International Journal of Machine Learning and Cybernetics
https://doi.org/10.1007/s13042-025-02552-w

ORIGINAL ARTICLE

Tripm: a multi-label deep learning SCA model for multi-byte attacks

Lianrui Deng[1,2] · Lang Li[1,2] · Yu Ou[1,2] · Jiahao Xiang[1,2] · Shengcheng Xia[1,2]

Received: 9 July 2024 / Accepted: 16 January 2025

Abstract
Deep learning methods have significantly impact in the side-channel attack (SCA) community. However, the training and verification phases of deep learning-based side-channel attacks (DL-SCA) typically focus on a single byte, which leads to the requirement of training numerous models to recover all partial key bytes. To resolve the problem, this paper proposes the TripM model, triple-keys attack model, which can attack three bytes in a single training session. First, TripM leverages label groups black to learn multiple bytes of leaked information in a single training session, where the label groups refers to divide labels to different groups according to the different attack bytes. The labels of TripM comprise three label groups, each group containing the point-of-interest information of the corresponding key. Second, the architectural design of TripM features two identical convolutional branches, allowing for the application of weight-sharing techniques. Both branches utilize the same weights, reducing the size of the model parameters and accelerating the training process. Finally, the TripM model employs a multithreading technique in the key recovery phase, where three threads concurrently compute the 3-byte Guessing Entropy (GE) value. Experimental results demonstrate that TripM can efficiently process the public ASCAD and TinyPower datasets, with an average of 80 and 89 traces required to recover a key. Average Layer-wise Correlation (AVE-LWC) visualization techniques also illustrate that TripM possesses excellent feature extraction capabilities.

Keywords Deep learning · Side-channel attack · Multilabel · Machine learning · Information security

1 Introduction

The leakage of side-channel information from cryptographic devices can make a significant vulnerability during the encryption process performed. This vulnerability could potentially be exploited by attackers to obtain critical key details, thereby leading to unintentional exposure of confidential data. Information leakage can manifest in various forms, including power consumption [1], electromagnetic emissions [2], and timing [3], thereby posing a substantial challenge to the information security framework of these devices. The power consumption is a prominent factor among these for an attacker.

Power attacks can be illustratively categorized into for distinct types: simple power analysis (SPA), differential power analysis (DPA) [4], correlation power analysis (CPA), and template attacks (TA) [5]. SPA, DPA and CPA are typically characterized as nonprofiling attacks, while TA is classified as a profiling attack. The conventional TA method involves two sequential stages: the profiling stage and the attack stage. The conventional TA differentiation quandary can be effectively reframed as a classification problem by deep learning methodologies. Consequently, the integration of deep learning with the SCA field becomes feasible.

The integration of deep learning into the SCA field led to several advancements. For instance, Hettwer [6] mentioned that Support Vector Machine (SVM) techniques successfully retrieve comprehensive key information for the first

✉ Lang Li
lilang911@126.com

Lianrui Deng
dlr8661@outlook.com

Yu Ou
blink_zip@126.com

Jiahao Xiang
jiahaoxiang2000@gmail.com

Shengcheng Xia
scxia_6@163.com

[1] College of Computer Science and Technology, Hengyang Normal University, Hengyang 421002, China

[2] Hunan Provincial Key Laboratory of Intelligent Information Processing and Application, Hengyang Normal University, Hengyang 421002, China

Published online: 31 January 2025

time. However, this approach required the training of a distinct model for each individual bit. The goal of the attacker is usually to recover the full round keys of the encryption algorithm. For example, the attacker needs to train 256 times to recover all keys in AES algorithm, which is complex and time-consuming. To improve attack efficiency, researchers use the Hamming weight or Hamming distance of intermediate values during the encryption process as trace labels [7, 8]. These labels contain eight bits, representing one byte of key information. As a result, training the model once can recover one byte of key information, enhancing the attack efficiency.

While eight bits (1 byte) can be recovered in a single training session, full-round attacks may still consume significant training time. To reduce training time on a training session, Ref. [9] proposed a DL-based SCA model called Parallel. Training were performed faster with the proposed algorithm than with the previous algorithm in their approaches. Reference [10] proposed a method for the optimal selection of support vector machine parameters that resulted in a 40% reduction in training time. Excellent attack performance and training speed were achieved by Li [11], who reduced training time by transforming the key recovery problem into a similarity measurement problem in Siamese neural networks.

However, there is a significant limitation of the existing research. The prevailing focus of existing research is predominantly on the recovery of single bytes. Advanced work such as [3, 8, 12] contributes to the domain of SCA, although it does not address the recovery of multiple bytes. The recovery of all 16 bytes of the AES key still requires training the model 16 times. Too many training iterations massively increase the time required for an attacker to recover the full round of keys. This approach is not only time-consuming but also computationally expensive, as each byte requires a separate training session. Consequently, the overall efficiency of the attack is significantly reduced, making it impractical for real-world applications where time and computational resources are critical constraints.

To address this limitation, we introduce the TripM model, designed for efficient recovery of three bytes. The TripM model incorporates several complementary strategies to enhance attack efficiency, including weight-sharing, label grouping, and multithreading. The model's feature extraction capability is analyzed using Average Layer-wise Correlation (AVE-LWC) techniques, and the feasibility of label groups is formally illustrated in SCA communication. This study makes the following contributions:

- Weight-sharing technique is employed by the TripM model, which is architecturally designed with two identical convolutional branches. The dual-branch design with shared weights amplifies the features extracted from a single convolutional branch, enabling the model to capture double the leakage information from traces.
- The concept of label groups is introduced and applied to the TripM model. Mathematical methods are employed to provide a formal justification for the choice of label groups, demonstrating the feasibility of label groups in the SCA domain. This technique allows the model to learn features of multiple bytes in a single training session by dividing labels into different groups according to the attack bytes.
- A multithreaded key parallel recovery strategy is proposed to facilitate multibyte attacks. This strategy improves attack efficacy by allowing three threads to concurrently compute the 3-byte Guessing Entropy (GE) value, further speeding up the key recovery process.

The remainder of this paper is organized as follows. Section 2 is the related work section, Sect. 3 is the background section, in which we introduce convolutional neural networks, and present the attack process for a single byte. Section 4 focuses on the proposed TripM model and label groups. Section 5 presents the experiments about the tripM model. Section 6 presents the discussions. Finally, Sect. 7 concludes the study.

2 Related Work

The application of deep learning techniques, particularly convolutional neural networks (CNNs), has significantly advanced the side-channel attacks (SCA) domain. CNNs are considered a better choice for the SCA field in existing research. Reference [13] demonstrated the powerful performance of several types of neural networks, most notably CNNs. Reference [14] recommended CNNs as the algorithm of choice for performing deep learning-based attacks. Reference [15, 16] mentioned that CNN-based models perform better than transformer-based models on synchronized datasets. Reference [8] mentions that CNNs have the potential to deal with trace misalignment, so they focus on the family of CNNs.

Building on these foundational works, several researchers proposed various CNN-based models and strategies to enhance the effectiveness of profiled SCA. Di Li [11] et al. designed a Siamese neural network model with a CNN architecture, which can be successfully trained with 1000 power traces and has good attack capability and training speed. Thapar [17] et al. proposed TranSCA, a new DL-SCA strategy that combines the CNN architecture with multilabel classification and migration learning techniques, aiming to reduce the number of power traces required from the target device to obtain the key. Ni [18] et al. proposed a CNN attack scheme that utilizes CNN model fusion for

side-channel attacks. High-dimensional features in the middle layer of the base model were merged to increase the amount of effective information and enhance the generalization ability of the neural network model. A new fusion model was then constructed through training. Cagli and Dumas [19] et al. proposed an end-to-end attack strategy based on convolutional neural networks, greatly simplifying the attack roadmap by requiring neither prior realignment of traces nor precise selection of points of interest. Zaid et al. [12] introduced weight visualization and heatmaps to analyze patterns similar to points of interest and proposed a method for generating suitable CNN architectures based on the degree of desynchronization.

CNNs are also utilized in the non-profiling attacks domain. Reference [20] creates a non-profiled SCA utilising convolutional neural networks (CNNs) on the AES-128 cryptographic algorithm. Experiments show that they method can recover more bytes successfully from SCA compared. Reference [21] finds that the MOR-CNN model executes attacks at a speed 40 times faster than the DDLA model on non-profiled SCA.

Multilabel classification is commonly used in the computer vision (CV) field, such as image annotation, image search, and image retrieval [22]. There are also some studies on the use of multilabel techniques in SCA context. Zhang [23] applied multilabel classification in SCA domain, where the binary form of intermediate values was transformed into labels of the dataset for key recovery. Fukuda [24] uses multilabel against hardware-implemented AES with rotating S-boxes masking (RSM) countermeasures.

3 Background

3.1 CNNs on SCA

CNNs are a subset of feedforward neural networks that incorporate convolutional computations into their deep structures. They are representative algorithms in the field of deep learning, comprising a convolutional layer and a pooling layer [25, 26]. The convolutional layer extracts features from the power traces in the SCA domain in SCA communication. The convolutional kernel systematically sweeps over the power traces to generate the feature map as part of its operation. Different convolution kernels extract distinct features from the power traces. If the input data is denoted as t, the convolution kernel as g, and the bias of the convolution kernel as b, the output of this convolution kernel can be expressed as follows:

$$y = f(t * g + bias) \tag{1}$$

The side channel traces, which are one-dimensional data inputs, are fed into the convolution block. Assuming that the trace inputs to the convolution layer are denoted as x, encompassing a total of n dimensions, and the size of the convolution kernel is represented as a, the feature z extracted by this convolution layer can be articulated as follows:

$$z[i] = \sum_{j=0}^{n} (x[i-j+1] * g[j]) \tag{2}$$

The operational mechanism of the convolution block is shown in Fig. 1. Initially, the power traces are fed into the convolution layer as one-dimensional vectors. The convolution kernel then performs a convolution operation on the input data, extracting the features of its window while

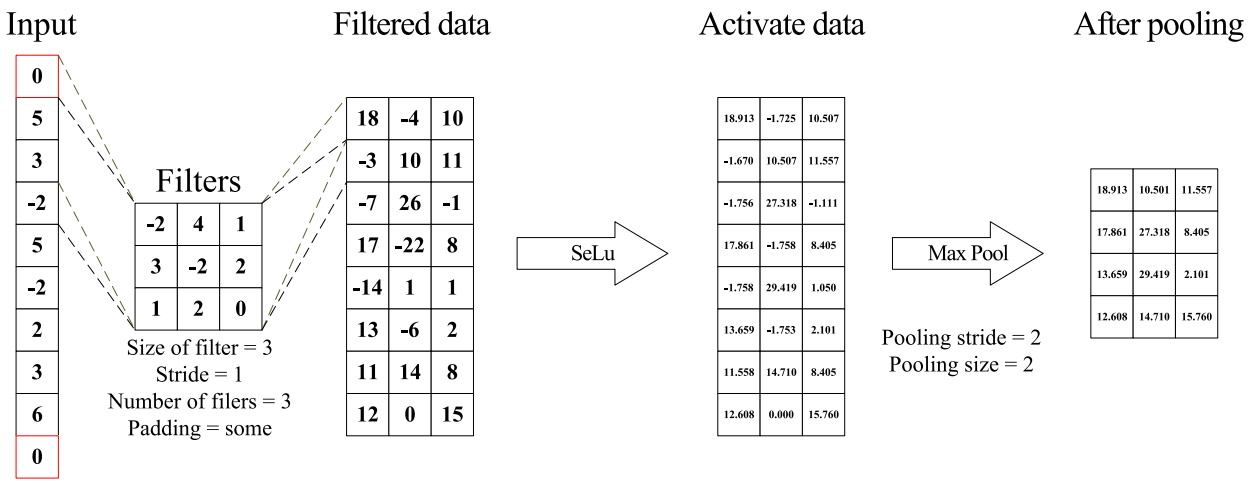


Fig. 1 Operations in convolutional blocks

simultaneously smoothing the shift in the input data. This process is repeated until all the input power data have been processed, resulting in the features extracted by this convolution kernel.

Nonlinearity is enhanced by appending an activation function after the convolutional layer, which activates the output data of the convolutional layer. A pooling layer, introduced following the convolutional layer, addresses the issue of complexity explosion owing to an excessive number of parameters. This layer samples the inputs in reduced dimensions using a pooling window.

The operational rules of the pooling layer allow to be categorized into a maximum pooling layer and an average pooling layer. The maximum pooling layer outputs the maximum value and the average pooling layer outputs the average of the values in the pooling window.

3.2 The phase of DL-based profling attacks

A comprehensive analysis of DL-SCA delineates two distinct phases: the initial profiling phase, often referred to as the learning phase, and the subsequent attack phase, known as the verification phase.

1) *The profiling phase* This phase requires the initial capture of power traces. Each trace corresponds to a specific label, and the ensuing training dataset can be formally represented by the following equation:

$$D(p) = \{(X_i, p_i, k_i) | 1 <= i <= N_p\} \tag{3}$$

Equation (3) uses X_i to represent the power traces, while p_i and k_i are associated with the input data. Here, p_i is the i-th plaintext, and k_i is the key corresponding to the i-th plaintext. N_p represents the number of power traces collected from all plaintext encryptions. A power trace corresponds to a one-hot vector, with the label representing an intermediate value. This is due to the inherent relationship that exists between intermediate values and their corresponding power traces. The label of the trace is typically the output value of the encryption algorithm's nonlinear structure. For example, the S-box can serve as label in AES and the label is denoted as follows:

$$Label = Sbox(p_i \oplus k_i) \tag{4}$$

The training dataset is formulated by pairing the one-hot vector of intermediate value with its associated power traces and then employed as an input to train a neural network, which ultimately yields 256 predicted values as its output in AES:

$$P = \{P_n | 0 <= n <= 255\} \tag{5}$$

In Eq. (5), n stands for the label of the output and P_n stands for the probability that the label (intermediate value) is n.

2) *The attack phase* The appropriate metric must be selected to evaluate the performance of an SCA. Researchers use accuracy and loss values to assess classification performance in many deep learning domains,. However, these metrics have been shown to be unsuitable for the SCA domain [27]. The Guessing Entropy (GE) is a more appropriate metric for evaluating the performance of an DL-SCA. GE is a metric that quantifies the model's uncertainty regarding the key. The GE value is inversely proportional to the model's knowledge of the key. The lower the GE value, the higher the model's knowledge of the key. The GE calculation process can be described as follows. The attack set is fed into the model for prediction during the attack phase. The model calculates the probability of each trace's corresponding label. Then, the labels corresponding to the guessed key are determined for each value ranging from 0 to 255. The probabilities of these labels are summed up. The real key's rank among these summed probabilities is calculated multiple times. This rank is also known as the GE. The formula can be expressed as follows:

$$GE = \sum_{N=1}^{N_{attack}} rank_i(k^*)/N_{attack} \tag{6}$$

Equation (6) uses N_{attack} to denote the number of attacks, i to represent the i-th traces used for computing the ranking, k^* as the correct key, and $rank_i(k^*)$ as the ranking of the correct key.

4 TripM model

4.1 Summary

TripM model is engineered with the capacity to recover multiple keys through a single training iteration. This is accomplished by altering the label selection strategy of the model. Rather than utilizing one-hot vectors for single-byte attacks, label groups are established, with each label group embedding one key piece of information. Structurally, TripM incorporates two identical convolutional branches that employ a weight-reuse technique. This method effectively diminishes the model's parameter count and expedites the training process. The key recovery phase requires a different approach, as the single-byte GE sequential computation process is not applicable. Instead, a multithread technique is applied to the multibyte attack, with three threads concurrently computing the GE of three bytes. This multithread attack approach transforms a conventional sequential key

attack into a concurrent recovery process, thereby enhancing the efficiency of the attack. The overall attack process of TripM is shown in Fig. 2 and can be summarized as follows:

1. Dataset transformation: The training set is redesign to 1200 dimensions, each of which contains 3 bytes of exposure power data. Concurrently, a trace's labels are divided into three label groups.
2. TripM model training: The redesigned dataset is utilized to train the TripM model.
3. Key recovery process initiation: Three distinct key recovery processes are instituted, each assigned to retrieve one byte of key information.
4. Concurrent key recovery: These processes simultaneously undertake the key recovery, thereby augmenting the operation's efficiency.

4.2 Structure of TripM model

The construction of deep learning architectures usually presents considerable challenges in DL-SCA domain, The development of optimally performing SCA models often rely upon methodologies for hyperparameter research or draw upon insights from existing literature. This study adopts a methodological approach informed by the accumulated expertise of researchers within the field, thereby facilitating the design of the TripM models.

- *Hyperparameter selection for convolutional layers* Zaid [12] noted that when points of interest are in close proximity, an excessive number of convolutional layers may unintentional extract noise from the traces, thereby disrupting the feature extraction process. Lennert Wouters et al. [28] further affirmed in their research that the network's performance can be enhanced by enlarging the filter size and increasing the number of convolutions. Consequently, these provide a basic principle for the design of the neural network structure designated the filter size as 50 and the number of convolutional kernels as 64.
- *Innovation in weight sharing and dual-branch* One of the key innovations of the TripM model is its effective use of weight-sharing between two identical convolutional branches. Ni [18] has previously highlighted the efficacy of the CNN model fusion method in enhancing the attack capability of SCA. The design of tripM, which incorporates two convolutional branch as inputs to the dense layer, is inspired by the work of Lei Ni. TripM advances this by employing weight-sharing between the two convolutional branches. This weight-sharing technique allows the model to capture similar features at the same points of interest in power traces, which often exhibit jitter. The dual-branch design with shared weights amplifies the features extracted from a single convolutional branch, enabling the model to capture double the leakage information from traces. Those techniques not only reduce the overall complexity of the model by cutting down on the number of parameters but also enhance its ability to learn more generalizable features by encouraging the two branches to focus on shared underlying patterns across multiple bytes of data.
- *Dimensionality reduction and robustness* It is important to note that the final extracted features from the convolutional layers remain high-dimensional, which can make further analysis difficult. To address this, a fully connected layer with a small number of neurons was added after the convolutional block to reduce the dimensionality and increase the robustness of TripM. Specifically, three fully connected layers, each containing 25 neurons, were introduced into the structure. This compact design ensures that the final feature representation is concise, aiding in both training efficiency and inference speed.

Table 1 lists the architecture of the tripM model. In the table, the notation *X*–*Y* in the convolutional layer signifies that this layer comprises *Y* convolutional kernels, each of size

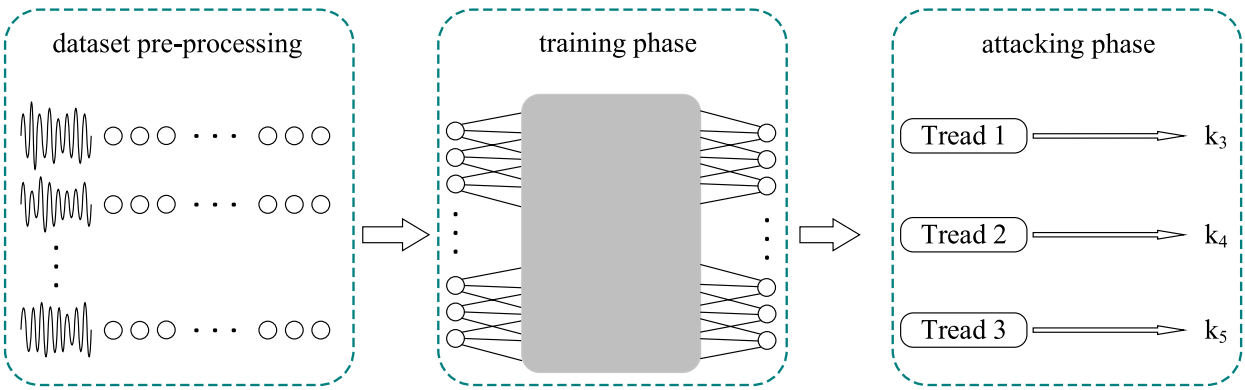


Fig. 2 The overall process of TripM

X. Similarly, the notation A–B in the pooling layer indicates that the pooling size is A, with a stride of B. For instance, the term "Conv1D 50-64 selu" refers to a one-dimensional convolutional layer equipped with 64 convolutional kernels, each of size 50, and employs the "selu" activation function. "AveragePooling1D 50-50" represents an average pooling layer with a pooling size and stride of 50. The abbreviation "BN" is used to denote batch normalization. Figure 3 illustrates the architecture of the TripM model.

4.3 Label groups

Traditional side-channel attack methodologies predominantly address multiclass classification problems. This approach is extended by introducing a multilabel classification problem by using label groups, allowing the model to learn leakage information for multiple keys concurrently within a single training session.

An attacker must capture power traces generated during the device's encryption process in the context of SCA. Capturing this leakage information necessitates the recording of numerous intermediate operations throughout the encryption process, typically involving nonlinear operations of the encryption algorithm. The traces of the entire encryption process encompass the handling of all intermediate operations within a given round, each processed in relation to different subsets of encryption keys and associated with distinct intermediate operations.

Let's denote the plaintext and the set of keys as P and K, respectively. The set of power traces resulting from encryption is represented by $t(P, K)$. The leak information from the operation of the subset nonlinear structures is denoted by ϕ. The relationship is as follows:

$$\phi \subsetneq t(P, K) \tag{7}$$

Consider a key space $K = \{k[1], k[2], \ldots, k[n]\}$ and an intermediate value space $Z = \{0, 1, \ldots, m\}$. The size of the element $K[i]$ in the key space varies across different encryption algorithms and is related to the output size of the Non-Linear components of the encryption algorithm. The n and m are intricately linked to the encryption algorithm under consideration. It is discernible that a nonlinear operation, like Sbox operation,for ϕ, a lemma, grounded on the fundamentals of SCA, is established:

Table 1 Model hyperparameters

Layer (left branch)	Layer (right branch)
Input 1200	Input 1200
Conv1D 30-64 selu	Conv1D 30-64 selu
BN	BN
AveragePooling1D 50-50	AveragePooling1D 50-50
Conv1D 3-32 selu	Conv1D 3-32 selu
BN	BN
AveragePooling1D 2-2	AveragePooling1D 2-2
Concatenate	
Dense 25 selu	
Dense 25 selu	
Dense 25 selu	
Dense 768 sigmoid	

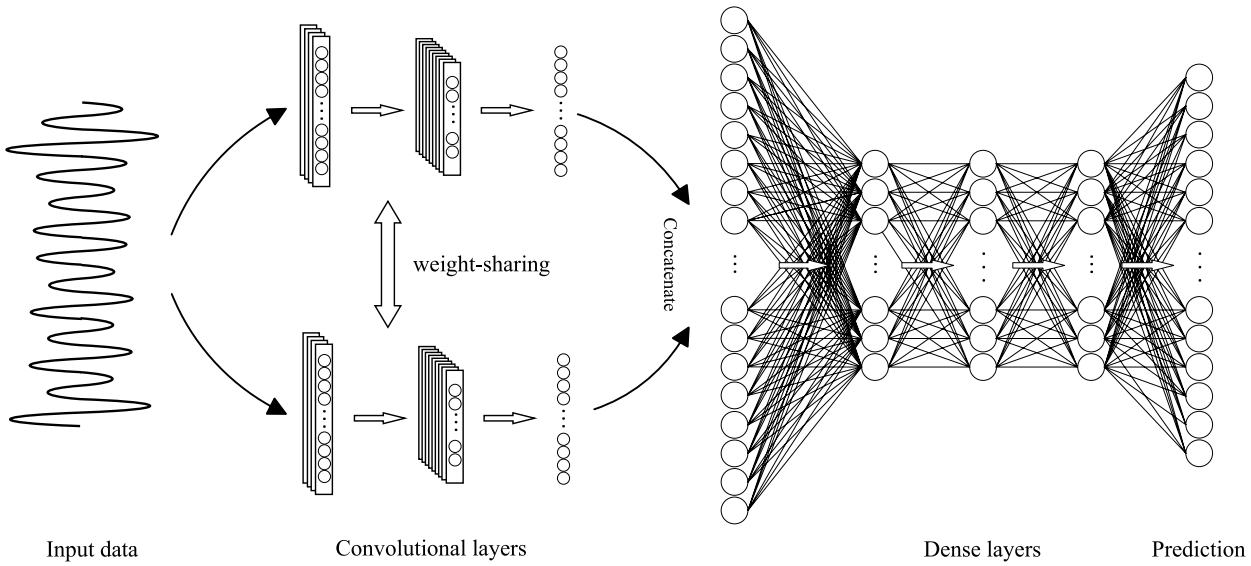


Fig. 3 TripM model architecture

Lemma 1 $\exists\ \beta : \Phi \rightarrow K.\ \forall\ \Phi[x],\Phi[y] \in \Phi$ *and* $x \neq y$, *then* $\beta(\Phi[x]) \neq \beta(\Phi[y])$.

Proof Equation 7 shows that ϕ is a subset of $t(P, K)$, with P representing the plaintext and K the key. An intermediate value must correspond to function f for any ϕ in t during a single encryption. This intermediate value is closely related to participation in this nonlinear operation through the function g. Simultaneously, the operations of each nonlinear component are independent of each other, implying that for any ϕ in t, a corresponding encryption key can be found. β can be represented as follows:

$$\beta = f(g(x)) \tag{8}$$

□

This indicates that for each element $\phi[i]$ in ϕ, a unique corresponding element exists in K. Furthermore, β establishes a one-to-one correspondence from ϕ to K. The objective of deep learning is to ascertain the mapping β.

Considering the preceding discussion, the lemma is proposed by the following:

Lemma 2 $\exists\ \beta^{-1}, \forall\ K[x], K[y] \in K$ *where* $x \neq y$, *it holds that* $\beta^{-1}(K[x]) \neq \beta^{-1}(K[y])$

Proof Multiple elements in the leakage space correspond to an element in the key space, as hypothesized. However, because the cardinality of the key space equals that of the leakage space, as per foregoing, this would imply the existence of an element in the key space without a corresponding element in the leakage space. This is clearly not possible within the scope of side-channel analysis. Evidently, A bijective relation exists between ϕ and K, with every element in ϕ and K corresponding one-to-one. □

Lemma 2 establishes that the mapping from K to ϕ is also an injective relation. Each element in key space K corresponds uniquely to an element in leakage space ϕ throughout the encryption process. This injective relation is crucial for the successful application of multi-label classification in the SCA domain.

Building upon the foundations laid by Lemma 1 and Lemma 2, Theorem 1 is proposed to validate the feasibility of label groups in the side-channel context.

Theorem 1 *During the encryption process, the operations on the different key subsets are independent. Similarly, when the labels of side-channel traces are divided into multiple groups, these groups can also be considered independent of each other.*

Proof Different key subset operations during the encryption process are independent, as established in Lemmas 1 and 2. Similarly, when the labels of side-channel traces are partitioned into multiple groups, these groups can be regarded as independent. This forms the foundation for the application of multilabel classification in the SCA domain. Theorem 1 is thus proven. □

Theorem 1 establishes the feasibility of using multiple independent label groups within the SCA domain. Multiple label groups can be designated as model labels. The size of a label group is related to the output size of the nonlinear component of the encryption algorithm. For each quantity of label groups, the following properties hold:

Property 1 *When the output size of the non-linear component corresponding to a subset of the key is N bits, the size of this group of keys subsets is* 2^N.

Proof A single label within the dataset comprises elements of 0 or 1. Given that the output of the nonlinear structure of the encryption method is N, the size of the label must be at least 2^N to retain comprehensive information regarding this intermediate value. If the size of the label group is less than 2^N, the labels will not be able to represent all the information of the intermediate value and the model will not be able to learn the information of the intermediate value. □

The AES encryption algorithm uses an S-box with both input and output sizes of eight bits, resulting in a label size of 2^8 for this group of key subsets. On the other hand, the DES encryption algorithm uses an S-box with an output size of four bits, resulting in a label size of 2^4 for this group of key subsets.

Property 1 outlines the methodology for determining the size of a label group in SCA domain. According to Property 1, if the intermediate value of a trace is denoted as n, the label selection strategy for this label group can be represented as follows:

$$Group[x] = \begin{cases} 0 & \text{if } x \neq n \\ 1 & \text{if } x = n \end{cases} \quad x \in [0, 2^N - 1] \tag{9}$$

The $n-$1-th label in this group is assigned a value of 1, while all remaining labels are assigned a value of 0, as defined in Equation (9).

Experiments were conducted using the ASCAD dataset [16]. This dataset is based on the AES algorithm; thus, each label group in this study has a size of 256 (2^8). Each trace in this study is associated with three distinct label groups. Specifically, the first label group contain the intermediate value information of the third byte, the middle label group contain the information of the fourth byte, and

the last label group contain the information of the fifth byte. Each label group encompasses the output data from the corresponding S-box. The labels for a given trace can be articulated as follows:

$$\begin{aligned} \text{label} = (&Group_{sbox(P_2,K_2)}, \\ &Group_{sbox(P_3,K_3)}, \\ &Group_{sbox(P_4,K_4)}) \end{aligned} \tag{10}$$

$Group_{sbox(P_2,K_2)}$, $Group_{sbox(P_3,K_3)}$, and $Group_{sbox(P_4,K_4)}$ in Equation (10) represent the S-box outputs of the third, fourth, and fifth bytes, respectively.

The label schematic depicted in Fig. 4. The figure elucidates the label selection strategy employed for TripM model. Labels are partitioned into three groups, each corresponding to the output of an S-box. Subsequently, these labels are transformed into a multi-label format, with each group representing a distinct subset of keys. This label selection strategy enable the model to recover multiple keys within a single training session.

4.4 The use of multi-threaded

Traditional models of power attacks often rely on single-byte attacks, which require sequential attacks on each byte during the attack phase. This approach requires the secret key to be attacked in sequence based on the leakage information of the trained bytes, which poses a significant challenge for the attacker. Our research addresses this issue by focusing on multibyte attacks. The tripM model is capable of learning the leakage information of three bytes in a single training session. Consequently, a multibyte attack method is proposed, which is a multithreaded attack strategy implemented in Python.

The effectiveness of key recovery procedures is enhanced by integrating the principles of concurrent programming with side-channel attacks. The threading module in Python facilitates the implementation of multi-threaded programming. The pseudocode representing the attack flow is presented in Algorithm 1.

Algorithm 1 Key recovery based on python multithreading

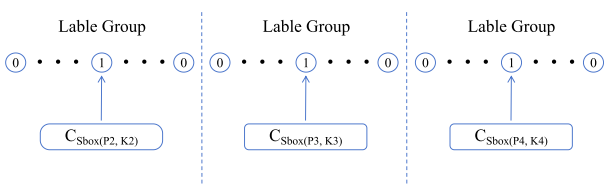


Fig. 4 Label Schematic

Input: $predictions$: Prediction results of the attack set. K_s: Quantity of keys retrieved in a single attack. $targets$: The middle value of the keys.$ntraces$,$nattack$: the parameters of attack phase.
Output: GE of correct keys

```
for i in [1, K_s] do
    Thread[i] ← attackThread(predictions, k[i], targets[i], ntraces, nattack) {Start a new attack thread}
    Thread[i].start() {Start the thread}
end for
for i in [1, K_s] do
    Thread[i].join() {Wait for the thread to finish}
end for
for i in [1, K_s] do
    Result ← Thread[i].getResult() {Get the result from the thread}
end for
return Result
```

Thread in Algorithm 1 represents a Python thread object. We initiate each *Thread* instance by invoking the *start*() method first, and then merge it with the main thread by calling the *join*() method.

5 Experimental results

5.1 Experimental setup

The performance of the tripM model was assessed using the software implementation of the ASCAD [16] and TinyPower dataset [29].

- *ASCAD dataset* The ASCAD dataset is a prevalent choice in the field of side channels, serving a similar purpose to the MNIST database in creating a comprehensive dataset. This dataset comprises power traces from the initial round of a software AES implementation executed on the ATMega8515 device, which has been subjected to mask protection.
- *TinyPower dataset* Its target chip is the STM32F3, and the acquisition platform is the ChipWhisperer. It is not mask protected. Random noise with a mean of 0 and a variance of 5 was added to the dataset in the experiments, making it more closely match real-world scenarios.

The specifics of the dataset used are delineated in Table 2.

Table 2 ASCAD (fixed) and TinyPower dataset parameters

Parameter	Value
Profiling_traces	50,000
Attack_traces	10,000
Raw_dim	100,000
Select_dim	1200 (3*400)
Attack byte subscripts	2,3,4

Table 3 Relationship between power intervals and corresponding bytes on the original dataset

Position	Intervals	
	ASCAD	TinyPower
Third byte	47,000–47,400	6950–7350
Fourth byte	34,500–34,900	6490–6890
Fifth byte	49,088–49,488	9400–9800

A signal-to-noise ratio analysis is necessary when the power traces are captured. This is a standard method of analysis for side-channel attacks. The leakage information for the corresponding bytes can be obtained through the calculation of the SNR of the leaked power traces. The formula for the SNR is as follows:

$$\mathrm{SNR} = \frac{\mathrm{Var}[E[T_i|Z]]}{E[\mathrm{Var}[T_i|Z]]} \tag{11}$$

The original ASCAD dataset or TinyPower dataset is represented by T in Equation (11). Each column of data, denoted as T_i, can be divided into 256 classes based on the division of the middle value Z. The variance and expectation of these 256 classes are subsequently calculated. Finally, the expectation of the variance and the variance of the expectation are computed separately. The signal-to-noise ratio is then obtained by dividing these two quantities.

This study extracts three bytes of leaked power intervals, analyzes the signal-to-noise ratio of the entire raw ASCAD dataset and TinyPower dataset, and compresses the power points that leak three bytes into 1200 dimensions. The SNR for the keys with index as 2, 3, and 4 has also been extracted from the original ASCAD and TinyPower dataset, along with the SNR with S-boxes in ASCAD. The leaked power intervals and exposed bytes are presented in Table 3.

Our experimental implementations are based on Keras library with Tensorflow backend using Python scripts, and run on an Intel Core i7-12650 H CPU @ 4.70 GHz with one NVIDIA GTX 3060 GPU.

5.2 The feature selection for the model

To evaluate the model's performance, we employ visualization tool, which helps analyze the TripM model's feature extraction capabilities. In their study, Pei Cao [8] et al. introduced Layer-wise Correlation (LWC) as a method to explain network performance. LWC serves two main purposes:

- It visualizes the feature selection process.
- It reveals if the network is combining leakage features.

This approach is crucial for validating the TripM architecture's attack efficiency. To present our findings clearly, we use AVE-LWC, a variant of LWC, to assess the TripM model's feature extraction capabilities. AVE-LWC is calculated by averaging the LWC of all neurons in a fully connected layer. Let us denote i as the i^{th} fully connected layer within the neural network architecture, n is the amount of the neuron in this layer, and k is the sampling point of the input data, the AVE-LWC can be calculated as follows:

$$AVE - LWC_i[k] = \frac{1}{n}\sum_{a=1}^{n} LWC_i[k] \tag{12}$$

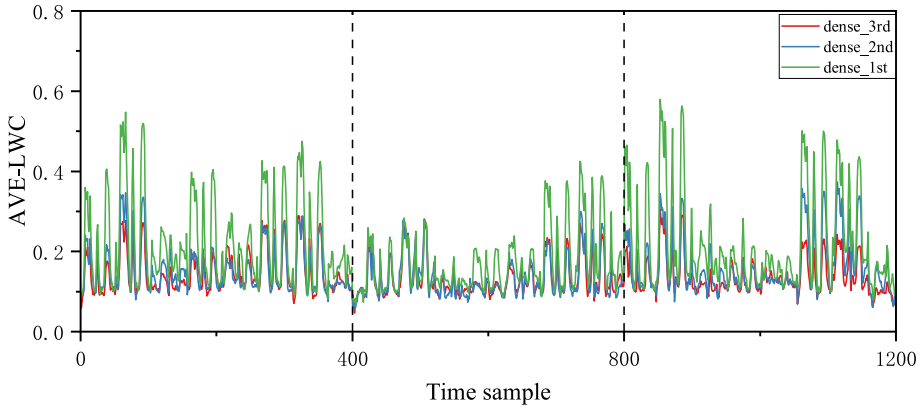
The TripM feature extraction capability is demonstrated through the visualization of AVE-LWC on the ASCAD and TinyPower dataset, as depicted in Figs. 5a and 6a. This figure illustrates the AVE-LWC results for the input traces of TripM, highlighting a high correlation between the AVE-LWC of each fully connected layer and the SNR(see Figs. 5b and 6b) spikes. Such correlation underscores the TripM's efficiency in extracting leakage information from the input traces, revealing the Points of Interest.

5.3 Result on ASCAD and TinyPower dataset

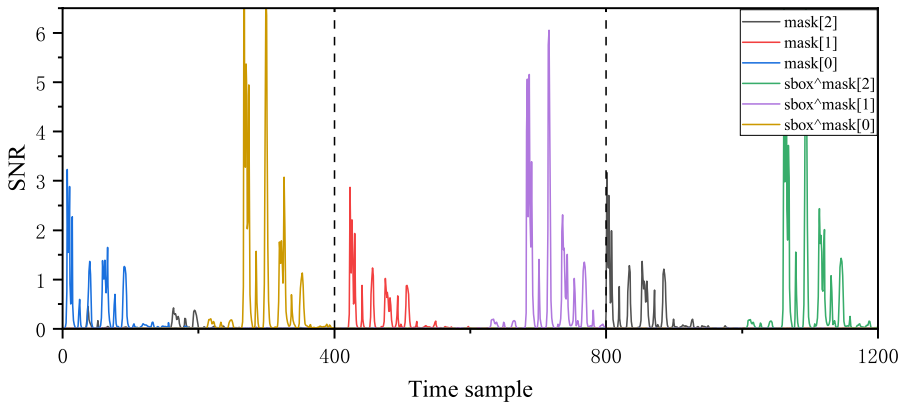
The TripM model effectively enhances the efficiency of attacks in the side-channel domain. Experiments are conducted on the ASCAD synchronization and TinyPower dataset, with the signal-to-noise ratio approach utilized to re-extract the necessary power traces.

Firstly, the average training time per byte was examined on the ASCAD dataset. The model's input and output layers were modified to accommodate both single-byte and multi-byte attacks, and the training time was recorded before and after the modification (Table 4).

Experimental results demonstrate that, on average, tripM requires 73.62 s to train the model for one byte under a multi-byte attack condition, compared to 127.19 s under a single-byte attack. This represents a reduction in training time of approximately 42.12% when in a multi-byte attack



(a) LWC_sel on fully-connected layers



(b) The SNR value of input data

Fig. 5 The SNR value of input data and the LWC value of the fully-connected layers on the ASCAD dataset

state, thereby enhancing the efficiency of side-channel recovery.

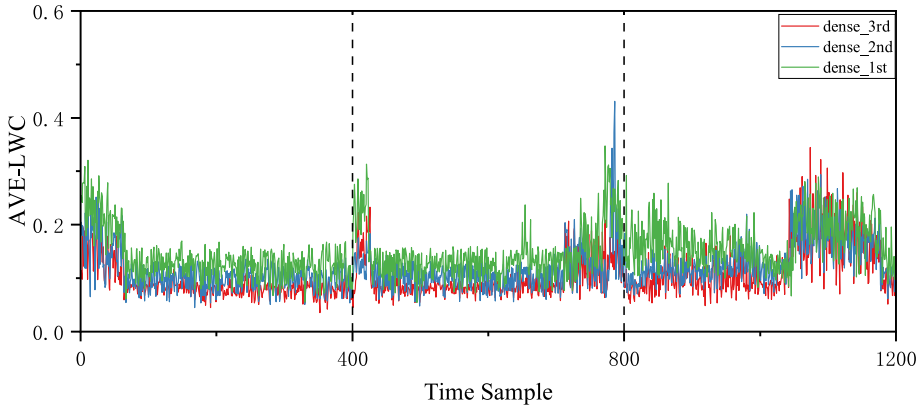
Next, the performance of TripM was evaluated on the ASCAD dataset. The attack result depicted as 7, and indicate that, on average, the recovery of the key indexed as 2 in the ASCAD dataset necessitates 58 traces, the key indexed as 3 requires 138 traces and the key indexed as 4 necessitates only 43 traces. Table 5 depicts the average number of traces required by the tripM model to recover three keys, compared with those reported in state-of-art model. The results show the ability of the tripM model to recover keys using fewer traces. Finally, the performance of TripM was tested on the TinyPower dataset, and the results are shown in Figure. 7.

The results show that the TripM model can recover the key indexed as 2 with 169 traces, the key indexed as 3 with 95 traces, and the key indexed as 4 with 5 traces. The average number of traces required by the TripM model to recover three keys is 89. This result is attributed to the differences in signal-to-noise ratios(see Figs. 5b and 6b), which affect the trees needed to recover different bytes. For example, the bytes index as 2 in the noisy TinyPower dataset expose very little snr among the input traces, so the highest number of traces is required, whereas the bytes index as 4 expose the most signal-to-noise, so only 5 traces are required. This also can prove that the SNR can directly affect the effectiveness of the deep learning model learning.

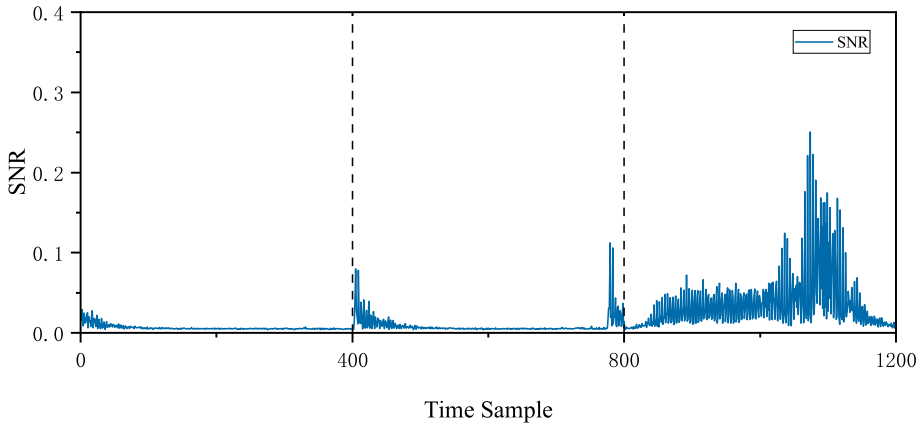
The comparison of the number of power traces required by TripM and TinyPower is shown in Table 6.

5.4 The use of weight sharing

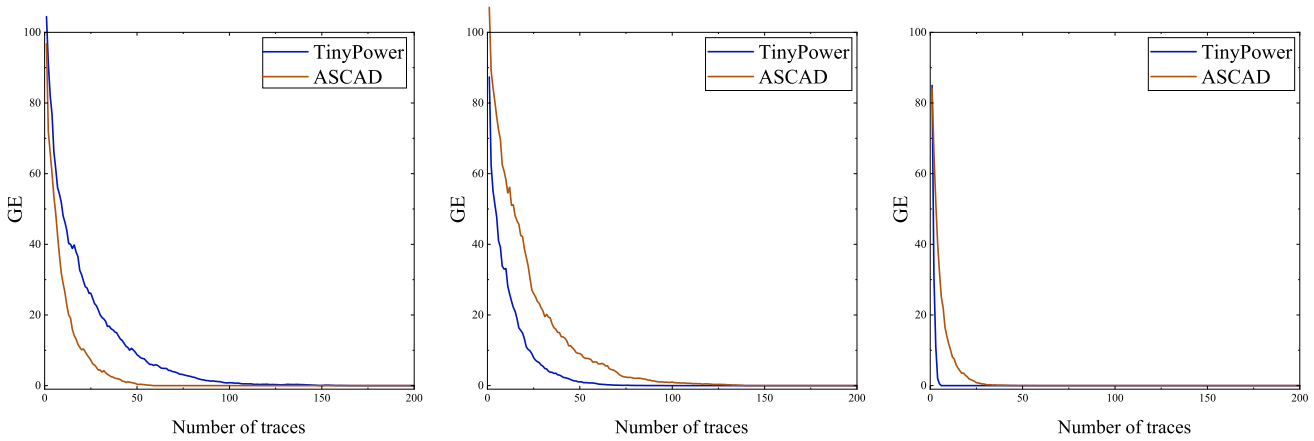
The architecture of the TripM model includes two identical branches. The use of the weight sharing technique in these branches effectively reduces the number of parameters, which in turn accelerates the training speed of the model. Both single-byte and multi-byte attacks can be implemented by modifying the input and output structure of the same model. For comparison, the training time and parameter count for each attack method are documented in this section.



(a) LWC_sel on fully-connected layers



(b) The SNR value of input data



(a) Index as 2 (b) Index as 3 (c) Index as 4

Fig. 6 The SNR value of input data and the LWC value of the fully-connected layers on the TinyPower dataset

Fig. 7 GE-traces figures of the three bytes on the ASCAD and TinyPower dataset

Table 4 Average training time required for a single byte (in s)

	Learning time
Multi byte	73.62 (220.85/3)
Single byte	127.19
Improvement percentage	42.12%

Table 5 Comparison of the number of power traces required by TripM and other models

Reference	Architectures	Label	Ns
[12]	CNN	One-hot	230
[30]	CNN	One-hot	171
[28]	CNN	One-hot	≈ 200
[31]	CNN	One-hot	155
[8]	CNN	One-hot	95
[32]	CNN	One-hot	232
[16]	MLP	One-hot	253
[16]	CNN	One-hot	157
[15]	Transformer	One-hot	214
[7]	MLP	Distributed	182
[7]	CNN	Distributed	87
[23]	CNN	Multi-label	202
This work	CNN	Multi-label	80

The average GE value of three bytes is used as the comparison data in this work

Table 6 Comparison of GE for different datasets

Index	ASCAD	TinyPower
2	58	169
3	138	95
4	43	5

Table 7 Comparison of training time with and without weight sharing (in s)

	Multi-byte	Single-byte
With weight sharing	220.85	127.19
Without weight sharing	397.64	203.34
Improvement percentage	44.46%	37.45%

A shared weighting technique was employed in TripM, effectively reducing the model's parameters without altering its structure. The number of model parameters decreased by 13.14% for multi-byte attacks and 23.92% for single-byte attacks, following modifications to the model's input and output.

Training time is decreased directly by the reduction in model parameters. Our experiments demonstrated that the training speed of the same model improved by 44.46% under a multi-byte attack, and by 37.45% under a single-byte attack. This technique effectively mitigates two major drawbacks associated with multi-byte attacks on the model: an increase in the number of parameters and an extended training time. Those results are summarized in Tables 7 and 8.

Table 8 Comparison of parameters after using weight sharing

	Multi-byte	Single-byte
With weight sharing	65,037	35,725
Without weight sharing	73,581	44,269
Improvement percentage	13.14%	23.92%

Table 9 Impact of weight-sharing on GE

	Index as 2	Index as 3	Index as 4
With weight sharing	58	138	43
Without weight sharing	126	189	93
Improvement percentage	54.0%	26.9%	53.8%

Table 10 Experimental variables

Variable	Value
n_attack	100
n_traces	200

Finally, we examined the change brought by weight sharing technique in the model attack efficiency, which can assess the impact of this technique on GE. The experimental results are presented in Table 9.

It can be observed that the weight sharing technique can reduce training time by decreasing the number of model parameters, according to combining Tables 7, 8 and 9. This reduction in turn can further reduce the full-round attack time of the cryptographic algorithm. This design enables tripM to process multiple bytes of information simultaneously without a significant increase in computational complexity. When two identical branches of tripM do not utilize the weight reuse technique, there is an increase in GE to recover all three bytes, thereby reducing the efficiency of the model's attack.

5.5 Experimental results of multi-threading

The multi-threading module of Python has seldom been utilized in the key recovery phase of side-channel analysis, according to previous studies. This study marks the application of the multithread module to the recovery process of TripM. During the attack phase, we initially establish the experimental variables, as depicted in Table 10.

Table 11 Key recovery speed under multithreading (in s)

	Time taken (s)
First byte	14.5
Second byte	14.7
Third byte	15.2

Table 12 Hyperparameters for the deep-TripM model

Parameters	Value
Convolution layers	{5, 10, 15}
Filters	{2, 4, 8, 16, 32, 64, 128}
Kernel size	{1, 3, 5, 10, 25, 50, 100}
Fully-connected layers	3
Fully-connected neurons	25

Table 10 uses n_{attack} to denote the number of times the GE calculation is performed during the recovery phase. The final GE result is the average of 100 GE calculations. n_{traces} refers to the number of attack sets used in each GE calculation. This study involved the random selection of 200 traces from the attack set for each GE calculation.

The speed of recovering three bytes using Python's multithreading module is presented in Table 11.

The experimental results demonstrate successful recovery of a 3-byte key within 15.2 s, averaging a recovery time of merely 5.1 (15.2/3) seconds per byte.

5.6 Comparison with deep-tripM

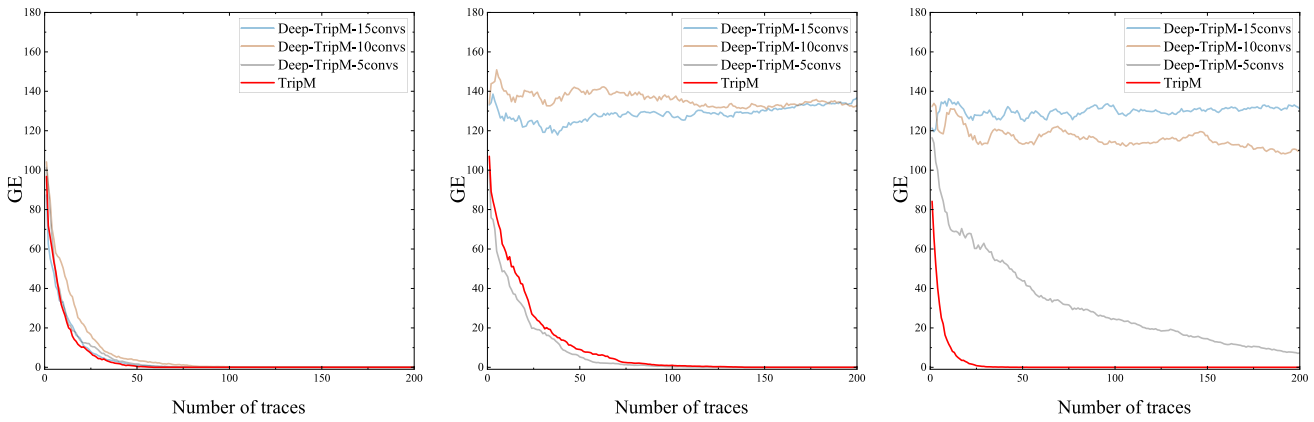
Deep-TripM was be designed to explore the influence of the depth of CNN structure on GE. Structurally, deep-TripM still utilizes a dual-channel architecture and employs weight reuse technology. The hyperparameters are all determined by the hyperparameter search space presented in Table 12. All the experimental data related to deep-TripM are obtained from the optimal model derived through this hyperparameter search.

The experimental results are presented in Fig. 8 and show that the effectiveness of the model's attack is not significantly enhanced as the number of convolutional layers increases, especially when attacking bytes with index 3 and 4. When the convolution depth reaches 5, the traces required to recover the bytes with subscript 4 increases substantially, while the bytes with subscript 3 and 4 cannot be recovered when the convolution depth reaches 10 and 15. This is because as the depth of convolution increases, the model becomes more complex and therefore introduces more noise, making the model less efficient at recovering keys. Therefore, the model structure of tripM can effectively reduce the complexity of the model and thus increase the efficiency of key recovery.

6 Discussion

6.1 Reasons for choosing to attack three bytes

TripM is an DL-SCA attack model for three bytes, which is mainly due to the consideration of less time-consuming full-round attacks. Attackers often consider all bytes attacks in real-world attack scenarios, so full-round attack time is more be concerned. Under the consideration of only training time and multi-threaded concurrent attack time, considering the model can only recover one byte, we need to train it 16 times theoretically, thus 16 attack iterations time are needed. When considering that the model can recover two bytes, 8 (16/2) attack iterations are needed, and when considering three bytes, it takes 5 iterations and a one-byte attack time,



(a) Index as 2 (b) Index as 3 (c) Index as 4

Fig. 8 Comparison of different convolutional layers of TripM

Table 13 Comparison of different attack bytes

	Attack 1 byte	Attack 2 bytes	Attack 3 bytes	Attack 4 bytes
Learning time (s)	98	156	220	380
Attack time (s)	15	15	15	15
Total time (s)	113	171	235	395
Full round attack time (s)	113 * 16 = 1808	176 * 8 = 1368	235 * 5 + 113 * 1 = 1288	395 * 4 = 1580

and so on. We record the time required by tripM in recovering 1, 2, 3, and 4 bytes as shown in Table 13, respectively.

As shown in Table 13, the training time is minimal for a single byte, but full-round attacks takes more time. Both the number of input samples to the model and the number of label groups increase as the number of bytes attacked in each iteration increases. Consequently, the model's complexity rises, leading to an increase in the training time for each round. However, we find that attacking three bytes requires the least amount of time for a full round attack when full round attacks are considered. The full round attack time increases when the size of attack bytes more than 3 bytes. Therefore, TripM chooses to attack three bytes in an attack iteration because attacking three bytes is experimentally proven to be the less time-consuming and higher efficiency for a full round of attacks.

6.2 Comparison with Non-Deep Learning Methodology

Traditional non-modelling attacks include DPA and SPA. They rely on statistical analyses such as mean, variance, and correlation to identify leakage in power consumption signals. These methods are simple to implement, have low computational cost, and can quickly identify significant features, but are less accurate and inflexible when dealing with complex or highly noisy signals.

Template attacks, on the other hand, use previously collected power consumption data to construct models to predict the power consumption behaviour of the target device. While template attacks can achieve high recovery accuracy under certain conditions and are effective for specific algorithms and devices, they are limited by the need for large amounts of training data and accurate template construction, and are less adaptable to dynamically changing environments.

The TripM model has several advantages compared to these non-deep learning methods. Firstly, TripM is able to automatically learn complex features in signals through training and can adapt to different attack scenarios. Second, TripM can recover multiple bytes at the same time, which effectively improves the attack efficiency and reduces the training and execution time. However, TripM also has its limitations, mainly in the higher demand for computational resources, especially in the multi-threaded attack phase. Additionally, the model requires a large amount of training data to achieve high accuracy.

6.3 Security Implications of TripM

TripM has a significant impact on cryptographic security. On the one hand, TripM improves the efficiency of key attacks by being able to recover multiple bytes at the same time, which exposes existing cryptographic algorithms such as AES to greater security threats. The ability of the attacker to recover the key in a shorter period of time may lead to its wider application in practical attacks.

There are some potential countermeasures to face the challenges brought by TripM. Firstly, physical security can be enhanced by adopting protective measures such as physical isolation, shielding or encapsulation to reduce the success rate of side-channel attacks. Second, security practitioners can consider introducing noise in the power signal to mask useful information and reduce the attacker's ability to analyse the signal. Further, introducing masking techniques to hide the intermediate result by performing XOR operations or other operations between the original data and a random mask value.

TripM also have a profound impact on future cryptographic algorithms. Cryptographic algorithms may need to incorporate security evaluation criteria against side-channel attacks in their design and implementation to ensure the algorithms' robustness under multiple attack scenarios.

6.4 Simplified implementation of concurrent recovery

Single-threaded concurrency in thread-constrained environments can be achieved in several ways.

- *Coroutines* Coroutines provide a lightweight implementation of concurrency, allowing for suspension and resumption during execution. This enables switching between multiple tasks without relying on multiple threads. Coroutines are particularly efficient for I/O-intensive tasks.
- *Asynchronous programming* This approach uses event loops and callbacks to handle multiple tasks in a single thread. It schedules tasks to execute when I/O operations

or other time-consuming tasks are completed, avoiding blocking the main thread.
- *Task queue* Tasks are placed into a queue sequentially, and the main thread executes them one by one. Although execution is sequential, concurrent processing can be simulated through efficient task scheduling.

7 Conclusion

The TripM model, a CNN designed to recover three bytes simultaneously, is introduced. The architecture of TripM comprises two identical convolutional branches, using a weight reuse technique to ensure identical weights across both branches. The concept of label groups is proposed, its feasibility in the SCA domain is demonstrated, and it is applied to the label selection of tripM. Additionally, a multithreaded attack method is proposed.

Experimental results indicate that the weight reuse technique significantly reduces the number of model parameters. This reduction leads to a decrease in the training time for a single byte by 44.46%. Additionally, the multi-byte attack strategy reduces the number of model parameters by 16.33% and decreases the average single-byte training time by 42.12%. The experiments also demonstrated that the model's attack efficiency increased by an average of 44.9% after implementing weight sharing. TripM approaches were conducted on two datasets, ASCAD and TinyPower. TripM model requires an average of 80 traces for the recovery of one byte on ASCAD dataset and 89 on TinyPower. The model's feature extraction capability is analyzed using AVE-LWC techniques. A deeper version, named Deep-TripM, indicates that increasing the number of CNN layers does not necessarily improve the model's attack efficiency.

Future studies might focus on recovering more keys in a single training session. The concept of label groups also offers a solution for executing multi-byte attacks.

Acknowledgements This research was supported by the Hunan Provincial Natural Science Foundation of China (Grant No. 2022JJ30103), Postgraduate Scientific Research Innovation Project of Hunan Province (CX20240977), "The 14th Five-Year Plan" Key Disciplines and Application-oriented Special Disciplines of Hunan Province (Grant No. Xiangjiaotong [2022] 351), and the Science and Technology Innovation Program of Hunan Province (Grant No. 2016TP1020).

Author Contributions Lianrui Deng: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing, Visualization, Reviewing and editing. Lang Li: Methodology, Investigation, Resources, Writing, Supervision, Project administration. Yu Ou: Validation, Writing. Jiahao Xiang: Methodology, Software, Resources, Writing, Visualization. Shengcheng Xia: Conceptualization, Data curation.

Data Availability The data that support the findings of this study are openly available in ASCAD at https://github.com/ANSSI-FR/ASCAD.

Declarations

Conflict of interest No potential Conflict of interest was reported by the authors.

References

1. Katz E, Avital M, Weizman Y et al (2023) Analytical side channel EM models, extending simulation abilities for ics, and linking physical models to cryptographic metrics. IEEE Trans Comput Aided Des Integr Circuits Syst 42(12):4463–4476
2. Cleemput JV, Sutter BD, Bosschere KD (2020) Adaptive compiler strategies for mitigating timing side channel attacks. IEEE Trans Depend Secure Comput 17:35–49
3. Baseri Y, Chouhan V, Hafid A (2024) Navigating quantum security risks in networked environments: a comprehensive study of quantum-safe network protocols. Comput Secur 142:103883
4. Pourrahmani H, Yavarinasab A, Monazzah AMH et al (2023) A review of the security vulnerabilities and countermeasures in the internet of things solutions: a bright future for the blockchain. Internet Things 23:100888
5. Bout E, Loscri V, Gallais A (2022) How machine learning changes the nature of cyberattacks on iot networks: a survey. IEEE Commun Surv Tutor 24(1):248–279
6. Hettwer B, Gehrer S, Güneysu T (2020) Applications of machine learning techniques in side-channel attacks: a survey. J Cryptogr Eng 10(2):135–162
7. Wu L, Weissbart L, Krček M et al (2023) Label correlation in deep learning-based side-channel analysis. IEEE Trans Inf Forensics Secur 18:3849–3861
8. Cao P, Zhang C, Lu X et al (2022) Improving deep learning based second-order side-channel analysis with bilinear cnn. IEEE Trans Inf Forensics Secur 17:3863–3876
9. Kwon D, Hong S, Kim H (2022) Optimizing implementations of non-profiled deep learning-based side-channel attacks. IEEE Access 10:5957–5967
10. Hou S, Zhou Y, Liu H et al (2017) Wavelet support vector machine algorithm in power analysis attacks. Radioengineering 26(3):890–902
11. Li D, Li L, Ou Y (2024) Side-channel analysis based on siamese neural network. J Supercomput 80(4):4423–4450
12. Zaid G, Bossuet L, Habrard A et al (2020) Methodology for efficient cnn architectures in profiling attacks. IACR Trans Cryptogr Hardw Embedded Syst 1–36
13. Maghrebi H, Portigliatti T, Prouff E (2016) Breaking cryptographic implementations using deep learning techniques. Security, privacy, and applied cryptography engineering. Springer International Publishing, Cham, pp 3–26
14. Xiangliang M, Bing L, Hong W et al (2021) Non-profiled deep-learning-based power analysis of the sm4 and des algorithms. Chin J Electron 30(3):500–507
15. Hajra S, Saha S, Alam M et al (2022) Transnet: shift invariant transformer network for side channel analysis. In: International conference on cryptology in Africa. Springer, pp 371–396
16. Benadjila R, Prouff E, Strullu R et al (2020) Deep learning for side-channel analysis and introduction to ascad database. Journal of Cryptographic Engineering 10(2):163–188
17. Thapar D, Alam M, Mukhopadhyay D (2021) Deep learning assisted cross-family profiled side-channel attacks using transfer learning. In: International symposium on quality electronic design (ISQED). IEEE, San Jose, CA, USA, pp 178–185
18. Ni L, Wang P, Zhang Y et al (2023) Profiling side-channel attacks based on cnn model fusion. Microelectron J 139:105901

19. Cagli E, Dumas C, Prouff E (2017) Convolutional neural networks with data augmentation against jitter-based countermeasures: profiling attacks without pre-processing. In: Cryptographic hardware and embedded systems. Springer, Taipei, pp 45–68
20. Ahmed AA, Hasan MK, Islam S et al (2023) Design of convolutional neural networks architecture for non-profiled side-channel attack detection. Elektronika Ir Elektrotechnika 29(4):76–81
21. Do NT, Hoang VP, Doan VS (2024) A novel non-profiled side channel attack based on multi-output regression neural network. J Cryptogr Eng 14(3):427–439
22. Faraji M, Seyedi SA, Tab FA et al (2024) Multi-label feature selection with global and local label correlation. Expert Syst Appl 246:123198
23. Zhang L, Xing X, Fan J et al (2020) Multilabel deep learning-based side-channel attack. IEEE Trans Comput Aided Des Integr Circuits Syst 40(6):1207–1216
24. Fukuda Y, Yoshida K, Hashimoto H et al (2023) Profiling deep learning side-channel attacks using multi-label against aes circuits with rsm countermeasure. IEICE Trans Fundam Electron Commun Comput Sci 106(3):294–305
25. Alzubaidi L, Zhang J, Humaidi AJ et al (2021) Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. J Big Data 8:1–74
26. Dong S, Wang P, Abbas K (2021) A survey on deep learning and its applications. Comput Sci Rev 40:100379
27. Picek S, Heuser A, Jovic A et al (2019) The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Trans Cryptogr Hardw Embedded Syst 1:209–237
28. Wouters L, Arribas V, Gierlichs B, et al (2020) Revisiting a methodology for efficient cnn architectures in profiling attacks. IACR Trans Cryptogr Hardw Embedded Syst 147–168
29. Li H, Ninan M, Wang B et al (2024) Tinypower: side-channel attacks with tiny neural networks. In: 2024 IEEE international symposium on hardware oriented security and trust (HOST), IEEE, pp 320–331
30. Rijsdijk J, Wu L, Perin G et al (2021) Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans Cryptogr Hardw Embedded Syst 3:677–707
31. Wu L, Perin G, Picek S (2024) I choose you: automated hyperparameter tuning for deep learning-based side-channel analysis. IEEE Trans Emerg Top Comput 12(2):546–557
32. Luo Z, Zheng M, Wang P et al (2021) Towards strengthening deep learning-based side channel attacks with mixup. In: International conference on trust, security and privacy in computing and communications (TrustCom). IEEE, Guangzhou, China, pp 791–801