Securing On-device Transformer with Hardware Binding and Reversible Obfuscation

Peichun Hua[1], Hanxiu Zhang[1], Tuo Li[2], and Yue Zheng[1]

1 The Chinese University of Hong Kong, Shenzhen

2 Shandong Yunhai Guochuang Cloud Computing Equipment Industry Innovation Co., Ltd

Annual Computer Security Applications Conference (ACSAC 2025)

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

ACSAC
WWW.ACSAC.ORG

ADVANCING CYBERSECURITY SINCE 1985
ACSA
APPLIED COMPUTER SECURITY ASSOCIATES

Background

Deep learning models are considered valuable assets of the owner:

- Proprietary Training Dataset
- Compute Power
- Training Expertise

Deep learning at the edge:

- Real-time processing with low latency
- Reduced cloud dependence
- Powerful functionality without internet connectivity
- Enhanced user privacy

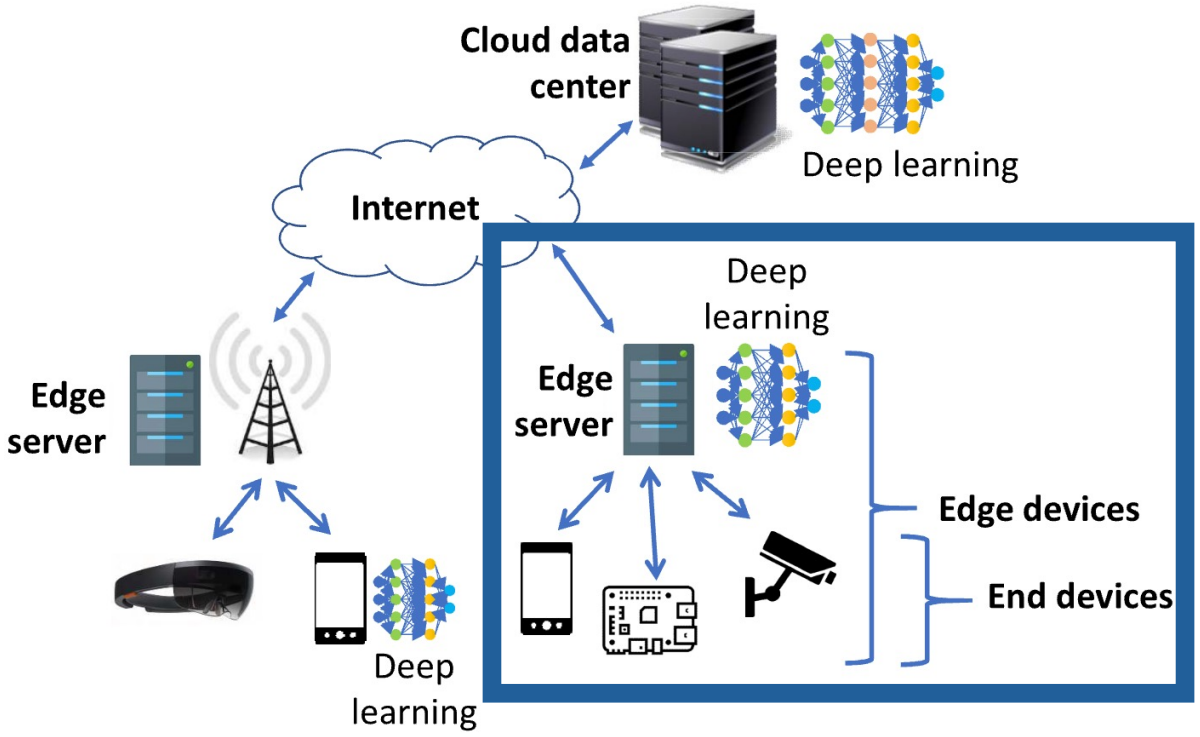


Figure: Chen, Jiasi, and Xukan Ran. "Deep learning with edge computing: A review." Proceedings of the IEEE 107.8 (2019): 1655-1674.

Motivation

Research Question: How to protect the **IP** (Intellectual Property) of a **transformer** model when it is **deployed at the edge**, potentially controlled by malicious parties?

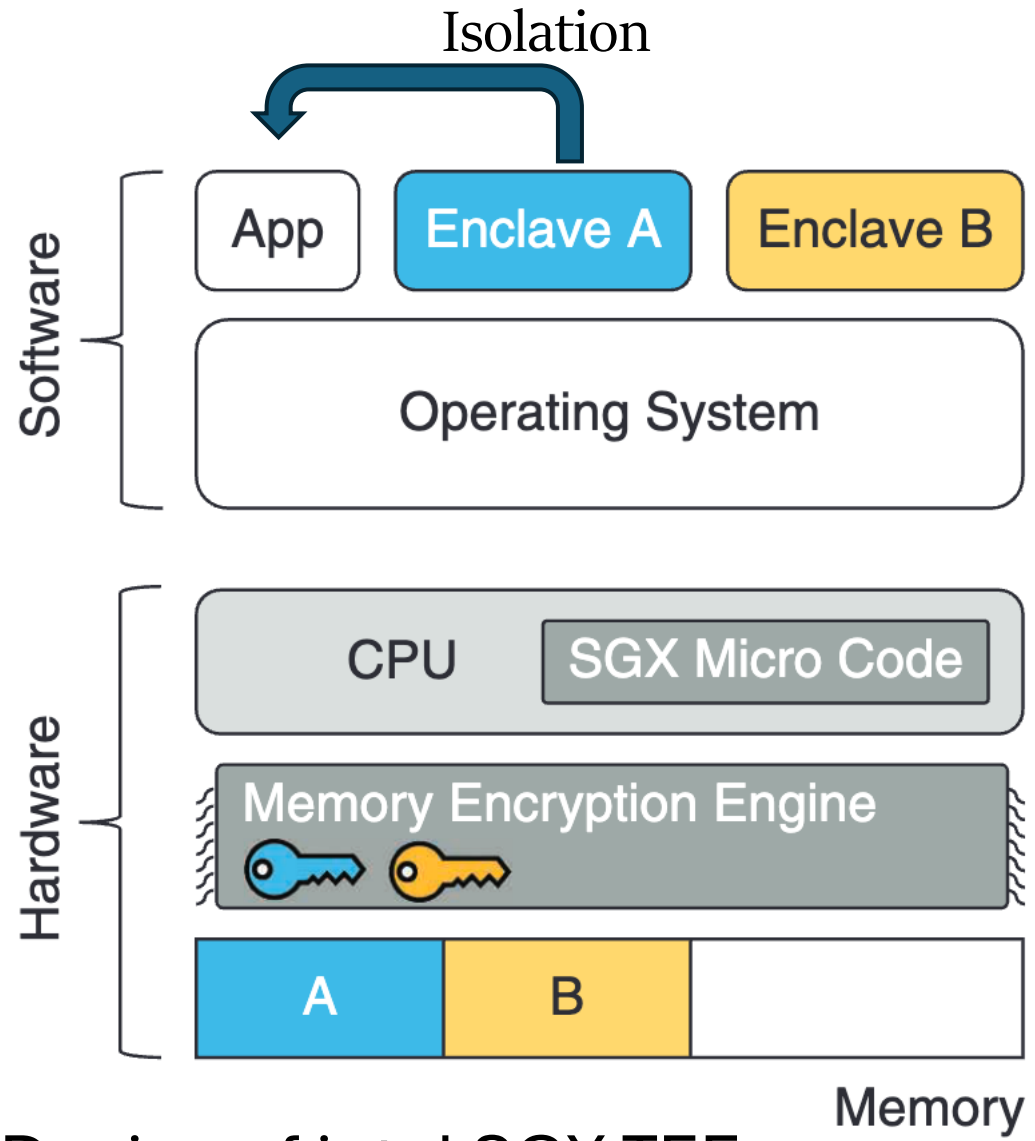
- On-device Model Extraction Attacks [Sun et al., 2021; Deng et al., 2022, Oygenblik et al., 2024]
  - Model De-compilation; Memory Search; API hooking
- Sun et al., 2021: 41% APPs leave plaintext weights from popular ML frameworks (e.g. PyTorch)
- Oygenblik et al., 2024: Recover weights from memory snapshots
- **Threats: Model Inversion Attacks on the Stolen Model, etc.**

Sun, Zhichuang, et al. “Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps.” *30th USENIX security symposium.* 2021.
Deng, Zizhuang, et al. “Understanding real-world threats to deep learning models in android apps.” Proceedings of the *2022 ACM SIGSAC CCS.* 2022.
Oygenblik, David, et al. “AI Psychiatry: Forensic Investigation of Deep Learning Networks in Memory Images.” 33rd USENIX Security Symposium. 2024.

Related Works

Solution 1:

- With TEE (Trusted Execution Environment)
  - Execute something on your machine
  - But you don't know what it is
- Problem:
  - ARM TrustZone: small memory size – not for ML workloads
  - Commercially Built on CPU – significant overhead compared to GPU baseline (5-10 x)



Design of intel SGX TEE.

Figure: Jauernig, Patrick, et al. "Trusted execution environments: properties, applications, and challenges." IEEE Security & Privacy 18.2 (2020): 56-60.

Related Works

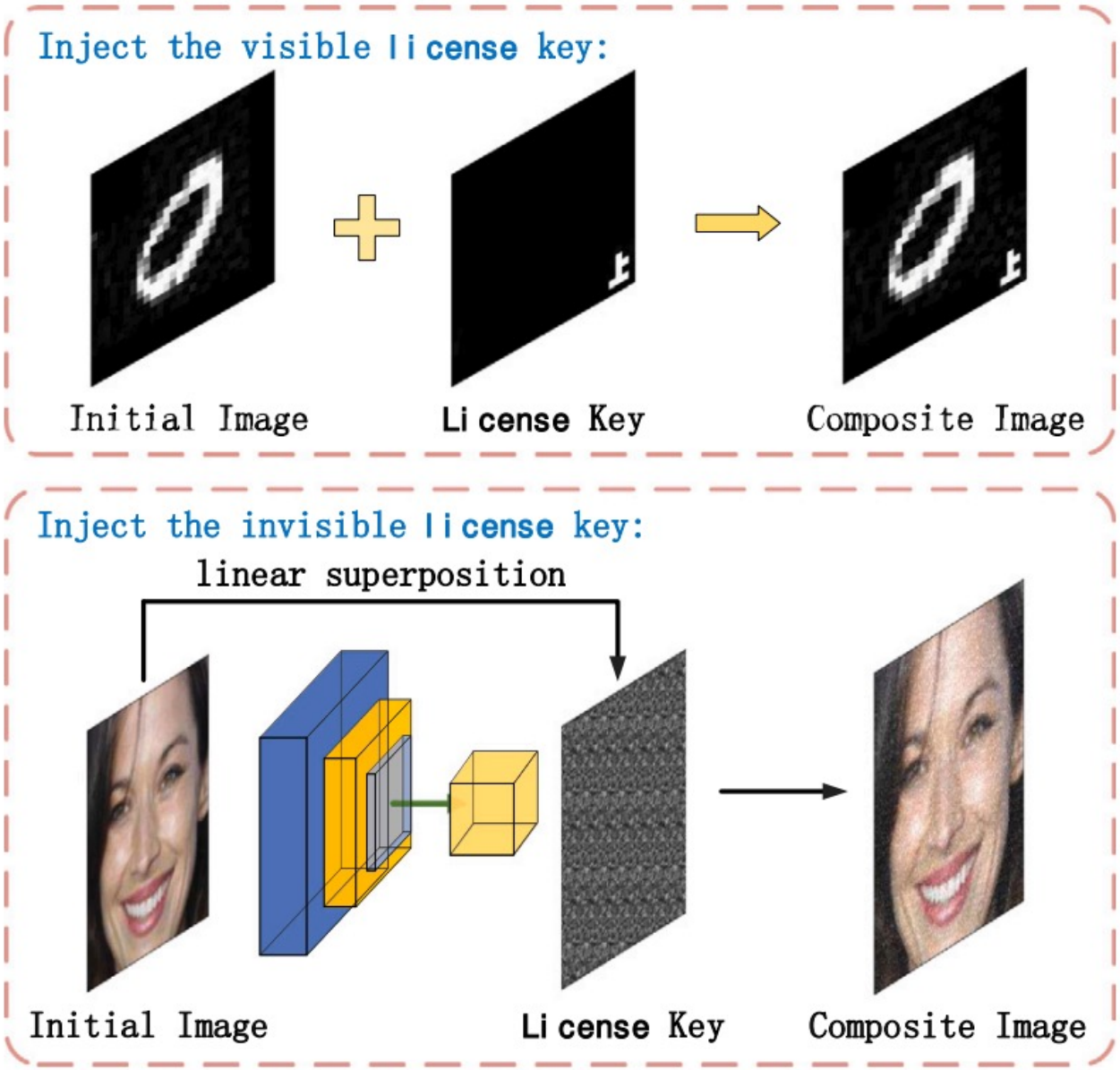
Solution 2: Ownership Verification

- **Watermark:** Embed hidden information into the model
- **Fingerprint:** Extract model-specific features
- Both:
  - **Passive** protection: not prevent misuse
  - **Impractical:**
    - Verification requires access to the suspected model
    - No trusted third party for verifying ownership
  - **Prone to Attacks**
    - Ambiguity Attacks
    - Removal Attacks

Related Works

Solution 3: Active Access Control

- **Retraining needed**
- e.g. only an image with a certain tag can be predicted correctly
- Problem:
  - **Reverse-engineering**
  - Application to other tasks (e.g. segmentation)



Li, Peihao, et al. "SecureNet: Proactive intellectual property protection and model security defense for DNNs based on backdoor learning." *Neural Networks* 174 (2024): 106199.

Related Work

Solution 4: GPU TEE

1. Mostly need host CPU TEE
2. Expensive Hardware (e.g. H100)
3. Significant HW/SW Changes

Existing systems	Host TCB	Required changes	
		HW	SW
Graviton [16]	Intel SGX + ○	SC	Runtime, drivers, CUDA
HIX [17]	Intel SGX + ●	SGX instruction, MMU, PCIe	GPU enclave, inter-enclave communication, CUDA
GraphcoreIPU [18]	○	CCU	XLA, poplar compiler, runtime
NvidiaCC (H100) [15]	C-VM + ●	Security processor	CUDA, C-VM
Apple PCC [39]	Enclave + PCC-OS + TXM [40] + ●	Custom Apple Silicon	SepOS, SW stack
sNPU [24]	Penglai Enclave + ◑ + SM	SoC-NoC, SC	SMMU, SM
StrongBox [25]	TrustZone + ◑ + SM	-	Runtime, driver, MMIO, TASK protector
Honeycomb [41]	AMD-SEV + ○ + Validator + SM	-	Validator, SVSM, SM, runtime
CAGE [23]	ARM CCA + ◑ + SM	-	API, monitors, ShadowTask
ACAI [22]	ARM CCA + ● + PCIe port	-	TF-A, SMMU, RMM
GR-T [26]	Trustzone + ● + cloudVM	-	Driver Shim, GPU_shim
HETEE [42]	● + SM	HETEE box, PCIe interconnect, (SC)	SC, API
ShEF [20]	○	-	ShEF runtime, Shield
GuardAIn	○	-	Driver, runtime, kernels (operators)

Table from: Dhar, Aritra, et al. "GuardAIn: Protecting Emerging Generative AI Workloads on Heterogeneous NPU." 2025 IEEE Symposium on Security and Privacy (SP). IEEE, 2025.

Related Work

Weakness of previous works:

- **Watermark/Fingerprint**: No Active Protection
- **Watermark**: Apply only on small networks (e.g. need retraining)
- **Active Access Control**: Require retraining the model, not scalable
- **CPU-TEE Protection**: Optimize compared to CPU baseline
- **CPU-/GPU-TEE Architecture**: Need significant changes of the whole stack

Design Goal:

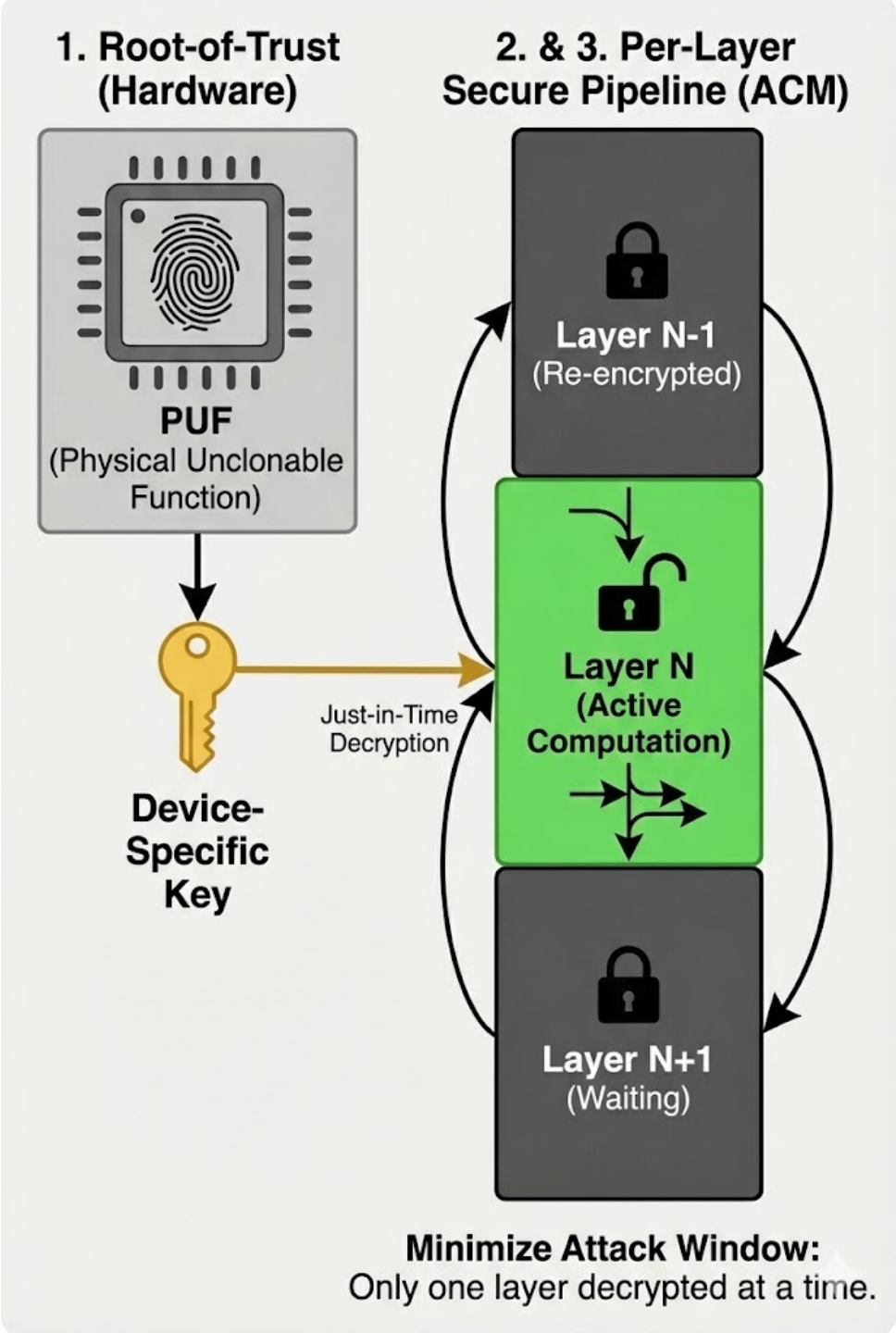
- Actively protect the model to **prevent stealing**
- Make good use of accelerators
- Not change the HW/SW significantly
- Lightweight and efficient

Threat Model

- **Attacker Can:**
  - Access **encrypted model weights** stored on disk.
  - Know the full model architecture.
  - Possess **partial training data** (e.g. up to 20% for ImageNet).
  - Perform **offline computation** to try recovering the model.
- **Attacker Cannot:**
  - Access **runtime plaintext weights** in **memory**.
  - Physically tamper with the hardware and extract keys from the root-of-trust.

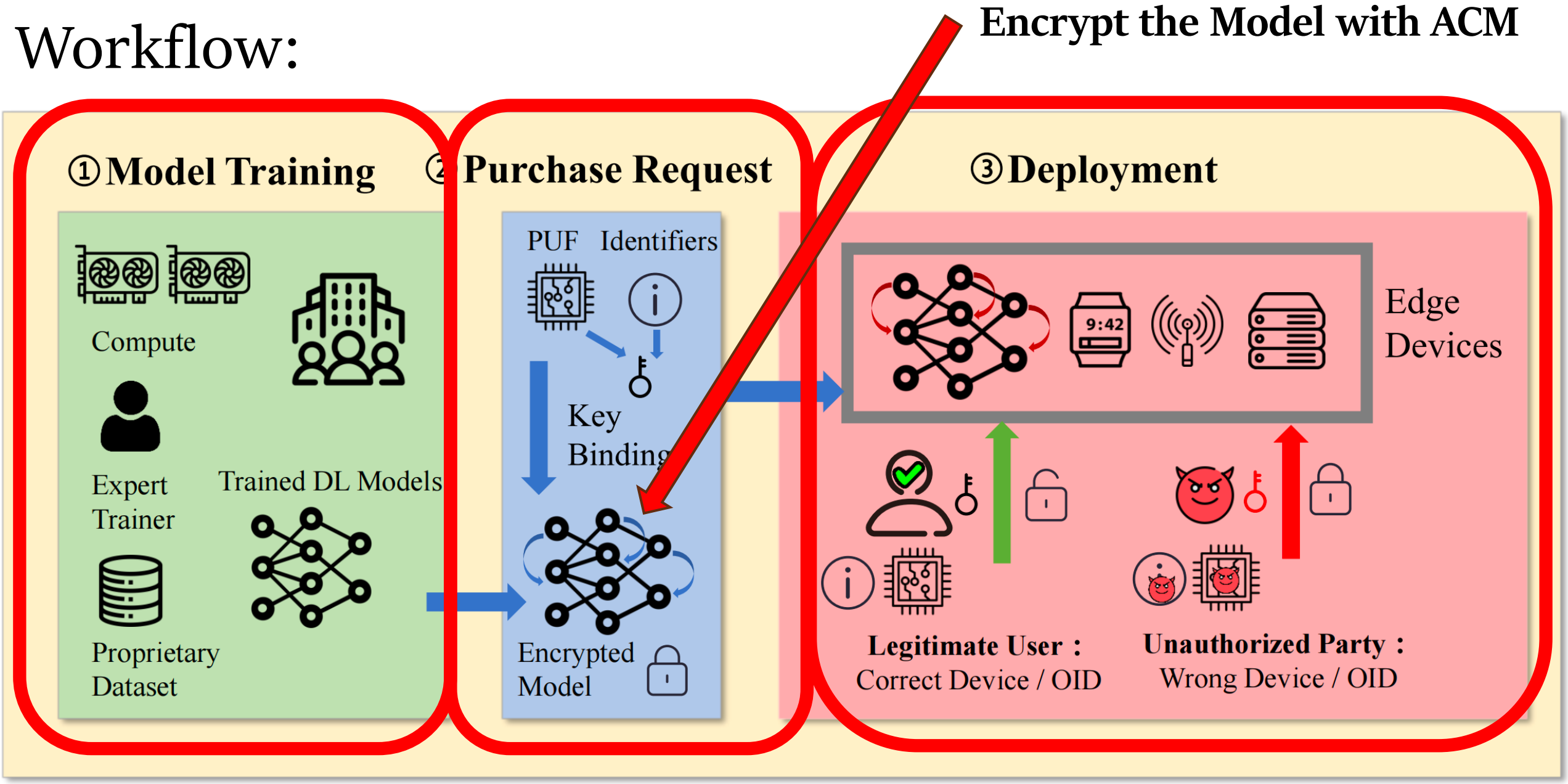
Overview

1. **Physical Unclonable Functions (PUF)** as a Root-of-Trust (RoT)
   - Lightweight, Stable
2. **Arnold Cat's Map (ACM)** for better efficiency
   - Weights have high redundancy, like images
   - Faster than AES on CPU/GPU
3. **Per-layer Decryption/Encryption** Pipeline:
   - Minimize Attack Window



Overview

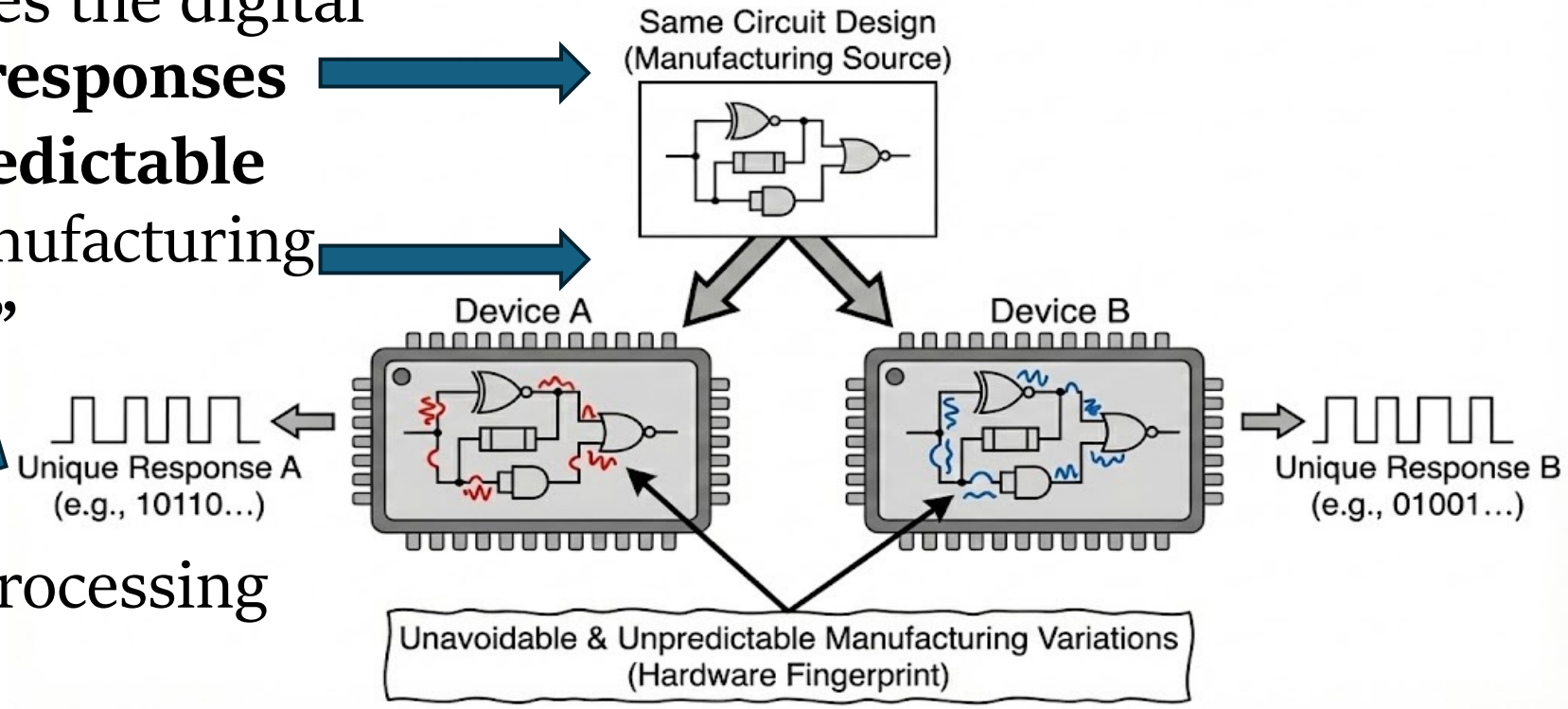
Workflow:



Methodology

Design Choice 1: PUF

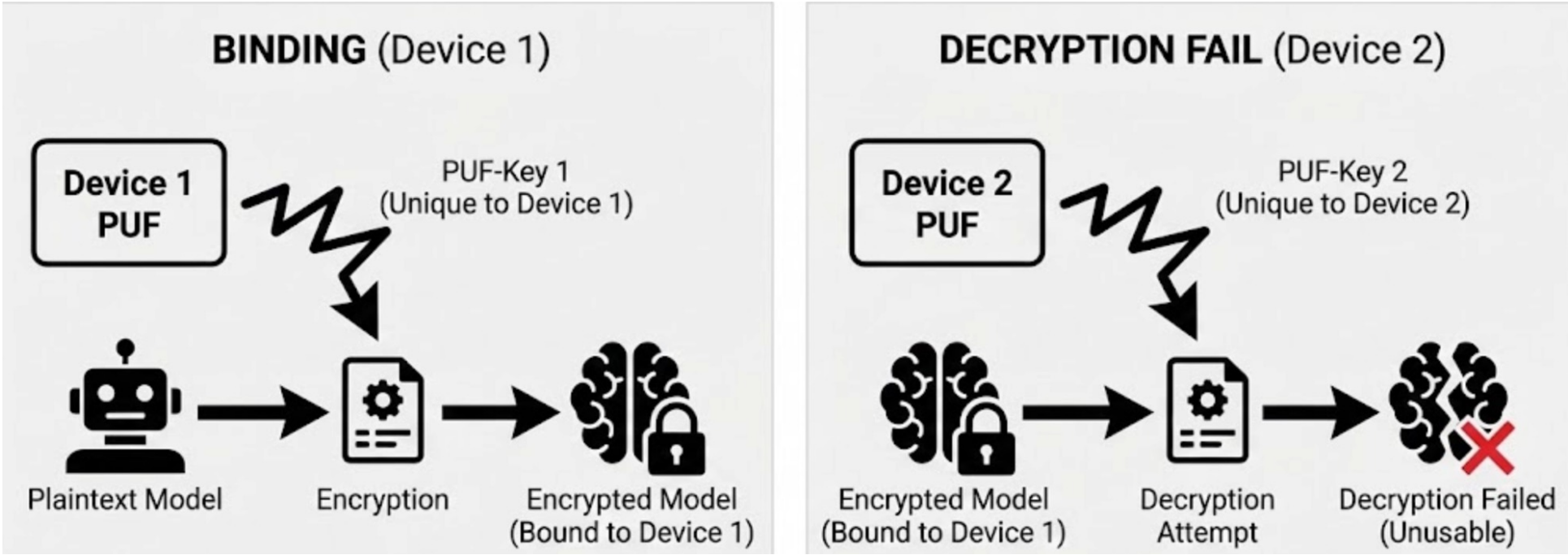
1. A circuit design that takes the digital **challenges** and output **responses**
2. **Unavoidable** and **Unpredictable** variations during the manufacturing
3. **"Hardware Fingerprint"**
4. **Advantages:**
   - Lightweight
   - Reliable (with Post-processing such as ECC)
   - Some designs don't require extra hardware
     - E.g. Memory PUFs with existing DRAM and SRAM



Methodology

Design Choice 1: PUF

- Use it as a key to encrypt the model **(Binding)**
  - only one specific device can unlock it during inference
- No training -> Scalable
- Stealing the encrypted weights do **not** give a usable model (shown later)

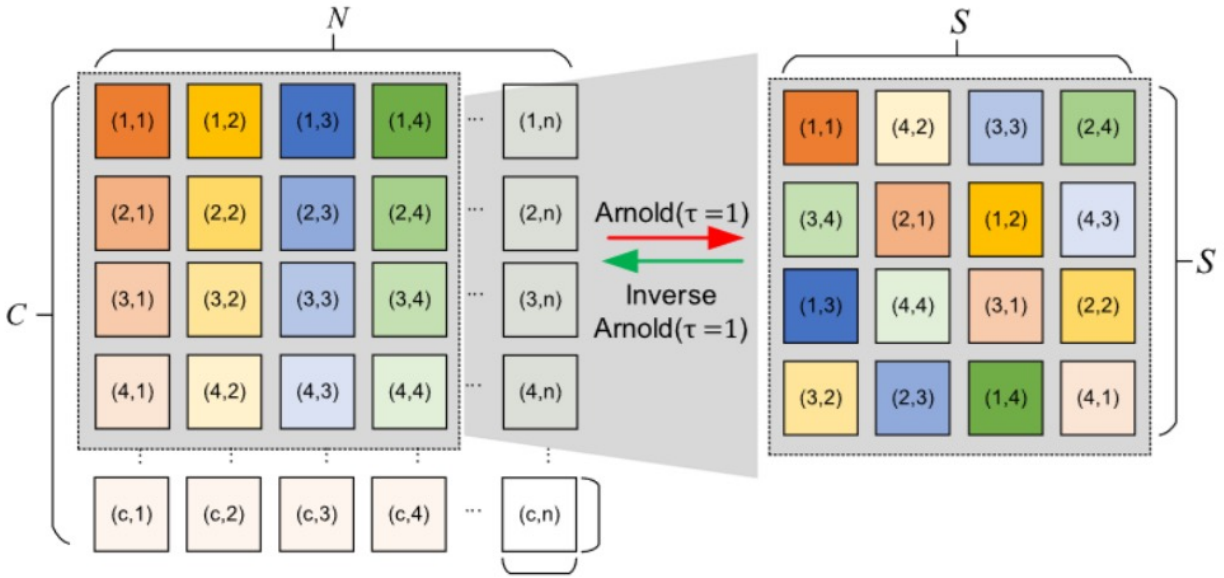


Methodology

Figure: Lin, Ning et al. (2020). Chaotic weights: A novel approach to protect intellectual property of deep neural networks. *IEEE TCAD.*

Design Choice 2: ACM Weight Permutation

- **What it is:** A chaotic map -> avalanche effect
- **Highly parallelizable** -> Fast on both CPUs and GPUs
- **Effective** -> Degrading performance



Encryption Key: 2×2 Matrix A

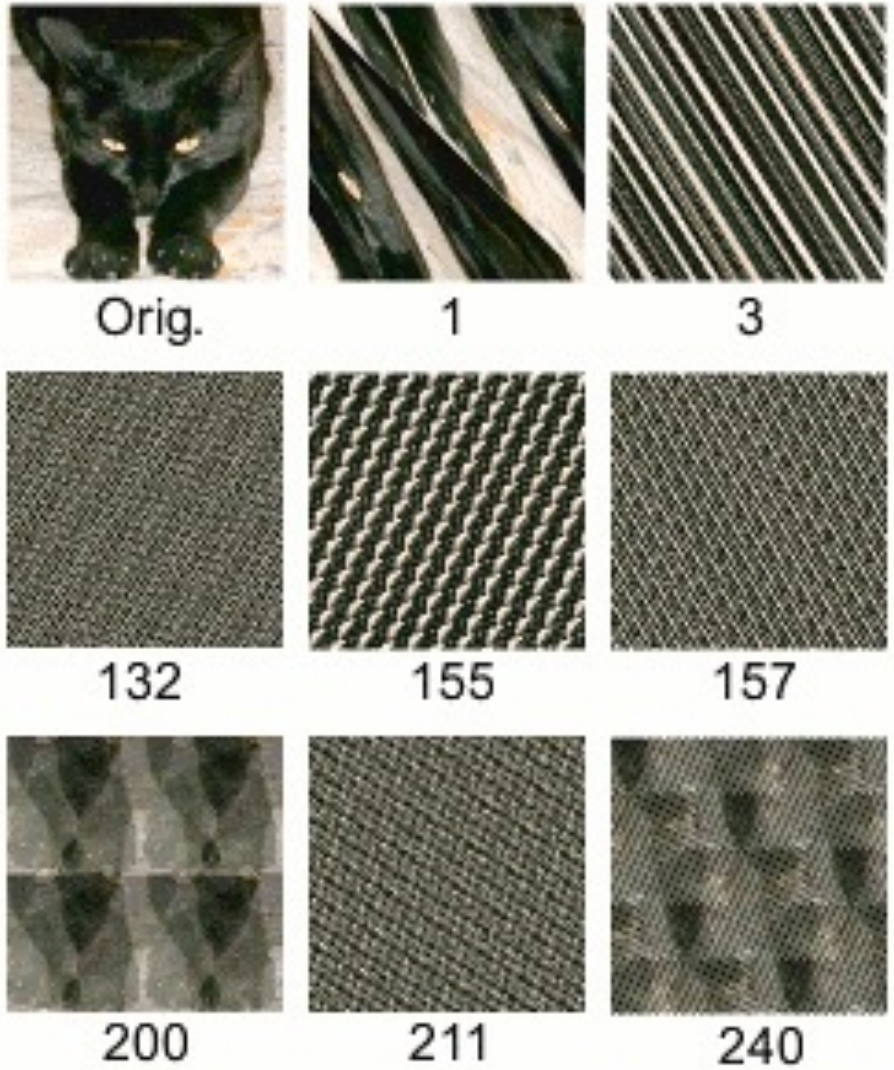
$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = A^n \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \pmod{S} \qquad (1)$$

Decryption:

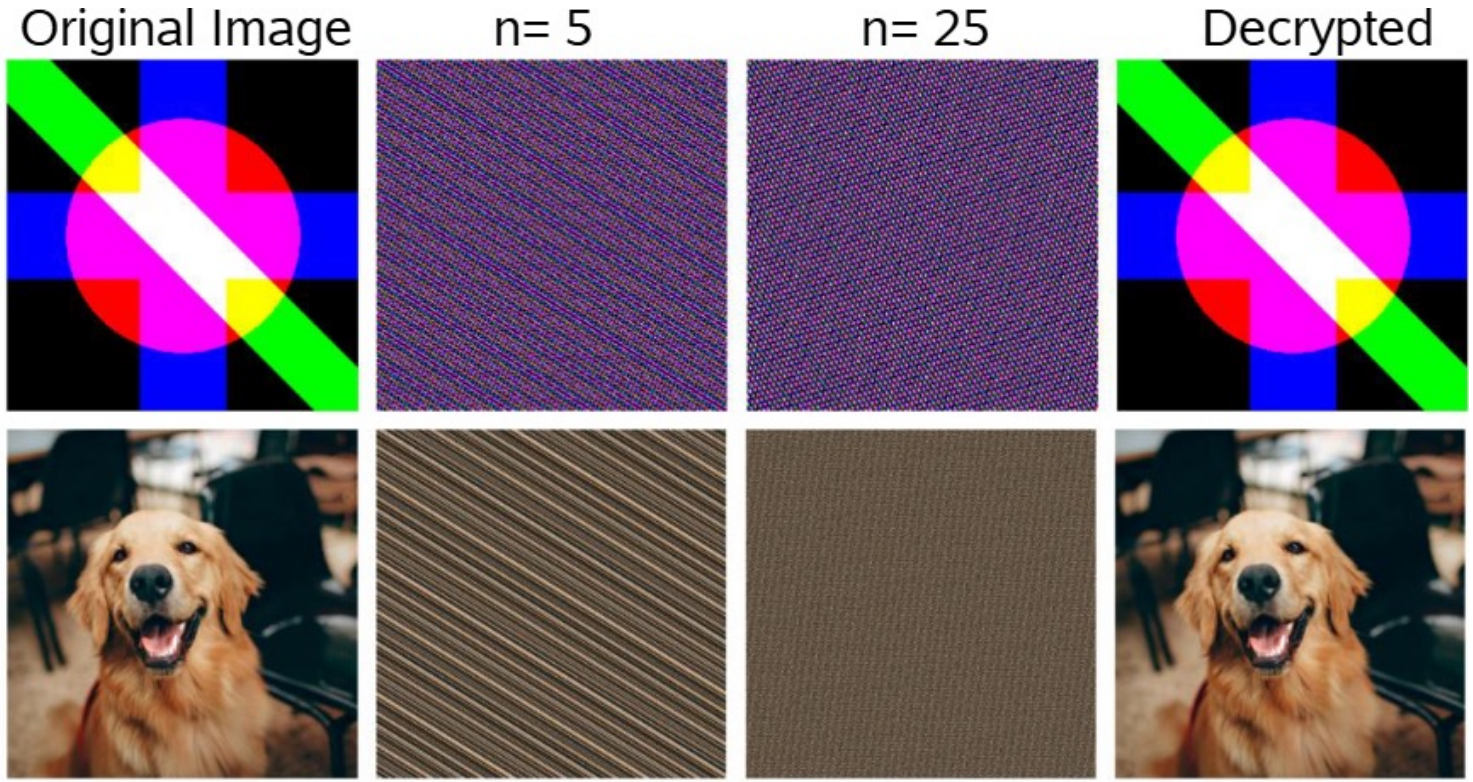
$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = A^{-n} \begin{bmatrix} x_n \\ y_n \end{bmatrix} \pmod{S} \qquad (2)$$

Methodology

Design Choice 2: ACM Weight Permutation



Example: Image Encryption



Methodology

Design Choice 2: ACM Weight Permutation

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.

Faster on both GPU and CPU

Including AES with AES-NI (hardware-accelerated) support

Methodology

Design Choice 3: Per-layer decryption/encryption

- **To reduce the attack window**
- Only one decrypted layer will exist in the memory at a time

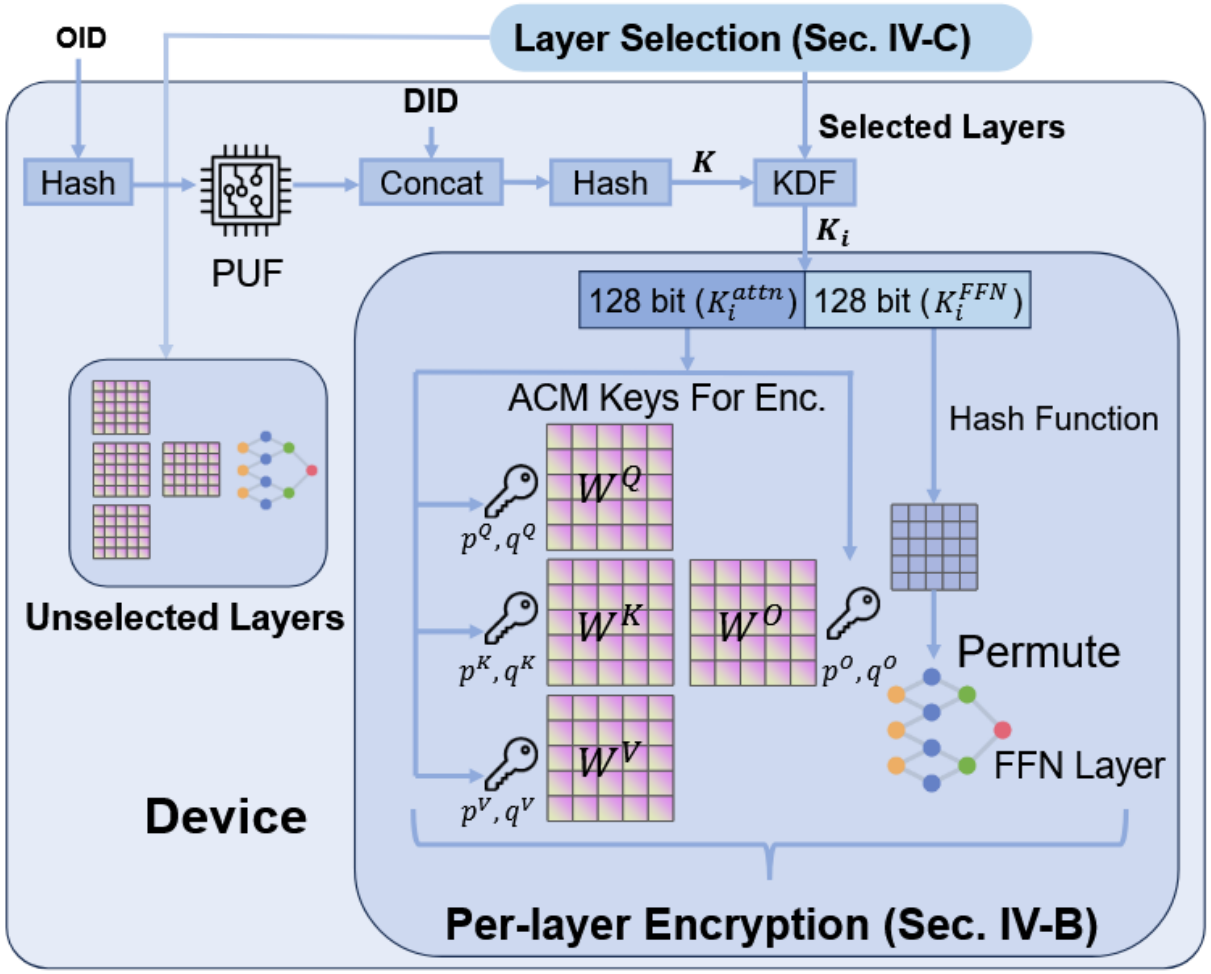
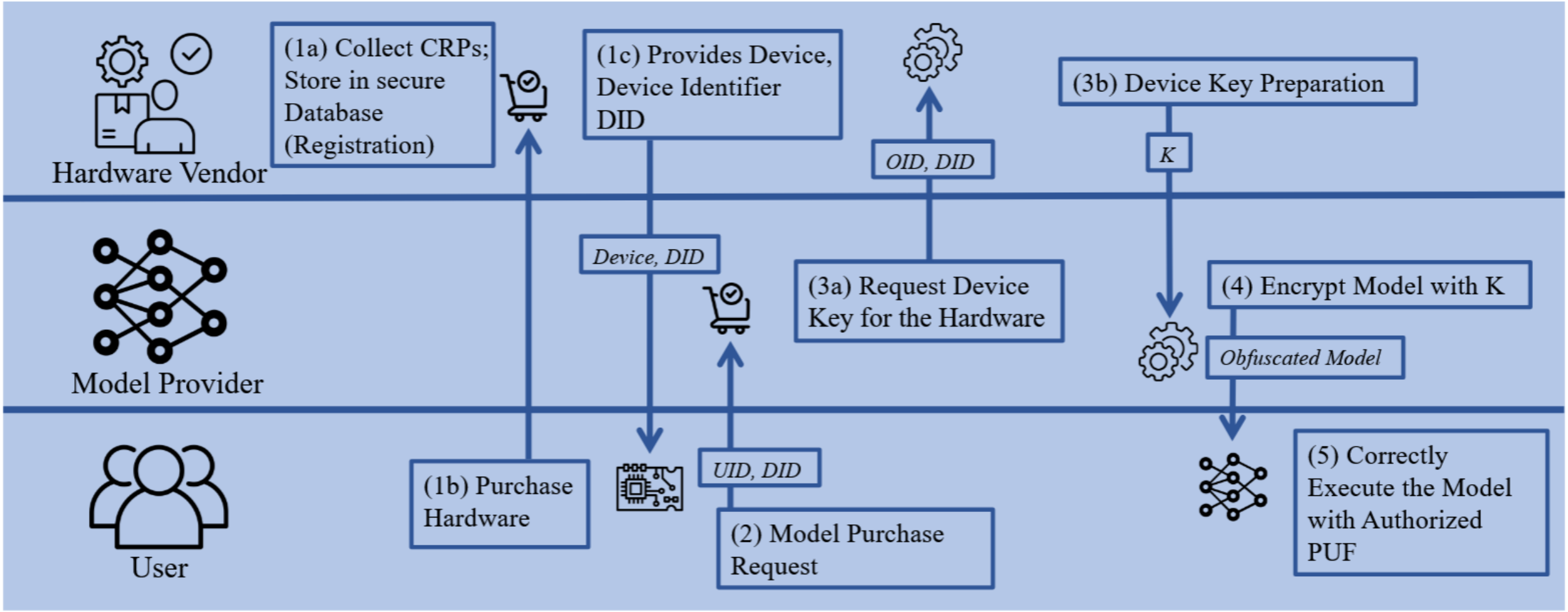


Figure 4. Our obfuscation scheme detailed in Sec. 4.

Methodology

Overall Flow (Proposed protocol)



Evaluation

Experiments

1. **Effectiveness:** one layer of encryption is enough to degrade the model into random guessing
2. **Fully Reversible:** important as an encryption

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.

Evaluation

Experiments

1. **Effectiveness:** one layer of encryption is enough to degrade the model into random guessing
2. **Fully Reversible:** important as an encryption

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.

Evaluation

Experiments

1. **Effectiveness:** one layer of encryption is enough to degrade the model into random guessing
2. **Fully Reversible:** important as an encryption

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.

Evaluation

1. Re-training attacks: we assume the attackers could possess up to 20% of the original dataset

- **More data** could lead to higher attack accuracy
- **>15% performance gap (71% vs. 86%)** compared with the original model

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.

Evaluation

2. Transfer Learning Attacks

- **Up to 20%** ImageNet + **full** downstream datasets
- **Still non-trivial performance gap (81% vs. 91% for CIFAR-100)**

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.

Evaluation

3. No hint for correct decryption

- When more than one layer is encrypted, **decrypting one layer does not bring noticeable accuracy boost**

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 0^{th} layer of the ViT-B model.

Evaluation

4. Even knowing which layers are encrypted does not lead to better attacks

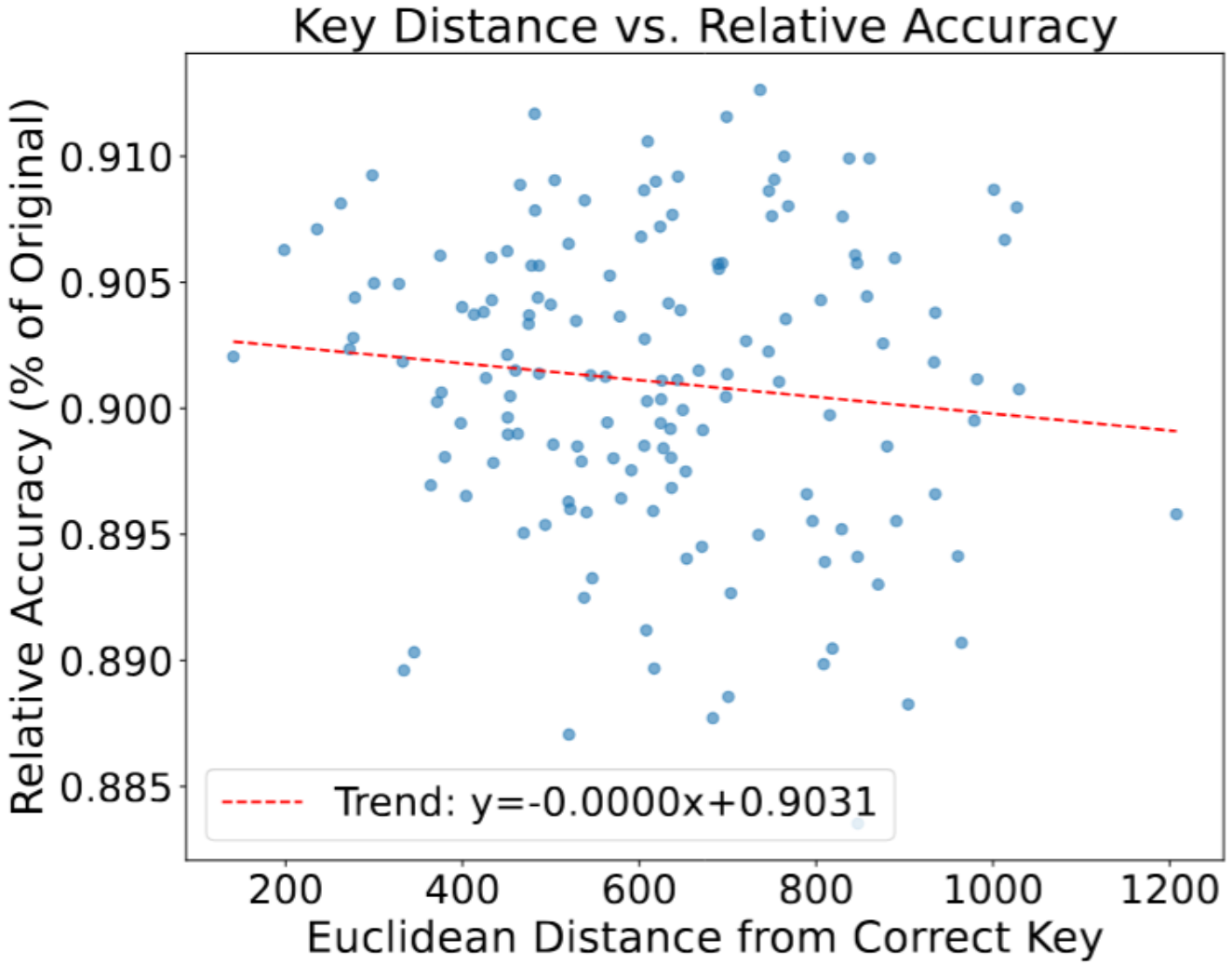
- Training **only on encrypted weights** do **not** give higher accuracy

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

Evaluation

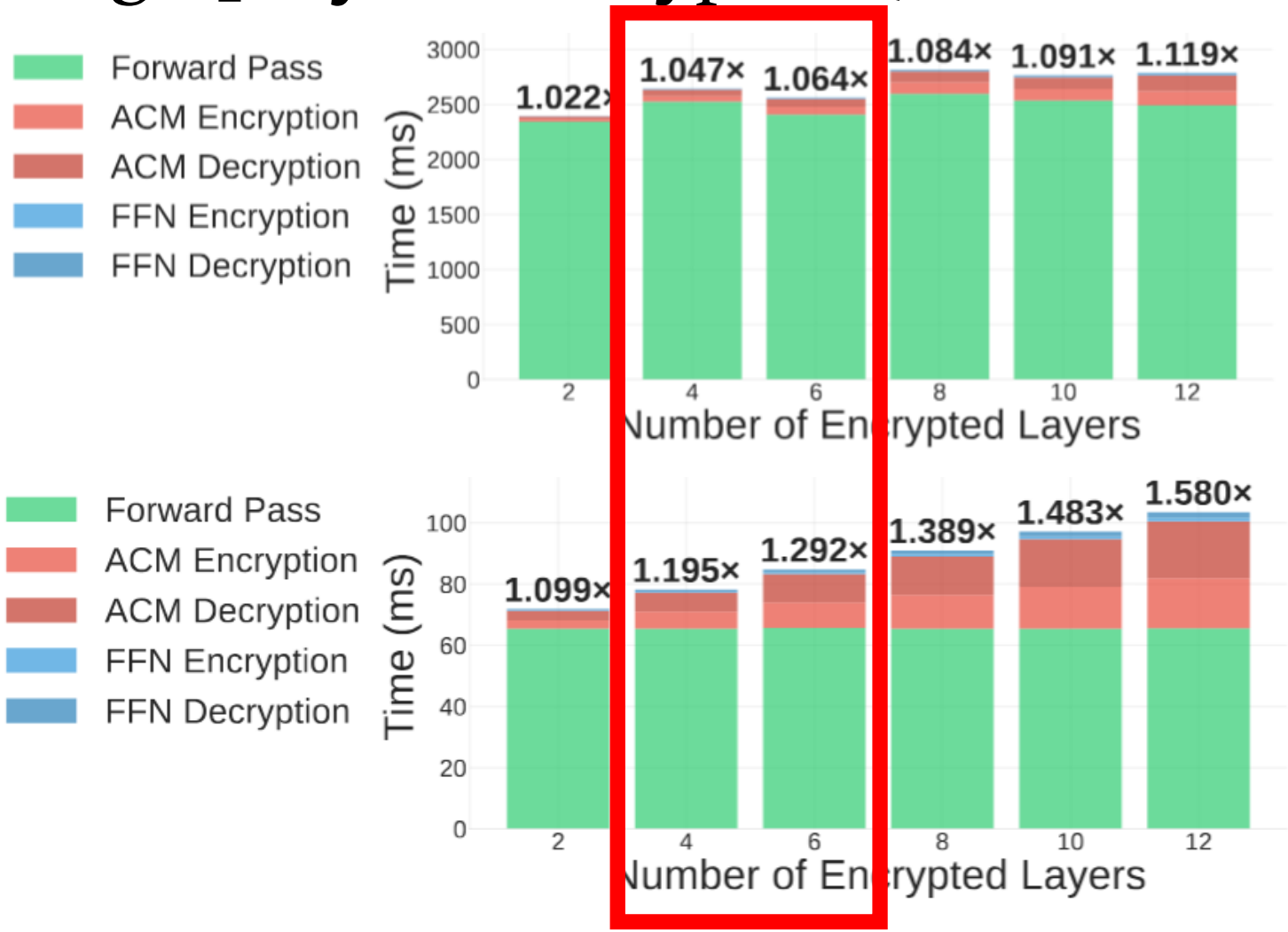
5. **Key guessing attack is hard**, as the accuracy drop is **not correlated** with the Euclidean distance from the correct key (avalanche effect)



Evaluation

6. Moderate or low overhead on both CPU and GPU

Default config: **4** layers encrypted (robustness evaluation)



Evaluation

7. Encrypting more layers enhance robustness

- After a certain number, the robust encrypting more layers saturates.

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

Summary

- **Contribution compared with previous work:**
  - **Active Protection** vs. passive protection (Watermark, Fingerprint)
  - **Training-free** vs. retraining required (Watermark, Active Access Control)
  - **Less hardware dependency** vs. high-end hardware support (CPU&GPU TEE)
  - **Focus on Large Transformers** vs. small traditional neural networks
  - **Secure Weights Enc.** vs. linear cipher for efficient encryption (TransLinkGuard)

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†)	×
Ours	**Active**	Weight Encryption	**No**	**Transformers**	Low ($\approx 0.2x^{\ddagger}$)	×

Summary

- **Active IP Protection:** hardware-bound framework that **proactively** prevents unauthorized execution of **on-device Transformer** models.
- **Security Primitives:** Our solution combines Physical Unclonable Functions (PUF) for device binding with reversible Arnold's Cat Map (ACM) for weight obfuscation.
- **Robust Defense:** The method renders stolen models useless (random accuracy) and resists retraining, key-guessing, and leakage attacks.
- **Efficient Deployment:** This approach requires no model retraining and incurs minimal runtime overhead, making it practical for edge devices.

Limitation and Future Work

1. **Speed**
   - Possible speedup: NVIDIA TensorRT
2. **Other Applications:** CNNs, LLMs, Diffusion Models…
3. **Protect Model Architecture:** Compatibility with model architectural obfuscation (Li et al., 2021)
4. **Granularity:** protect in a more fine-grained way (e.g. distinguishing Attention and FFN weights)
   - Less weights encrypted, less overhead

Li, Jingtao, et al. "Neurobfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing." *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021.

Thank You Very Much for Listening!

For any additional questions:

Peichun Hua
The Chinese University of Hong Kong, Shenzhen
(peichunhua@link.cuhk.edu.cn)

Securing On-device Transformer with Hardware Binding and Reversible Obfuscation

Peichun Hua[1], Hanxiu Zhang[1], Tuo Li[2], and Yue Zheng[1]

1 The Chinese University of Hong Kong, Shenzhen

2 Shandong Yunhai Guochuang Cloud Computing Equipment Industry Innovation Co., Ltd

Annual Computer Security Applications Conference (ACSAC 2025)

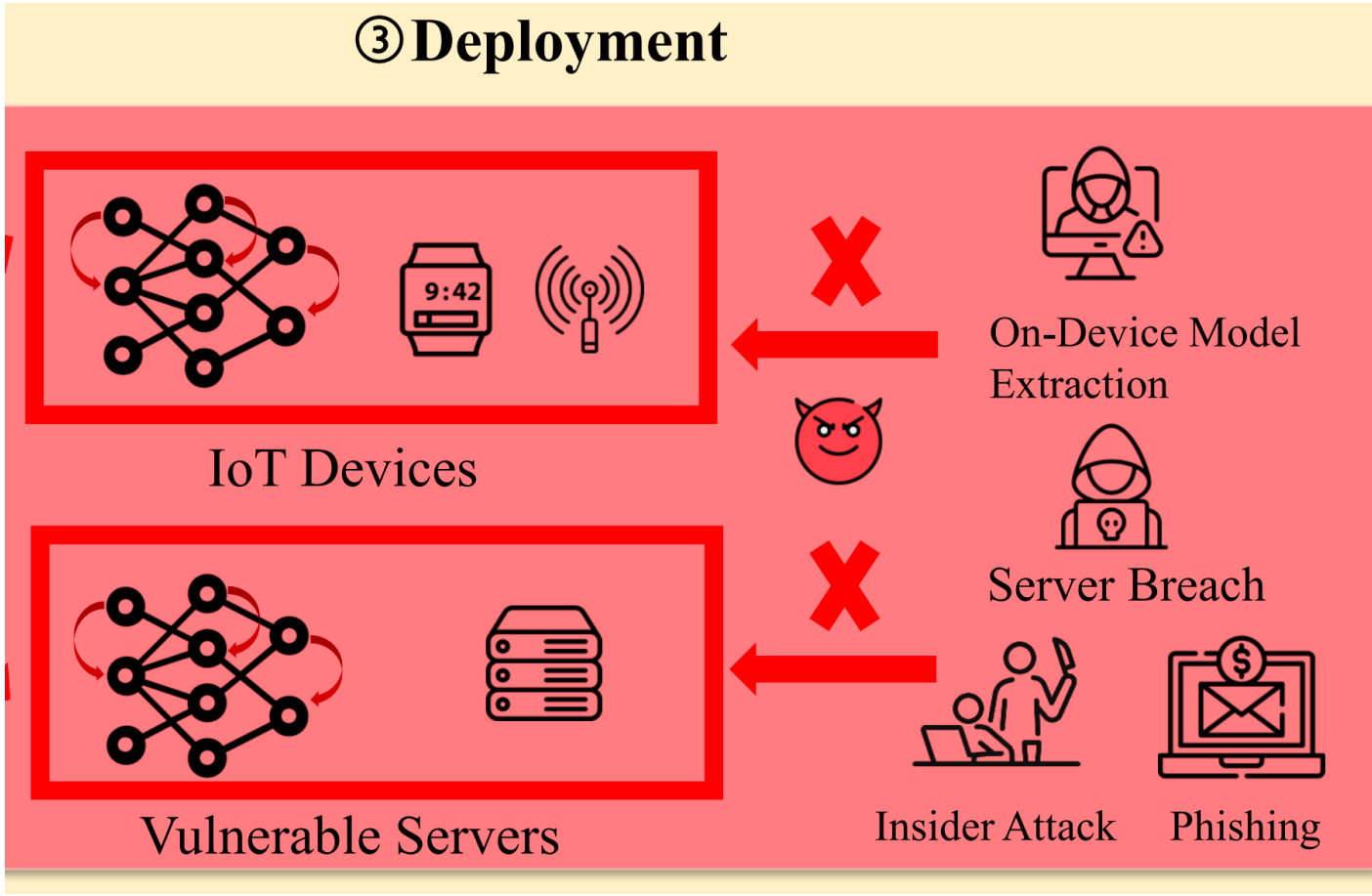
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

ACSAC
WWW.ACSAC.ORG

ADVANCING CYBERSECURITY SINCE 1985
ACSA
APPLIED COMPUTER SECURITY ASSOCIATES

Practical Scenario

1. ⭐ **Reverse engineering of edge software:** from an app/firmware image
2. **Insider Attack**: Model weights cannot be copied and transferred
3. **Supply-chain and distribution compromise:** update server, CDN, or customer distribution channel is breached
4. **Phishing**: Model Exfiltration



Details on Encryption

- **ACM typically works on squares**
  - Most transformers have square attention weights
  - For FFN weights we use linear transformation for better efficiency

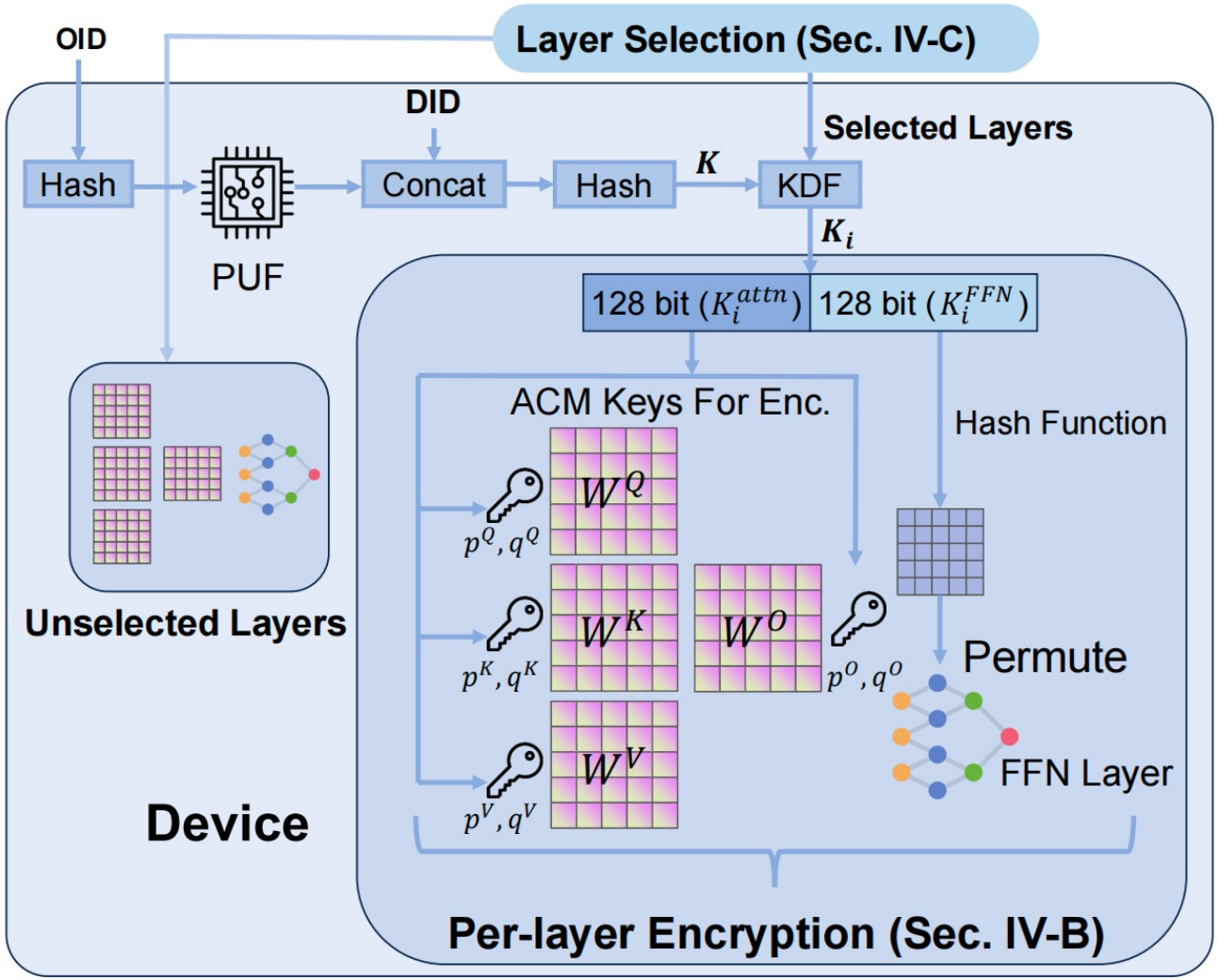


Figure 4. Our obfuscation scheme detailed in Sec. 4.

Details on Encryption

- **Known problem (after we finish this work):** linear transformation preserves the *relative* directions of the weight vectors
- Fine-tuning in transformers typically does not alter this fact much (Low-rank adaptation)
- Easy to guess and reverse-engineer the FFN weights -> impair security

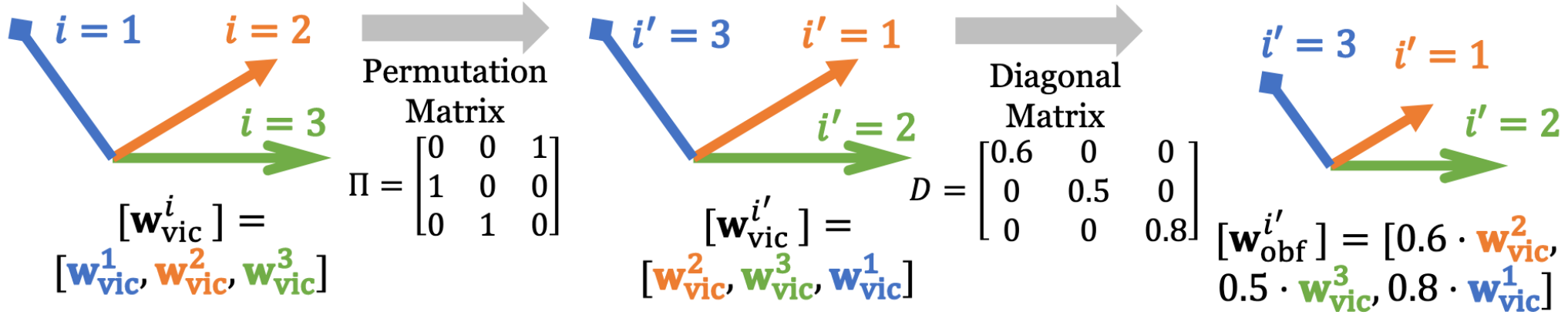


Figure: Wang, Pengli, et al. "Game of Arrows: On the ({In-) Security} of Weight Obfuscation for {On-Device}{TEE-Shielded}{LLM} Partition Algorithms." *USENIX Security 2025*

Details on Encryption

- **Alternative Strategy:**
- **GroupCover (ICML 2024):** Applies **matrix–vector** (or small matrix–matrix) multiplication within each group of weight vectors (slightly slower)
- **ArrowCloak (USENIX Security 2024):** Intentionally perturb the direction
- **O2Splitter (ACNS 2025):** Add Gaussian Noise with Differential Privacy (DP) guarantee (seeded with a secret key).

GroupCover [82]	ICML'24	$g(W_{\text{vic}}) = [A_1 W^{(1)}, \cdots, A_k W^{(k)}] \cdot \Pi,$ $g^{-1}(Y_{\text{obf}}) = ([A_1^{-1} Y_{\text{obf}}^{(1)}, \cdots, A_k^{-1} Y_{\text{obf}}^{(k)}]) \cdot \Pi^{-1}$
ARROWCLOAK	-	$g(W_{\text{vic}}) = (W_{\text{vic}} \cdot D_1 + \mathbf{v} \cdot \mathbf{1}_n \cdot D_2) \cdot \Pi$ $g^{-1}(Y_{\text{obf}}) = Y_{\text{obf}} \cdot \Pi^{-1} \cdot D_1^{-1} - X \cdot \mathbf{v} \cdot \mathbf{1}_n \cdot D_2 \cdot D_1^{-1}.$

Zhang, Zheng, et al. "Groupcover: a secure, efficient and scalable inference framework for on-device model protection based on tees." *ICML 2024*
Sun, Yulian, et al. "Obfuscation for Deep Neural Networks Against Model Extraction: Attack Taxonomy and Defense Optimization." *ACNS 2025*

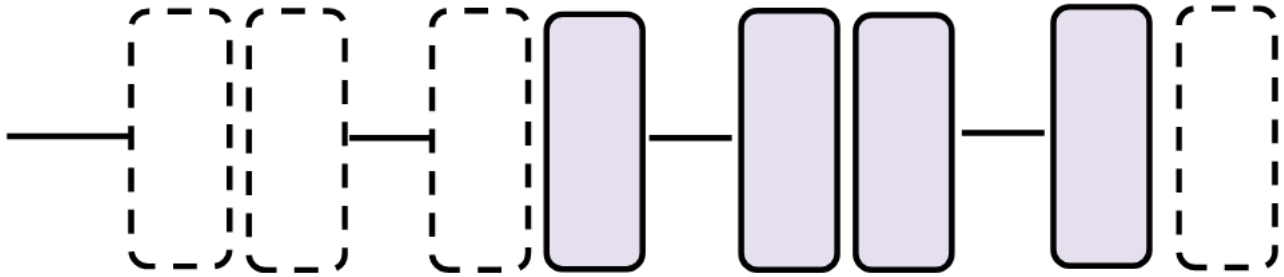
Details on Encryption

- **Which layers to encrypt?** 🤔
- Short answer: does not matter for robustness when the number >= 2
- Previous search: protect earlier layers (concept formation and feature extraction) first
- In our scheme: a simple greedy search for higher accuracy degradation or even random selection would bring similar good results.
- **Left as future work:** less weights (better efficiency) and good robustness
- E.g. An extreme: Corelocker locks 5% weights at neuron-level
- Caveats: Broken later by identifying these weights and fine-tune only on them

Wang, Zihan, et al. “Corelocker: Neuron-level usage control.” *IEEE Security and Privacy (SP)* 2024
Sun, Yulian, et al. "Obfuscation for Deep Neural Networks Against Model Extraction: Attack Taxonomy and Defense Optimization." *ACNS 2025*

Theoretical Outlook

- **Okay. The results look fine. But why does it work by encrypting a subset of the weights?**
- Findings: though when encrypting >=2 layers, which 2 (or more) does not matter very much, when encrypting the first layer, earlier layers tend to be more effective in locking the accuracy
- **Why?** Transformers have residual connections
  - Some layers hardly change the representation of the input in later layers
  - Enables layer pruning in transformers!



Prune a whole layer without affecting outputs

Qiao, Kangyu, et al. "IG-Pruning: Input-Guided Block Pruning for Large Language Models." EMNLP 2025.

Theoretical Outlook

- **Okay. The results look fine. But why does it work by encrypting a subset of the weights?**
- Findings: though when encrypting >=2 layers, which 2 (or more) does not matter very much, when encrypting the first layer, earlier layers tend to be more effective in locking the accuracy
- **Why?** Transformers have residual connections
  - Some layers hardly change the representation of the input in later layers
  - Enables layer pruning in transformers!
- **Hand-wavy**: when multiple layers are encrypted with independent random keys, their **perturbations propagate and compound** toward a predictable “noise distribution”
  - >=2 layers would be too much that the representation shifts off the region later layers recognize, and they effectively see OOD inputs.