Multi-Leak Deep-Learning Side-Channel Analysis

Fanliang Hu
School of Physics and
Electronic Science
Hunan University of Science
and Technology
Xiangtan, Hunan, China
Email: fanliang@mail.hnust.edu.cn

Huanyu Wang
School of EECS
KTH Royal Institute of Technology
Stockholm, Sweden
Email: huanyu@kth.se

Junnian Wang
School of Physics and
Electronic Science
Hunan University of Science
and Technology
Xiangtan, Hunan, China
Email: jnwang@mail.hnust.edu.cn

***Abstract*—Deep Learning Side-Channel Attacks (DLSCAs) have become a realistic threat to implementations of cryptographic algorithms, such as Advanced Encryption Standard (AES). By utilizing deep-learning models to analyze side-channel measurements, the attacker is able to derive the secret key of the cryptographic alrgorithm. However, when traces have multiple leakage intervals for a specific attack point, the majority of existing works train neural networks on these traces directly, without a preprocess step for each leakage interval. This degenerates the quality of profiling traces due to noise and non-primary components. In this paper, we first divide the multi-leaky traces into leakage intervals and train models on different intervals separately. Afterwards, we concatenate these neural networks to build the final network, which is called multi-input model. We test the proposed multi-input model on traces captured from STM32F3 microcontroller implementations of AES-128 and show a 2-fold improvement over the previous single-input attacks.**

***Index Terms*—Side-channel attacks, Multiple leakage, Multi-input model, AES, Deep learning**

I. Introduction

Side Channel Attacks (SCAs) [1] are first proposed over 20 years ago and have become a realistic concern in the hardware security community. By analysing the unintentional physical leakage during the execution of the cryptographic algorithms, SCAs are able to break ciphers that are assumed to be mathematically secure. Since Kocher in 1996 [1] introduced the attack to reveal secrets from time, many other types of leakage have been proposed. For example, leakages via acoustic channels [2], power consumption [3], electromagnetic (EM) emissions [4], [5], and photon emissions [6] are now widely studied.

Recently, with advances in deep learning [7], SCAs can be more effective than conventional cryptanalysis and is more practical to mount. Since well-trained deep learning models are good at extracting features from raw data, they can help the attacker to find correlations between physical measurements and the internal state of the processing algorithm. Maghrebi et al. first applies deep-learning techniques to side-channel attacks' context in 2016 [8], in which they investigate how Multi-layer Perceptron (MLP) and Convolutional Neural Network (CNN) could make the SCAs more efficient. Subsequently, many softwares [5], [9]–[11] and hardwares [12]–[15] implementated with AES have been broken by DLSCAs. In [16], Cagli et al. evaluates the CNN network's performance in datasets with jitter based on countermeasure. In [9], Huanyu et al. study the impact of how diversity of target chips affects side-channel attacks. In [17], the influence of the depth of the network model on DLSCAs is studied by visualising the heatmap. These papers provide strong evidence for the effectiveness of deep learning in the field of side channel attacks, demonstrating the powerful potential of deep learning for SCAs.

In most existing DLSCAs, neural networks are trained by traces labeled by a single attack point. The attack point is a certain point in the encryption algorithm where known information (e.g., plaintext, ciphertext) and physical leaks can be linked for key recovery (note: this point is a continuous state, e.g. the entire process of SBox's output). The link between the power traces and the intermediate state is used to build a model that can be used to recover the victimised device's keys. There are several different attack points in AES with varying degrees of information leakage on power traces (e.g. *SubBytes*, *ShiftRows*, *MixColumns* in the first round of AES and *AddRoundKey* before the first round of AES). This means that each attack point can be used to recover the key by building a network model. This may have been noted before, but the advantages of attack points corresponding to multiple leaks have not been fully explored. Therefore, this paper uses a multi-input model to explore the benefits of using multiple leaks to recover keys.

Notice: The phenomenon of multiple leaks refers to the fact that a single attack point is leaked multiple times in power traces.

A. Our Contributions

In this paper, our main contributions could be summaried as:

- We propose a new multi-input model structure applied to the case where multiple leaks exist in power traces. The *Concatenate layer* is used for the fusion of the input layers and then two different connection methods are compared.
- Our results show that there are multiple leaks when the SBox's output of the first round of the AES algorithm is the attack point. The same phenomenon happens in the last round of the AES algorithm. When the SBox's input

of the last round is used as the attack point, we can also get that it corresponds to multiple leaks.

- Byte 5, Byte 6, Byte 7 and Byte 8 are chosen for the experiments and the results show that leakage of *ShiftRows* has a positive effect on the model when the SBox's output is used as the attack point.
- Our experimental results on the STM32 implementation of AES demonstrate that when using a multi-input model to recover keys with multiple leakage in power traces, the last round will outperform the first round because there are two *AddRoundKey* phases in the last round being used for the trained multi-input model.
- Finally, we validate this method's generality using the Screaming Channel dataset [18] and the AES_GPU dataset [19], and show that multi-input model achieves a 2-fold improve.

B. Paper Organization

The rest of the paper is organised as follows. In Section II, we analyse the causes of multiple leakages and build the multi-input model for the experiments. In Section III, the dataset used for the experiments and the experimental results of the multi-input model for multi-leaky traces are briefly described and the results are analysed. We conclude our work and discuss future directions in Section IV.

II. Multi-leakage and Multi-input Model

This section first analyses the principles of leakage for different attack points in AES [20], then describes the reasons why a specific attack point corresponds to multiple leakages, and finally describes the structure of the multi-input model and how to use it to solve the multiple leakage problem.

A. Leakage Analysis

Power consumption of the cryptographic device mainly derives from the bit transitions in the CMOS cells. Thus data processed in the device dominate its power dissipation. As previous researchs on SCAs [8], [21], attackers have commonly chosen the output of the SBox as the attack point for recovering keys from device, based on the fact that the non-linear output of the SBox has a higher level of confusion. However, in the AES algorithm, not only the output of the SBox that can be used as a attack point, other phases of key-related information can also be used. Next, we analyse the first and last rounds of the AES algorithm according to the principles of Correlated Power Analysis (CPA). Then we can determine the attack points that can be used. As the attack points shown in Fig. 1, F_leak is attack points in the first round and L_leak attack points in the last round.

CPA is based on the Pearson Correlation Coefficient [22], a method that uses statistical power analysis on the correlation between real power consumption data obtained from the target device when it performs cryptographic operations and experimentally calculated hypothetical power consumption values. The critical aspect of CPA to recover the key is to determine the leakage function (i.e. the intermediate value in CPA and the label in deep learning) at the attack points in the AES algorithm.

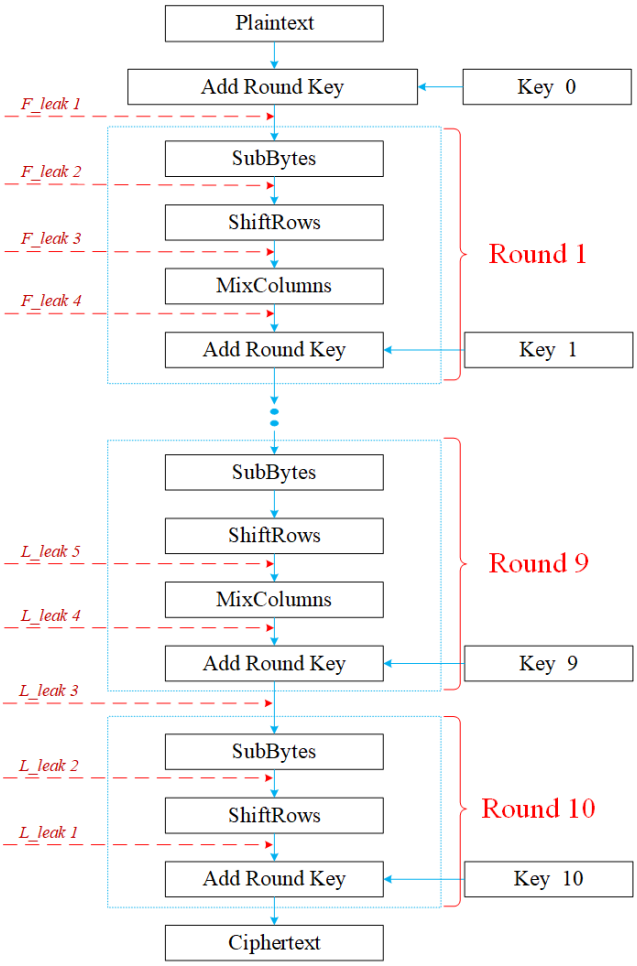

Fig. 1. Attack points.

In Fig. 1, $F\_leak\ 1$ represents the *AddRoundKey*'s output before the first round of the AES algorithm, and the CPA needs to construct the leakage function $V_{F\_leak\ 1}$ when it picks $F\_leak\ 1$ as the attack point, which is denoted by $V_{F\_leak\ 1}$:

$$V_{F\_leak\ 1} = Pt \oplus Key\ 0$$

Where Pt represents the plaintext and $Key\ 0$ represents the original key (the $Key\ i (i \in [1, 10])$ is obtained by the Key Expansion [20] of $Key\ 0$).

In Fig. 1, $F\_leak\ 2$ represents the output of the *SubBytes* in the first round of the AES algorithm (i.e. the output of the SBox), and the leakage function $V_{F\_leak\ 2}$ for $F\_leak\ 2$ as a attack point is expressed as:

$$V_{F\_leak\ 2} = SBox(Pt \oplus Key\ 0).$$

In Fig. 1, $F\_leak\ 3$ represents the output of the *ShiftRows* in the first round of the AES algorithm. The *ShiftRows* is a cyclic shift operation performed on different rows in the AES algorithm, keeping the first row unchanged, moving the second

row one byte to the left, the third row two bytes to the left, and the fourth row three bytes to the left. Thus the leakage function $V_{F\_leak\ 3}$ for $F\_leak\ 3$ as a point of attack is expressed as:

$$V_{F\_leak\ 3_i} = \begin{cases} V_{F\_leak\ 2_{i+0}} & i = 1, 5, 9, 13 \\ V_{F\_leak\ 2_{i+4}} & i = 2, 6, 10, 14 \\ V_{F\_leak\ 2_{i+8}} & i = 3, 7, 11, 15 \\ V_{F\_leak\ 2_{i+12}} & i = 4, 8, 12, 16 \end{cases}$$

Where i denotes the i-th byte, and $i + n = i + n - 16$ $(n = 0, 4, 8, 12)$ when $i + n > 16$ in the formula.

In Fig. 1, $F\_leak\ 4$ represents the output of the *MixColumns* in the first round of the AES algorithm. The *MixColumns* is achieved by multiplying matrices in a finite field GF(2^8) as follows, where $a = V_{F\_leak\ 3}$ denotes the middle state of the *ShiftRows*'s output and $b = V_{F\_leak\ 4}$ denotes the middle state of the *MixColumns*'s output.

$$\begin{bmatrix} b_{1,r} \\ b_{2,r} \\ b_{3,r} \\ b_{4,r} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{1,r} \\ a_{2,r} \\ a_{3,r} \\ a_{4,r} \end{bmatrix}.$$

Where $1, 2, 3, 4$ represent the rows in a 4×4 matrix and $r(r = 1, 2, 3, 4)$ is labelled as the column of the matrix. In the above equation, the a's left multiplication matrix is a fixed matrix of *MixColumns*. Therefore, when byte 5, byte 6, byte 7 and byte 8 in the second column are used as target bytes, the leakage function $V_{F\_leak\ 4}$ is expressed as:

$$V_{F\_leak\ 4_i} = \begin{cases} [2 \times a_5] \oplus [3 \times a_6] \oplus a_7 \oplus a_8 & i = 5 \\ a_5 \oplus [2 \times a_6] \oplus [3 \times a_7] \oplus a_8 & i = 6 \\ a_5 \oplus a_6 \oplus [2 \times a_7] \oplus [3 \times a_8] & i = 7 \\ [3 \times a_5] \oplus a_6 \oplus a_7 \oplus [2 \times a_8] & i = 8 \end{cases}$$

Where $a = V_{F\_leak\ 3}$ and i represents the i-th byte. '$\times$' denotes a multiplication operation in a finite field and '$\oplus$' denotes the XOR operation.

In the last round of the AES algorithm, the leakage function V_{L_leak} is calculated in a similar way to the leakage function V_{F_leak} in the first round. The leakage function for the last round V_{L_leak} is shown in Table I. Notice that the relevant information for $Key10$ present in the last round and in round 9 at same time.

TABLE I
LEAKAGE FUNCTION CORRESPONDING TO DIFFERENT ATTACK POINTS IN THE LAST ROUND.

$V_{L\_leak\ 1}$	$Ct \oplus Key\ 10$
$V_{L\_leak\ 2}$	$V_{L\_leak\ 1_i}$ $\quad i = 1, 5, 9, 13$ $V_{L\_leak\ 1_{i+12}}$ $\quad i = 2, 6, 10, 14$ $V_{L\_leak\ 1_{i+8}}$ $\quad i = 3, 7, 11, 15$ $V_{L\_leak\ 1_{i+4}}$ $\quad i = 4, 8, 12, 16$
$V_{L\_leak\ 3}$	$SBox^{-1}(V_{L\_leak\ 2})$
$V_{L\_leak\ 4}$	$V_{L\_leak\ 3} \oplus Key\ 9$
$V_{L\_leak\ 5_i}$ $i \in [5, 8]$	$[E \times a_5] \oplus [B \times a_6] \oplus [D \times a_7] \oplus [9 \times a_8]$ $[9 \times a_5] \oplus [E \times a_6] \oplus [B \times a_7] \oplus [D \times a_8]$ $[D \times a_5] \oplus [9 \times a_6] \oplus [E \times a_7] \oplus [B \times a_8]$ $[B \times a_5] \oplus [D \times a_6] \oplus [9 \times a_7] \oplus [E \times a_8]$

Where Ct is the ciphertext, $Key\ 10$ is the extended key for round 10, $Key\ 9$ is the extended key for round 9, and $SBox^{-1}$ denotes the inverse of SBox. In the last row of Table I, a_i is used in place of $V_{L\_leak\ 4}$ and $i(i = 5, 6, 7, 8)$ represents number of the bytes. In Table I, $9, B, D, E$ are the hexadecimal values in the inverse *MixColumns*'s left multiplication matrix in AES.

B. Label for Multi-input Model

The attacker has build leakage models in order to characterise the relationship between power consumption and attack points. Hamming Weight (HW) and Hamming Distance (HD) models are reasonable estimations but suffer from the issue of class imbalance in practice owing to Bernoulli Distribution [23]. The value model (identity (ID) model) assumes the power consumed by the device is propotional to the data processed at the attack point. We use ID model in our experiment to distinguish different power traces.

There is a correspondence between the attack point and the leakage function (the leakage function are defined as label in deep learning), and we can build a network model for recovering the key according to each attack point. However, in the process of this attack, the connection between different attack points in the AES algorithm and the feature that each attack point is correlated with the same key are not taken into account. In order to combine information from multiple attack points, we propose a multi-input model. Because each attack point corresponds to a different leakage function (e.g. the leakage function when the output of the *AddRoundKey* is used as the attack point is different from the leakage function of the SBox's output), and because there is only one output in the multi-input model, the leakage function V_{F_leak} of multiple attack points need to be unified. The following describes the way to unify the leak functions of different attack points and the reason why multiple leakages can exist at one attack point.

The first round of AES is used as an example to investigate the relationship between leakage functions at different attack points. We use the leakage function of $V_{F\_leak\ 2}$ instead of the other leakage functions as label for the multi-input model. By replacing $V_{F\_leak\ 1}$ with $V_{F\_leak\ 2}$, the one-to-one non-linear transformation of SBox does not affect the classification of the network model (e.g. after replacing all the labels of cats with dogs and all the labels of dogs with pigs in image classification, results show that the accuracy of the network model training does not change). The *ShiftRows*'s leakage function simply shifts $V_{F\_leak\ 2}$ without changing it, so $V_{F\_leak\ 2}$ can be used instead of $V_{F\_leak\ 3}$. The leakage function $V_{F\_leak\ 4}$ for one byte in *MixColumns* is obtained from four different bytes of the $V_{F\_leak\ 2}$ by the XOR operation, and $V_{F\_leak\ 2}$ is used as part of the *MixColumns* leakage function, so $V_{F\_leak\ 2}$ can be used instead of $V_{F\_leak\ 4}$ (e.g. when the *AddRoundKey* is used as a specific attack point, the Key can be used as the model's label). The leakage function for multiple attack points is unified as $V_{F\_leak\ 2}$. This is the first step in building a multi-input model, which is described below.

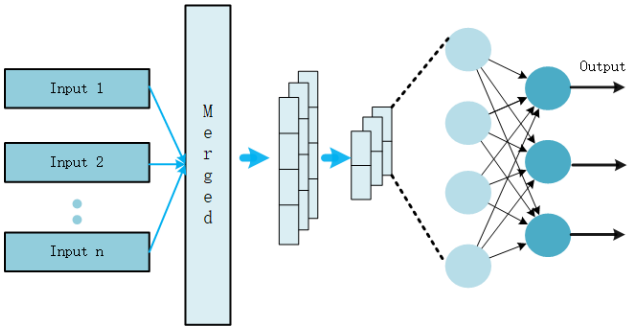

Fig. 2. Structural diagram of the multi-input model.

C. Construction of Multi-input Model

The basic architecture used in this work is a CNN with multiple input layers, as shown in Fig. 2. The multiple inputs are merged and connected to a *Convolutional layer* consisting of 32 neurons, then they are expanded by *Flatten layer* after passing through a three-strides *MaxPooling layer*. Afterwards, they are connected to two *Dense layers* of 128 neurons. Finally, a *Dense layer* with a *Softmax* activation function is used to generate 256 output predictions. The *Convolutional layer* and *Dense layers* are activated by function with Rectified Linear Units (*ReLU*).

Using the *Merged layer* is an important aspect when building multi-input models, providing the DNN architecture with great flexibility. Two fusion techniques are commonly used in existing works are early fusion, and late fusion [24], [25] (i.e. early use of the merged layer and late use of the merged layer). Early fusion is the combination of multiple inputs which are then connected to the first layer of the DNN. In the late fusion architecture, features are first extracted from the input data of individual channels. The specific information of the channels is eventually merged and processed in further network model layers responsible for the classification based on the extracted features.

We find that the late fusion network model is less accurate than the early fusion network model. Therefore, in the paper we only conduct experiments for the early fusion network model.

There are furthermore different types of methods to merge layers within DNN architectures, which are explained below:

- `Add`: Returns the element-wise sum of two inputs
- `Subtract`: Returns element-wise subtracts two inputs
- `Multiply`: Returns element-wise multiplication of inputs
- `Average`: Returns element-wise average of the inputs
- `Maximum`: Returns element-wise maximum of the inputs
- `Minimum`: Returns element-wise minimum of the inputs
- `Concatenate`: Returns concatenation of the inputs

In this paper, we choose the most commonly used *Concatenate layer* for our own experiments.

D. Method for Multi-input Model

In the AES_GPU [26], when the author use a template attack to recover the last byte of the key, they find that there are two discontinuous leakage intervals of that byte. However, the authors only choose the segment with higher Signal-to-Noise Ratio (SNR) [27] power traces as the experimental interval and don't consider about the impact of the segment with lower SNR on recovering the key. In the Screaming Channel dataset [18], the authors also find a specific attack point corresponding to multiple different leakage intervals when looking for POI but they choose only the highest SNR interval.

These works are two common ways of handling multiple leaks when the leaks are found. When different leakage intervals are close, the power traces of all the leakage intervals can be intercepted and used for experiments. When different leakage intervals are not close to each other, the interval with higher SNR is usually chosen as the experimental analysis interval. But the disadvantages of both methods are obvious. When a partially leakage interval is selected for experimentation, useful information for recovering the key may lost. When the leakage intervals are all selected, the intervals contain a lot of non-essential information, useless information or even interfering information for recovering the key. However, when a specific attack point corresponds to multiple discontinuous leakage intervals, we train a multi-input model by manually prioritising the extraction of each leaking segment and using the method in section II-B to make the same label for the leakage interval. This treatment makes full use of the information in each segment that is useful for key recovery, and by prioritising the extraction, also removes information that is not essential for key recovery.

Finally we apply the multi-input models to the AES_GPU and Screaming Channel datasets in Section III-F to obtain state-of-the-art results. Next we validate the advantages of the multi-input model on multiple leakage power traces for the STM32 implementation of AES.

III. Experimental Results

In this section, we first present the dataset used for the experiments and the evaluation metrics for the experiments. Then the validity of the multi-input model for multiple leakage power traces is verified. Finally, the experimental results are discussed and the experiments are validated on two other well-known datasets.

A. Dataset

The dataset used in the paper was captured by a ChipWhisperer-Lite [28] device at a sampling frequency of 40MHz. The experimental target cryptographic board is the CW308T-STM32F3, and the target cryptographic chip is the Arm Cortex M4, which runs the cryptographic algorithm TinyAES. The encryption mode of operation is the Electric Code Book (ECB) mode. For the first and last rounds of the AES algorithm, 60K power traces are captured as the experiment's dataset, in which, 50K are used as the training set used for the profiling phase using random plaintext and random key,

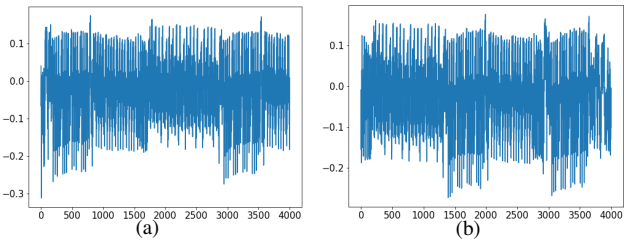

Fig. 3. (a) An example trace which represents the first round of AES; (b) The trace represents the last round of AES.

and 10K are used as the test set for the experiments using random plaintext and fixed key. Each power trace contains 4000 sampling points as shown in Fig. 3.

B. Evaluation Metrics

Model accuracy is defined as the probability of a model achieving correct classification results on a test set. As one of the most commonly used model evaluation metrics in machine learning, model accuracy is used to characterise a model's ability to classify data. An increase in model accuracy indicates that the backpropagation algorithm's optimization of the weights and bias parameters gradually converges to the correct values, and the model gradually converges to the optimal model. The loss of a model characterises the degree of deviation between a model's predicted and actual values. The smaller the loss is the closer the model's prediction is to the actual value. The loss function used in this experiment is the *Categorical Crossentropy*. The formula for the accuracy of the model is:

$$acc(X_{attack}) = \frac{|\{x_i \in X_{attack}\}\tilde{k}|}{X_{attack}}.$$

Where X_{attack} denotes the test dataset, x_i denotes the ith power trace in that dataset, $\tilde{k}$ denotes the calculation result, and $x_i \in X_{attack}$ is the set of all power traces when the guess keys are all equal to the correct key. The model's accuracy is the ratio of the number of power traces when the guessed key is equal to the correct key to the total number of power traces in all the test sets.

C. Results of The Multi-input Model on The First Round

In our experiments, we use the ρ-test as a leak detection method [29] to find the Point of Interest (POI) of each byte of the attack point. The POI of the first round of the AES algorithm is shown in Fig. 4. To simplify our experiments, we choose the 5th byte as the byte for the experimental demonstration. The validation of other bytes is also the same.

Fig. 4 shows five regions divided by red lines as A, B, C, D and E, representing the *AddRoundKey* before the first round of the AES algorithm, and the *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey* in the first round. We use the same leakage function (i.e. the leakage function for the first round of SBox's output which is noted as S_{out}), and it is easy to see from Fig. 4 that the E part (i.e. the *AddRoundKey* in

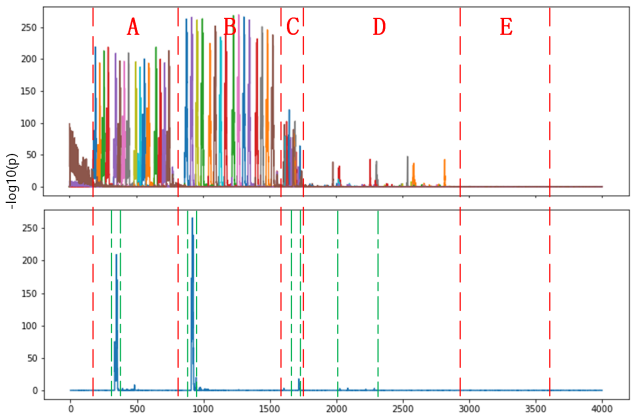

Fig. 4. POI of all bytes of the first round of the AES algorithm and the POI of the 5th byte.

TABLE II
RESULT OF THE 5TH BYTE OF THE SINGLE INPUT MODEL.

Input section	POI interval	Accuracy
AddRoundKey (A)	[315:375]	11.40%
SubBytes (B)	[890:950]	48.27%
ShiftRows (C)	[1690:1750]	0.85%
MixColumns (D)	[2000:2300]	0.47%

the first round) does not contain the leaked information of the S_{out}. The key in the E (i.e. *AddRoundKey* in the first round) of the encryption process is the first round of keys, where the output of *MixColumns* is used as plaintext. From part D, it can be concluded that some of the S_{out} are still leaked in the *MixColumns*. However, the part E does not leak the S_{out} information, indicating that the S_{out} information does not continue leaking after the *MixColumns*, proving that the *MixColumns* has good protection for the S_{out}.

Next, the power traces and leakage function of byte 5 are matched by a single input model. The part selected by the green lines in Fig. 4 is the POI corresponding to byte 5. The results predicted by the trained model on the test set are shown in Table II.

As shown in Table II, the models can learn intrinsic information about power traces and leakage function at the point of attack from each part of the AES algorithm, which is used to recover the key-related leakage function (we consider the model to be effective on the SCAs when the accuracy of the network model on the test set is higher than $1/256 \approx 0.39\%$).

The joint training of multiple leaks is achieved by changing the inputs to the network. In the experiments, the order in which the inputs are added was based on the single-input model's accuracy on the test set from highest to lowest. For the *MixColumns* interval input network model, we use a 5-strides to change its length so that it is as long as other inputs. When fusing the input data using the *Concatenate layer*, the second (axis = 1) and third (axis = 2) dimensions of the input data are allowed to fuse by changing the fusion layer's parameters. Table III shows the results for byte 5, byte 6, byte 7, and byte 8 when using a multi-input model.

TABLE III
RESULTS FOR BYTE 5, BYTE 6, BYTE 7, AND BYTE 8 IN THE FIRST ROUND OF THE AES ALGORITHM USING MULTI-INPUT MODELS.

	Byte 5	Byte 6	Byte 7	Byte 8
POI intervals (AddRoundKey (A), SubBytes (B), ShiftRows (C), MixColumns (D))	[315:375] [890:950] [1690:1750] [2000:2300]	[345:405] [1070:1130] [1585:1645] [1720:2020]	[375:435] [1240:1300] [1645:1705] [2560:2860]	[410:470] [1415:1475] [1670:1730] [2270:2570]
1 - input models (Accuracy)	11.40%	21.01%	23.10%	18.04%
	48.27%	47.20%	53.77%	48.86%
	0.85%	7.63%	1.93%	4.04%
	0.47%	2.85%	1.32%	2.82%
2 - input models (axis = 1, axis = 2)	81.60%	83.27%	86.39%	81.11%
	82.67%	83.73%	85.62%	82.06%
3 - input models (axis = 1, axis = 2)	78.14%	88.29%	86.69%	85.30%
	82.51%	90.15%	86.71%	85.16%
4 - input models (axis = 1, axis = 2)	69.79%	83.79%	75.23%	69.48%
	82.87%	91.69%	87.38%	85.64%

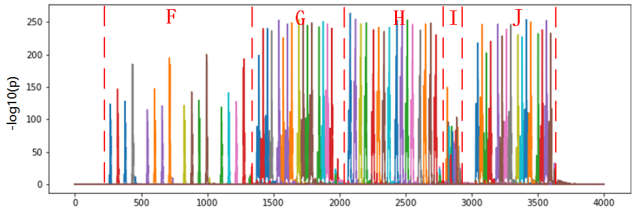

Fig. 5. POI for the last round of the AES algorithm.

From Table III, the accuracy of the models in predicting the correct key can be improved by using multi-input models when multiple leaks exist. When the fusion layer is being used, its parameters are changed so that the inputs to the models have two different states of the union. One is a serial (axis = 1) connection of the inputs, and the other is a parallel (axis = 2) connection of the inputs. This is the reason why we use *AveragePooling layer* for the *MixColumns* data to make the data length of the input models consistent. The two connection types used in the experiments will be discussed later. Next, we show the results of the last round of the AES algorithm for the multi-input models.

D. Results of The Multi-input Model on The Last Round

The POI for the last round of the AES algorithm is shown by Fig. 5. Byte 5, byte 6, byte 7, and byte 8 are the experimental target bytes, and the results are shown by Table IV.

The five regions J, I, H, G and F in Fig. 5 are divided by red lines, which represent the last round of *AddRoundKey*, *ShiftRows*, *SubBytes* and the 9th round of *AddRoundKey* and *MixColumns* of the AES algorithm. In the process of finding the POI, we used the same leakage function at the attack point (i.e. the input to the SBox in the last round which is noted as S_{in}). There are leaks after the 9th round of *ShiftRows*. However, the 9th round of *ShiftRows* does not leak information about the S_{in}, which again proves that *MixColumns* as a diffusion layer of the AES algorithm has good protection of the information.

Because the *MixColumns* information from round 9 have an accuracy of less than 0.39% on the test set when used in the single-input models. This segment of information will be discarded and will not be involved in the training of the multi-input models. The conclusion that the multi-input models have higher accuracy in the presence of multiple leaks in the power traces was also demonstrated in the last round of the AES algorithm.

E. Discussion

The results of the two sets of experiments show that the useing of multiple-input models can effectively improve the network model's accuracy when the power traces exhibit as multiple leaks. Which colud raise the following problems:

- Why are four different bytes used as the target bytes for the experiment;
- Why are two different connections used for the inputs of the multi-input models;
- Why are the first and last rounds of the AES algorithm compared for the experiment.

The AES algorithm operates on a 4×4 byte matrix, and the four bytes selected for the experiments are from different rows of this matrix. Experimental results show that the 5th byte is not shifted and it is not possible to use this information to improve the accuracy of the multi-input model. The other bytes where *ShiftRows* operations are present had an accuracy greater than 0.39% when trained by the network model. When using multi-input models and adding the *ShiftRows* information to the models, the models accuracy improve in all four bytes except for the 5th byte.

In the paper, we use two different connections when the inputs to the network model are fused, and experiments demonstrate better results when the data are connected in parallel. The results of single-input model indicates that the accuracy of the different inputs in the model vary considerably. When serial connections are used, some information that has less impact on the accuracy of the network model and even some information that interferes with the classification of the

TABLE IV
RESULTS FOR BYTE 5, BYTE 6, BYTE 7, AND BYTE 8 IN THE LAST ROUND OF THE AES ALGORITHM USING MULTI-INPUT MODELS.

	Byte 5	Byte 6	Byte 7	Byte 8
POI intervals (AddRoundKey (J), ShiftRows (I), SubBytes (H), AddRoundKey (G), MixColumns (F))	[3170:3230] [2890:2950] [2090:2150] [1510:1570] [520:580]	[3200:3260] [2800:2860] [2305:2365] [1695:1755] [855:915]	[3230:3290] [2835:2895] [2520:2580] [1875:1935] [1190:1250]	[3260:3320] [2880:2940] [2580:2640] [1450:1510] [410:470]
1 - input models (Accuracy)	14.73%	13.71%	23.22%	15.41%
	0.39%	2.58%	2.48%	2.38%
	45.32%	42.04%	48.59%	45.51%
	14.27%	22.45%	10.58%	18.64%
	0.37%	0.37%	0.33%	0.38%
2 - input models (axis = 1, axis = 2)	80.41%	88.70%	74.80%	71.49%
	82.20%	88.97%	76.16%	83.10%
3 - input models (axis = 1, axis = 2)	94.71%	91.63%	92.90%	88.83%
	95.17%	96.49%	94.75%	96.49%
4 - input models (axis = 1, axis = 2)	87.96%	94.02%	93.77%	89.47%
	94.27%	97.38%	95.10%	96.57%

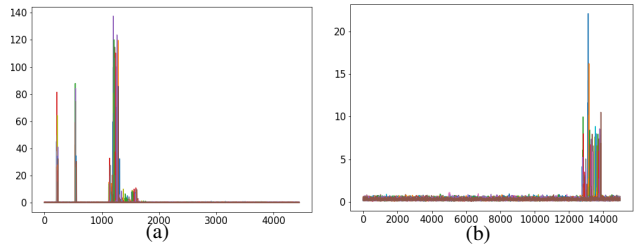

Fig. 6. (a) POI of the Screaming Channel dataset; (b) POI of the AES_GPU dataset.

TABLE V
COMPARISON OF EXISTING RESULTS AND OUR WORK.

Database	Methods	Accuracy	Mean rank
Screaming Channel	[18]	-	14
	Single input	3.65%	15
	Multi-input	6.25%	8
AES_GPU	[26]	-	50
	Single input	1.74%	37
	Multi-input	3.63%	13

network model is also used for training. This is the reason why the number of inputs increases but the accuracy of the network model decreases in a multi-input model. In the case of parallel connectivity, the higher dimensional data eliminates this detrimental effect, and this approach has been applied to image classification [30].

In the multi-input model experiments, the last round of the AES algorithm used for training has one more *AddRoundKey* information than the first round of the leakage interval, but one less *MixColumns* information. It is easy to see that *AddRoundKey* has a much greater impact on the multi-input model than *MixColumns*. So the multi-input model has better results in the last round.

F. Results of Multi-input Models on Other Datasets

Multiple leakages are also presented in the Screaming Channel dataset and the AES_GPU dataset. The POI for both datasets are shown by Fig. 6.

In Table V, we compare our work with other existing work. There is none of the studies using these two datasets recover the keys via the network model. We show the highest accuracy and corresponding mean rank [5] for key recovery using the single-input model while performing the same experiments using the multi-input model.

As demonstrated in two well-known datasets, multiple leaks are common in hardware implementations of the AES algorithm.Using multi-input models based on this situation can improve the accuracy of the model and the efficiency of recovering keys from the devices.

IV. CONCLUSION AND FUTURE WORK

In this paper, we propose a multi-input deep-learning model for side-channel attacks, which is dedicated for the case where multiple leakage intervals exist in traces. By utilizing these leakages as separate inputs instead of using the entire trace for profiling, the trained model can focus more on these leakages. Two well-known publicly available datasets and traces captured from a STM32 implementation of AES are used in our experiments. We show that the proposed multi-input model achieves a 2-fold improvement over the previous single-input attacks. Besides, we further compare different fusion layers for connecting leakage intervals. The result shows that concatenating leakage intervals in parallel outperforms other approaches.

Future work includes testing the proposed multi-input model on implementations of other cryptographic algorithms and mounting similar attacks on devices supporting AES with other countermeasures. Besides, we plan to further investigate the multi-leakage phenomena by training new models on other attack points. Certainly, the most important future work should

be designing countermeasures to against deep-learning based side-channel attacks.

ACKNOWLEDGMENT

This research was supported by National Natural Science Foundation of China (No.61973109).

REFERENCES

[1] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*. Springer, 1996, pp. 104–113.

[2] A. Shamir and E. Tromer, "Acoustic cryptanalysis," 2004.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397.

[4] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2001, pp. 251–261.

[5] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Study of deep learning techniques for side-channel analysis and introduction to ascad database," *ANSSI, France & CEA, LETI, MINATEC Campus, France. Online verfügbar unter https://eprint. iacr. org/2018/053. pdf, zuletzt geprüft am*, vol. 22, p. 2018, 2018.

[6] A. Schlösser, D. Nedospasov, J. Krämer, S. Orlic, and J.-P. Seifert, "Simple photonic emission analysis of aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 41–57.

[7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[8] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2016, pp. 3–26.

[9] H. Wang, M. Brisfors, S. Forsmark, and E. Dubrova, "How diversity affects deep-learning side-channel attacks," in *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*. IEEE, 2019, pp. 1–7.

[10] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen, "X-deepsca: Cross-device deep learning side channel attack," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[11] H. Wang, S. Forsmark, M. Brisfors, and E. Dubrova, "Multi-source training deep-learning side-channel attacks," in *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2020, pp. 58–63.

[12] T. Kubota, K. Yoshida, M. Shiozaki, and T. Fujino, "Deep learning side-channel attack against hardware implementations of aes," *Microprocessors and Microsystems*, p. 103383, 2020.

[13] H. Wang and E. Dubrova, "Tandem deep learning side-channel attack against fpga implementation of aes." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 373, 2020.

[14] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic, "Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 148–179, 2019.

[20] J. Daemen and V. Rijmen, "Reijndael: The advanced encryption standard." *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 26, no. 3, pp. 137–139, 2001.

[15] L. Masure, C. Dumas, and E. Prouff, "A comprehensive study of deep learning for side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 348–375, 2020.

[16] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 45–68.

[17] D. Moonen, "Little or large?: The effects of network size on ai explainability in side-channel attacks," 2020.

[18] G. Camurati, A. Francillon, and F.-X. Standaert, "Understanding screaming channels: From a detailed analysis to improved attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 358–401, 2020.

[19] Y. Gao, H. Zhang, W. Cheng, Y. Zhou, and Y. Cao, "Electro-magnetic analysis of gpu-based aes implementation," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[21] L. Zhang, X. Xing, J. Fan, Z. Wang, and S. Wang, "Multi-label deep learning based side channel attack," in *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2019, pp. 1–6.

[22] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[23] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni, "The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 1–29, 2019.

[24] D. Feng, C. Haase-Schuetz, L. Rosenbaum, H. Hertlein, C. Glaeser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[25] J. Xu and H. M. Heys, "Using deep learning to combine static and dynamic power analyses of cryptographic circuits," *International Journal of Circuit Theory and Applications*, vol. 47, no. 6, pp. 971–990, 2019.

[26] G. Yang, H. Li, J. Ming, and Y. Zhou, "Cdae: Towards empowering denoising in side-channel analysis," in *International Conference on Information and Communications Security*. Springer, 2019, pp. 269–286.

[27] D. H. Johnson, "Signal-to-noise ratio," *Scholarpedia*, vol. 1, no. 12, p. 2088, 2006.

[28] C. O'Flynn and Z. D. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 243–260.

[29] F. Durvaux and F.-X. Standaert, "From improved leakage detection to the detection of points of interests in leakage traces," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 240–262.

[30] X. Hu and Y. Zhou, "Cross-modality person reid with maximum intra-class triplet loss," in *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*. Springer, 2020, pp. 557–568.