lyapunov exponent is positive.

While our proof demonstrates that the ACM is mathematically chaotic, any digital implementation operates in a finite state space. Consequently, the transformation is ultimately periodic and not strictly chaotic in the mathematical sense, a phenomenon known as **chaos degradation** [116].

However, we argue that, for the purposes of cryptographic security, this distinction is inconsequential. The cycle periods of the map are designed to be astronomically long, rendering the periodicity practically unobservable. Crucially, the system retains its sensitivity to initial conditions. Therefore, the digital implementation of ACM serves as a secure and effective pseudorandom permutation for our IP protection framework.