Side-channel attacks based on power trace decomposition

Fanliang Hu[1], Huanyu Wang[2], Junnian Wang[1]*
[1]School of Physics and Electronic Science, Hunan University of Science and Technology, Xiangtan, China;
[2]School of EECS, KTH Royal Institute of Technology, Stockholm, Sweden;
Email: {fanliang, jnwang}@mail.hnust.edu.cn; huanyu@kth.se

***Abstract*—Side Channel Attacks (SCAs), an attack that exploits the physical information generated when an encryption algorithm is executed on a device to recover the key, have become one of the key threats to the security of encrypted devices. Recently, with the development of deep learning, deep learning techniques have been applied to side channel attacks with good results on publicly available dataset experiences. In this paper, we propose a power tracking decomposition method that divides the original power tracking into two parts, where the data-influenced part is defined as data power tracking and the other part is defined as device constant power tracking, and use the data power tracking for training the network model, which has more obvious advantages than using the original power tracking for training the network model. To verify the effectiveness of the approach, we evaluated the ATxmega128D4 microcontroller by capturing the power traces generated when implementing AES-128. Experimental results show that network models trained using data power traces outperform network models trained using raw power traces in terms of classification accuracy, training time, cross-subkey recovery key and cross-device recovery key.**

***Index Terms*—Power analysis, Side-channel attacks, Power trace decomposition, Deep learning, AES**

I. Introduction

Side Channel Attacks (SCAs) have been a problem that the hardware security community has had to confront since they were first introduced over 20 years ago [1]. An attacker recovers sensitive data from a target cryptographic device by analysing information such as algorithm runtime, power consumption and electromagnetic radiation that is unintentionally leaked during the execution of the cryptographic algorithm. Common cryptographic devices are small embedded devices such as smart cards, FPGAs, ASICs, IoTs and microcontrollers [2]–[5]. In addition some recent research has targeted remote devices such as Screaming Channels and Bluetooth [6], [7]. In recent years, with the advancement of deep learning techniques [8], Deep Learning Side-Channel Attacks (DLSCAs) have been shown to be more effective and easier to implement than traditional side-channel attacks. Since deep neural networks are good at extracting features from data, they can help attackers find correlations between physical leaks and the internal state of the processing algorithm.Maghrebi et al. first applied deep learning techniques to SCAs in 2016, when they analysed MultiLayer Perceptron (MLP) and Convolutional Neural Network (CNN). Convolutional Neural Network (CNN) on SCAs [9]. Subsequently, Cagli et al. proposed an end-to-end CNN-based approach for the analysis of SCAs in 2017, which does not require pre-processing of power traces nor precise selection of Points of Interest (POI) [10], in addition to improving the performance of convolutional neural networks using data augmentation [11]. In 2019, the Huanyu et al.'s study illustrated how the diversity of target chips affects the efficiency of side channel attacks [12]. 2020, Moonen et al, investigated the effect of the depth of the network model on the efficiency of side channel attacks by introducing a heat map [13]. These research works show that DLSCAs have good performance in terms of key attack efficiency and accuracy.

However, most of the current studies on DLSCAs use the original power traces and key-dependent sensitive values to build deep networks, which degrade the quality of side channel analysis because the original power traces contain power information unrelated to the sensitive values. In this paper, based on the fact that power traces consist of multiple power information, we propose a method to split the power traces by defining the data-dependent part (e.g., plaintext, ciphertext, key) as data power traces and the other part as constant power traces. By using the data power traces to train the network model, the efficiency of the side-channel attack can be improved. We conduct experiments using power consumption traces captured from an ATXmega28D4 microcontroller running AES-128. The experimental results show that (1) when the network model achieves the same classification accuracy, the model trained using the data power traces is 3.64 times more efficient compared to the model trained using the original power traces; (2) in the experiments on cross-byte key recovery, the model trained using the data power traces is 14.62 times more efficient compared to the model trained using the original power traces; (3) in the experiments on cross-device recovery (3) in the experiment of key recovery across devices, the model trained with original power traces cannot complete key recovery, while the model trained with data power traces only needs 511 power traces to recover all keys.

The chapters in this paper are organized as follows: in Section 2, we introduce the background knowledge of deep learning-based side channel attacks. In Section 3, we provide an analysis of the principles of power consumption trace decomposition and an explanation of why the network can be trained using a component of the power consumption trace. In Section 4, we describe the collection of data, the construction of the experimental environment, data pre-processing, network

models and evaluation criteria. In section 5, we describe how the models are trained and how they compare in terms of training time, classification accuracy and mean_rank. Finally, in Section 6 we summarise all the work in this paper and present the outlook for future work.

II. Fundamentals of Side Channel Analysis

A. Advanced Encryption Standard

The National Institute of Standards and Technology (NIST) published the Advanced Encryption Standard (AES) in 2001 [14]. AES can be divided into AES-128, AES- The side channel attack in this paper is based on the AES-128 cipher implementation, which is executed on a 4 $\times$ 4 byte matrix. The initial value matrix is obtained by a plaintext (p) of 128 bits in length (8 bits to a byte, a plaintext containing 16 bits distributed in a top-to-bottom, left-to-right order in a 4$\times$4 matrix) and a key (k) of 128 bits (the same distribution as the plaintext) in a corresponding byte anomaly operation. The encryption process requires a total of 10 iterations, each of which consists of four basic steps: byte substitution, row displacement, column obfuscation and round key addition, of which the last round consists only of byte substitution, row displacement and round key addition without column obfuscation.

In AES, a non-linear substitution is performed on the elements of the initial value matrix using an S-Box (S-Box, which is a matrix of 16 x 16 bytes containing one permutation of the 256 numbers that can be represented by 8 bits of data), an operation called byte substitution. Byte substitution is the only non-linear transformation in the AES algorithm and is key to the security of the algorithm. We define v to be the output of the S-box, the intermediate value of the AES encryption algorithm, with subscript i denoting the $1st$ to 16th byte, v_i denoting the output of the S-box at the $i-th$ byte, and similarly p_i, k_i denoting the input to the S-box at the $i-th$ byte.

$$v_i = SBox(p_i \oplus k_i), i = 1, 2, ..., 16 \tag{1}$$

where p denotes the plaintext, k denotes the key, $p \oplus k$ is also the initial value matrix mentioned earlier, and $\oplus$ denotes the iso-or operation. Since v depends on k, the output v of the SBox is sensitive data.

B. Side channel analysis

The aim of a side-channel attack is to recover the key using a physical leak generated by an encryption device implementing an encryption algorithm. Typically, a partitioning strategy is used to recover different sub-secret keys of the key separately. For example, an attacker recovers one 8-bit subkey byte at a time iteratively to recover the entire 128-bit subkey in an AES-128 cryptographic implementation.

One of the most powerful of the side-channel attacks is the modeling class side-channel attack, which assumes that an attacker can learn the leakage distribution offline in a supervised manner using an open copy of the target device in advance and attack the target device online using the learned model. In the analysis phase, the attacker has a device that knows the key and obtains N power traces $\mathbf{X}_{profiling} = xi, (i = 1, 2, ..., N)$. Each power consumption trace xi corresponds to a known plaintext (p), a key (k) and an energy leakage model θ (e.g., the Hamming weight model HW) with a sensitivity value function $v_i = \theta(p_i, k)$. In our experiments, we first collect the power consumption traces and once the collection of power traces is completed, the attacker will configure the appropriate model and compute the probability estimate: $Pr[x|V = v]$, the data set used for the analysis phase is denoted as $x_i, v_i, (i = 1, 2, ..., N)$.

In the attack phase, the attacker collects M power consumption traces, denoted as the set $\mathbf{X}_{attack} = xi, (i = 1, 2, ..., M)$. $\mathbf{X}_{profiling}$ and $\mathbf{X}_{attack}$ are independent of each other, and each power trace x_i corresponds to a fixed unknown key k^*. The attacker calculates the probability of the intermediate value corresponding to the guessed key k for each power consumption trace according to Bayes' theorem.

$$P_r[v_i|x = x_i] = \frac{P_r[x = x_i|v_i] \cdot P_r[v_i]}{P_r[x = x_i]} \tag{2}$$

The likelihood function value d_k corresponding to each guess key k is then calculated using the maximum likelihood probability criterion.

$$d_k = \prod_{i=1}^{M} P_r[v_i = \theta(p_i, k)|x = x_i] \tag{3}$$

Calculate the maximum likelihood function estimate $\widetilde{k}$ for k.

$$\widetilde{k} = \underset{k}{argmax}(d_k) \tag{4}$$

As the number of attack bars M increases, eventually $\widetilde{k}$ equals the correct key k^*.

C. Deep learning based on side channel analysis

What is the same in DLSCAs as in modelling class side channel attacks is that we consider that an attacker has access to the same pair of devices: an encryption device that uses a fixed and unknown key $k \in *K, (K = 0, 1, 2, ..., 255)$ to run the encryption device, and an analysis device that knows and controls the keys and inputs. A partitioning strategy is usually used, e.g. this attack recovers the key byte k_3. deep learning training is used as the analysis method, not multivariate Gaussian analysis as in the template attack. the DLSCAs are also divided into two phases.

Analysis phase: In the analysis phase, there exists a set of N power consumption traces $\mathbf{T}k = \mathbf{T}_i, k|i = 1, 2, ..., N$, and from each key the power consumption traces are collected to form the training set $\mathbf{X}$, so $\mathbf{X}$ can be defined as:

$$X = \bigcup_{k=0}^{255} T_k \tag{5}$$

The labels used for training are defined as **Y**. In order to use the neural network to explore the relationship between power consumption traces and labels, a neural network needs to be built that can be trained to classify power consumption traces according to their corresponding labels.

Attack phase: To recover the key k^* using M power traces collected from the target device, each trace $T_i(1 \leq i \leq M)$ is first evaluated using the trained network to obtain a score vector $y_i = N(\mathbf{T}_i \in R^{|K|})$. The highest scoring value is then selected as:

$$k = argmax(\sum_{i=1}^{M} y_i)_j \quad (6)$$
$$j \in K$$

If $k = k^*$ then the attack is successful.

In most of the current DLSCAs studies, the power traces used for analysis are the original power traces. In this paper, the original power traces are decomposed for use in DLSCAs with respect to the composition of the power traces, and the principles of the power trace decomposition are described below.

III. Power trace decomposition

A. Composition of power traces

The power consumption based side channel attack exploits the fact that different inputs and different operations generate different power consumptions during the execution of encryption by the cryptographic device. Thus, the data-dependent component of the power trace is denoted as P_{data} and the operation-dependent component is denoted as P_{op}. In addition, the power trace is dependent on two other factors, the electronic noise $P_{el.noise}$, which is present in every measurement, and it is due to the effect of the electronic noise that the power traces obtained are different even for the same input. In addition, due to the presence of leakage currents and transistor conversion activity unrelated to P_{data} and P_{op}, a constant power consumption is generated, noted as P_{const}. A power trace is the sum of these components, so that the power trace can be expressed by the following equation.

$$P_{total} = P_{data} + P_{op} + P_{el.noise} + P_{const} \quad (7)$$

In power consumption based SCAs, P_{data}, P_{op} and $P_{el.noise}$ are the key components, while Pconst is a component that is not relevant to the power analysis as it does not contain any information that can be exploited by an attacker. An attacker can usually only obtain the key through P_{data} and P_{op}. And as the component $P_{el.noise}$ increases as a percentage of the power consumption trace, the difficulty of the attack increases.

B. Power trace composition analysis

When dealing with classification problems using traditional deep learning algorithms, the data is divided according to its own characteristics, and the basic principle is to find a suitable classification curve or surface to solve the problem. However, from the composition of the power traces, based on this principle DLSCAs use some non-essential information related to the recovery key when training the network model using the original power traces. In the following we will analyse the components of the power consumption trace.

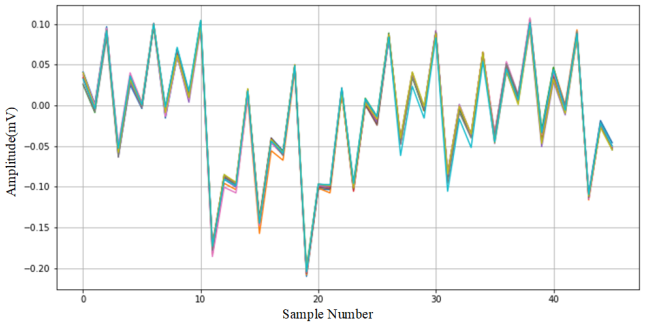

Fig. 1. The shape of 10 power consumption traces when processing the same data.

First we analyse the component $P_{el.noise}$. At time t, the power consumption trace of the sensitive variable v consists of a constant power consumption trace and noise, which can be defined as:

$$P_{total} = f_v(t) + P_{el.noise} \quad (8)$$

where v is a sensitive variable, $f_v(t)$ is a time-dependent function mapping from v to the actual power consumption leakage, and $P_{el.noise}$ is a time-dependent, v-independent and normally distributed noise. In order to have a better understanding of the noise in the power traces, we have collected 10 power traces that are generated when constant data is executed and have the same operation, as shown in Fig.1. It is clear that the 10 traces are very similar, but there are also some differences between these traces due to the presence of electronic noise. The effect of electronic noise on side-channel attacks is negative, and much research has been done on noise reduction [15], [16], which is not described in detail here.

The next step is to analyse the component P_{data}, which is not only dependent on the data being processed, but is also related to the encryption device, which is an 8-bit ATMEL microcontroller, a low-power 8-bit CMOS microcontroller with an AVR RISC architecture. The total energy consumption of the CMOS circuit is equal to the sum of the individual logic components of the CMOS circuit. As shown in Fig.2, the logic elements in the COMS circuit are powered by a constant voltage V_{DD} and process the input signal. When the total instantaneous current is denoted $i_{DD}(t)$, then the total instantaneous energy consumption is denoted $P_{cit}(t)$ and the average consumption over time T can be calculated using the following equation.

$$P_{cir} = \frac{1}{T}\int_0^T p_{cir}(t)dt = \frac{V_{DD}}{T}\int_0^T i_{DD}(t)dt \quad (9)$$

CMOS components are all based on complementary pull-up and pull-down networks, where, for example, a CMOS inverter contains two transistors $P1$ and $N1$, as shown in

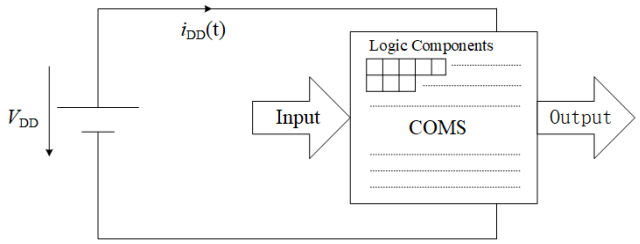

Fig. 2. Energy consumption of CMOS circuits.

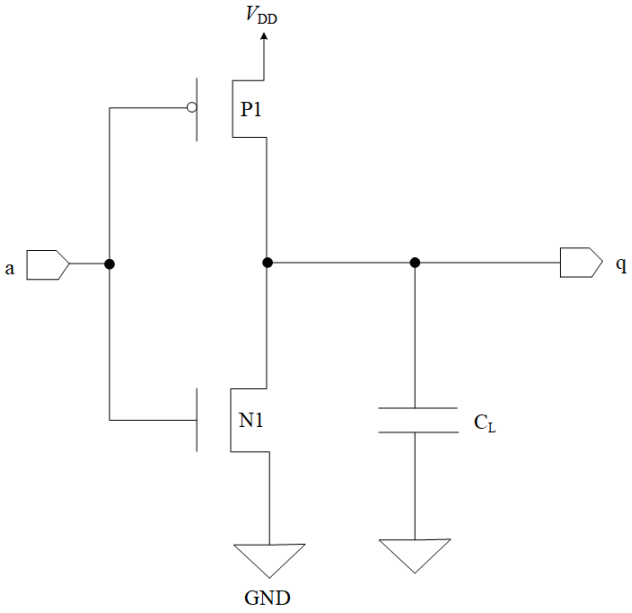

Fig. 3. Equivalent Capacitance of CMOS Contrarians.

Fig.3. The energy in an inverter consists of static energy P_{stat} and dynamic energy P_{dyn}. When there is no energy conversion in the element, the energy that the element has is called static energy, and the energy generated when there is an internal signal or an output signal is called dynamic energy. In the following, we focus on the analysis of dynamic energy, as shown in Table.I. In the cases $0 \rightarrow 0$ and $1 \rightarrow 1$, the logic element only generates static energy consumption, while the other two cases generate dynamic energy consumption. It is worth noting that in a typical CMOS circuit, dynamic energy consumption is always the dominant factor in energy consumption, which is dependent on the data being processed.

Simulation models for COMS energy consumption usually use Hamming weight or Hamming distance. Due to the simplicity of the principle and implementation, these two energy models are widely used in energy simulations and, although the simulation of energy is relatively crude, they are a way to calculate energy consumption quickly. A formal representation of the two energy models is given below. The Hamming distance between the values m_0 and m_1 is equal to m_0 isomorphic to m_1 and the Hamming weight is equal to the number of bits with a logical value of 1. Thus the Hamming weight represents the number of dissimilar bits in m_0 and m_1. Although no general statement can be made about the dependence of P_{data} on the data being processed, we can approximate the P_{data} component by a normal distribution when the data processed by the cryptographic device is uniformly distributed.

Since only $P_{el.noise}$ and P_{data} are used as experimental objects in this paper, the components P_{op} and P_{const} will not be described too much.

TABLE I
CMOS OUTPUT AND ENERGY CONSUMPTION COMPARISON TABLE.

Status	Energy consumption	Type of energy consumption
$0 \rightarrow 0$	P_{00}	Static
$0 \rightarrow 1$	P_{01}	Static+
$1 \rightarrow 0$	P_{10}	Static+Dynamic
$1 \rightarrow 1$	P_{11}	Static

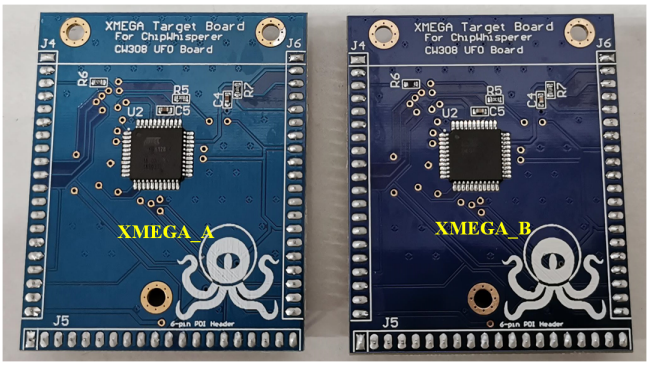

Fig. 4. Target boards XMEGA_A and XMEGA_B.

IV. Experimental Environment Configuration and Data Acquisition

A. Experimental environment configuration

All experiments in this paper were completed in the same experimental environment configuration. A total of two target boards were used for data acquisition, XMEGA_A and XMEGA_B as shown in Fig.4. They both contain the same 8-bit ATMEL microcontroller ATxmega128D4 to complete the encryption operation, and use the Chipwhisperer platform to complete the collection of all the power traces generated during the encryption process, with the encryption mode being the Electric Code Book (ECB) mode of AES-128. Fig.5 shows the experimental equipment and the data acquisition process. In this experiment, all our models were built and trained under the deep learning frameworks Keras-gpu 2.3.1 and tensorflow-gpu 2.1.0, with the main hardware configuration of the computer being an Intel(R) Core(TM) i7-9750H CPU @2.60GHz and an NVIDIA GeForce GTX 2060 6GB GPU to perform all numerical computations and model training in the experiments.

B. Data collection

A total of 70,000 power consumption traces were collected as the experimental dataset. Of these, 50,000 traces were

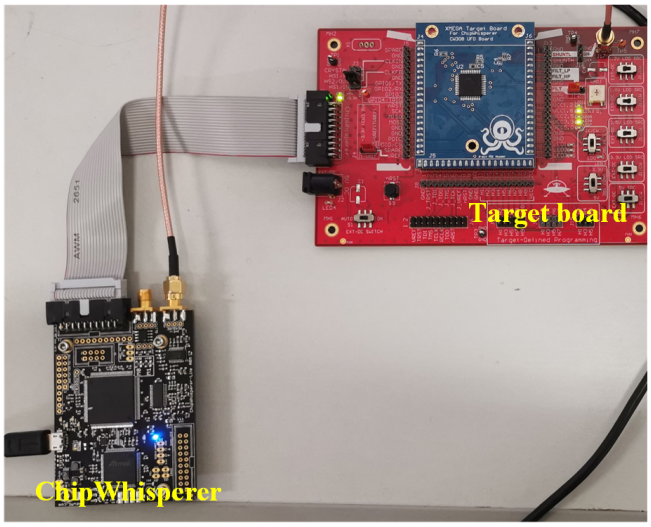

Fig. 5. Capturing power consumption traces with ChipWhisperer devices.

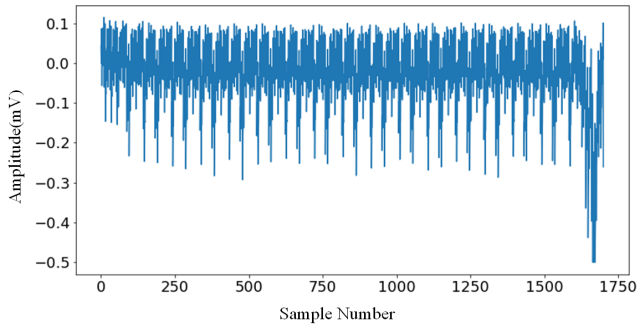

Fig. 6. Power trace waveform display.

generated by the cryptographic device XMEGA_A using random plaintext and random key, and the other 20,000 traces were generated by XMEGA_A and XMEGA_B using random plaintext and fixed key, respectively, as the test set. After alignment of the traces, each trace contains 1,700 samples. Fig.6 shows the example traces.

V. DLSCAs BASED ON POWER TRACE DECOMPOSITION

A. Choice of point of attack and interval of interest

When implementing a side-channel attack, researchers first need to model the energy consumption of the cryptographic device when running the encryption algorithm. There are three energy models, the Identity (ID), Hamming Distance (HD) and Hamming Weight (HW) models, and for different energy consumption models, the labels of the power consumption data vary. In this paper we choose the ID model, which corresponds to a total of $2^8 = 256$ types of labels.

After determining the energy consumption model, the location of the target attack point needs to be determined. In this paper, the encryption algorithm we target is AES-128, and the target attack point chosen is the output location of the byte substitution in the first round of encryption operations of that encryption algorithm (i.e. the output of the SBox). The ultimate goal of our side-channel attack experiment is to recover the first byte of the initial key block, denoted by k_0. We set the labels of each model trained in the experiment to the output state of the S-box byte substitution in the first round of encryption operations, denoted as:

$$state_0 = SBox(p_0 \oplus k_0) \tag{10}$$

In Equation 10, $\oplus$ indicates the byte-by-bit operation, p_0 and k_0 indicate the first byte of the plaintext and the first byte of the initial key respectively, and $state_0$ indicates the state after the output of the SBox, i.e. the tag. The main reason for setting the tag in this way is that when the target encryption chip runs the encryption algorithm, it first needs to call SBox from the internal registers to perform the byte substitution operation in the encryption algorithm, and then load the intermediate state after the operation onto the data bus, and the capacitive load of the data bus is generally very large, which has a great impact on the energy consumption of the encryption chip.

Since the target attack point of the experiment is the output location after the SBox byte replacement in the first round of encryption operation, and this location contains a total of 16 bytes of SBox output information, we need to find the leaked information interval (interval of interest) of the target byte (the first byte of the first round of SBox output state).

Common methods for finding interest intervals for target bytes include SNR, CPA, t-test and ρ-test [17]. In this paper, the method of averaging data power traces is used to find the intervals of interest, with the following main steps.

1) Collect raw power consumption data using a power consumption acquisition device and align all data.
2) Obtain the data power traces by using Algorithm.V-B.
3) Average the power consumption traces with the same intermediate values.
4) Reduce the dimensionality of all the raw power consumption data and retain only the power consumption data in the interval of interest.

The power consumption traces were analysed in the experiments using the method of averaging data power consumption traces. Since the data power traces contain only $P_{el.noise}$ and P_{data}, and since $P_{el.noise}$ obeys a normal distribution, the average data power traces with the same sensitive values are averaged out, the average data power traces contain only P_{data}. The area that is only relevant to the sensitive values is intercepted and is the interval of interest, which is shown in Fig.7, where the range marked by the dashed line is the target in the experiments of this paper The interval of interest for the byte is [56: 152]. As shown in Fig.8, the average data power consumption trace curves with sensitivity value of 0 and sensitivity value of 255 are approximately symmetrical. From the theoretical point of view, the two curves should be completely symmetrical, but since the actual encryption process is not simply a superposition of several power consumption trace components to form the original power consumption trace, the two curves are only approximately symmetrical.

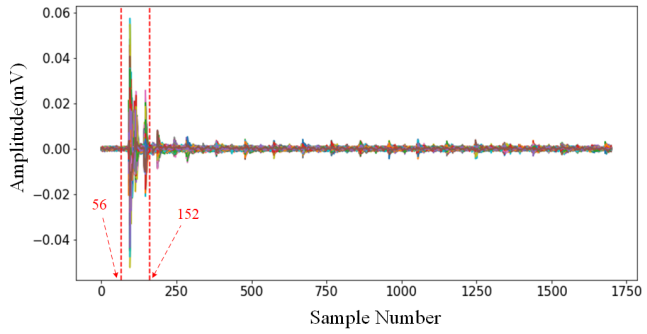

Fig. 7. Average power consumption trail points of interest.

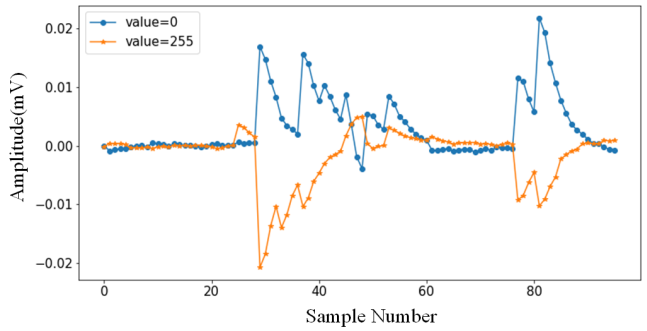

Fig. 8. Average data power traces with intermediate values of 0 and 255.

B. Power trace decomposition method

In order to reduce the information irrelevant to the recovery of sensitive values, the P_{data} component based on the processed data part should be extracted from the original power consumption trace so that the neural network can better learn the data features associated with the sensitive information. There is noise information in each power consumption trace that obeys a Gaussian distribution, and our current study can only extract a mixture of P_{data} and $P_{el.noise}$ power consumption traces, which is also considered by us as a data power consumption trace. Obtaining the data power traces is straightforward and simple, and first requires averaging the power traces in the sample. Once the average of the power traces is obtained, we can generate the data power traces by the method in Algorithm.V-B.

Average power trace: the average of all power traces. The average power trace is obtained by averaging the power traces of different intermediate values, and a trace containing only P_{op} and P_{const}, because P_{data} and $P_{el.noise}$ obey normal distribution, so after a large number of power traces are averaged, the average power trace then does not contain P_{data} and $P_{el.noise}$.

Data power trace: each power trace minus the average power trace. Theoretically, P_{op} and P_{const} are fixed, so the data power trace contains only P_{data} and $P_{el.noise}$.

In the following experiments, we will demonstrate through comparative trials that network models trained on data power traces are more efficient in side-channel attacks.

Algorithm 1 Method for obtaining data power traces

Input: original power trace $\mathbf{X}_{original} = x_i, (i = 1, 2, ..., N)$

Output: Data power traces $\mathbf{X}_{data} = \{x_i - \frac{1}{N}\sum_{j=1}^{N} x_j\}_{j=1}^{N}$

1: Initialize the data power trace dataset X_{data} to the empty set

2: Calculation of the average trace of the original power consumption trace

3: For all $x_i \in \mathbf{X}_{original}$ do

4: $x_{data_i} = x_i - \overline{x}$

5: Append(x_{data_i}) into $\mathbf{X}_{data}$

6: end for x_{data_i}

7: return $\mathbf{X}_{data}$

TABLE II
NETWORK MODEL STRUCTURE AND PARAMETERS.

Layer Type	Output Shape	Parameter #
Input	(None, 96, 1)	0
Conv1D 1	(None, 92, 64)	384
AveragePooling 1	(None, 46, 64)	0
Conv1D 2	(None, 40, 64)	28736
AveragePooling 2	(None, 20, 64)	0
Flatten	(None, 1280)	0
Dense 1	(None, 256)	327936
Dense 2	(None, 256)	65792
Output(Dense 3)	(None, 256)	65792
Total Parameter: 488,640		

C. Network model structure

In this experiment, the network structure is used in all experiments and the hyperparameters are the same. The network has 8 layers, two of which are convolutional layers, both with 64 convolutional kernels, a convolutional step size of 5 for the first layer and 7 for the second layer, two pooling layers with average pooling, a Flatten layer and three fully connected layers with 256 neurons. The output layer uses the $Softmax$ activation function and the rest of the network layers are set to $Relu$. 488,640 parameters are included in the final convolutional model, as shown in Table.II.

D. Evaluation metrics

(1) Model accuracy Model accuracy refers to the probability that the model achieves the correct classification result on the test set [18], [19]. An increase in model accuracy indicates that the back propagation algorithm's optimisation of the weights and bias term parameters gradually converges closer to the optimal parameter values and the network model gradually

converges to the optimal model. Model accuracy is generally defined as:

$$acc(X_{attack}) = \frac{|\{x_i \in X_{attack}\}|}{|X_{attack}|} \tag{11}$$

In the formula X_{attack} denotes the test dataset, x_i denotes the i_{th} power trace in the dataset, and $x_i \in X_{attack}$ denotes the set of all power curves when the guess key equals the correct key. Thus the accuracy of the model is also understood as the ratio of the number of power consumption curves when the guess key is equal to the correct key to the number of power consumption curves in all the verification sets.

(2) Mean_rank To evaluate the performance of a model in SCAs, a metric is needed to represent the number of power traces used to acquire keys using the model. This metric typically uses mean_rank (average rank) [20]. The mean rank indicates how many power traces are needed on average by the attacker to recover the key after performing the side channel analysis. For example, there are T power consumption traces in the attack phase and the attack outputs a guess vector $g = [g_1, g_2, ..., g_{[K]}]$, where g is decreasing in probability. The mean_rank is then the average ranked position of the correct key k^* in g over multiple experiments.

(3) Training time Training time is one of the most important criteria for evaluating the merits of a deep learning model. When two models are compared and the same accuracy is achieved, if the training time of a model is shorter, it indicates that the model performs better, the model converges faster and is easier to train for deep learning.

VI. Experimental results and analysis

A. Model training

In the experiments, the optimiser of the model was set to RMSprop, the corresponding learning rate size was set to 0.0005, the number of iterations was set to 500, and the random seed for the initialisation of the network model weights was set to 122.

(1) Model training using raw power traces The final results of the classification experiments of the model trained with the original power consumption traces are shown in Fig.9. From the experimental results, it can be seen that the model trained using the original power consumption trace has an accuracy of 97.70% on the training set and 95.68% on the validation set at epochs equal to 175. As the number of iterations increased, the accuracy began to converge slowly and eventually reached the optimal model at the 463rd epochs of training. Optimal model: acc=99.73%, val_acc=99.67%.

(2) Model training using data power traces The final results of the classification experiments of the model trained with data power traces are shown in Fig.10. From the experimental results, it can be seen that the model trained using the data power consumption trace has an accuracy of 99.81% on the training set and 97.88% on the validation set at epochs equal to 40. As the number of iterations increases, the accuracy starts to converge slowly and eventually reaches the optimal model at the 254th epochs of training. Optimal model: acc=99.99%, val_acc=99.92%.

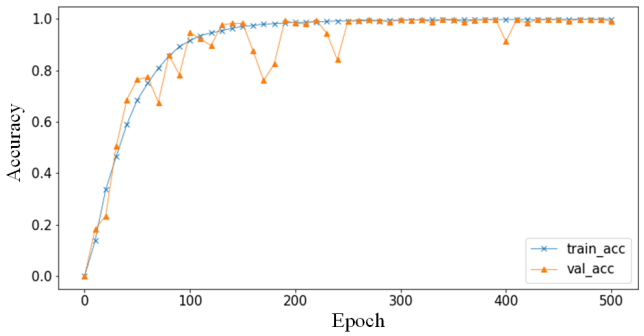

Fig. 9. Presentation of the results of the model trained with the original power traces (Model_raw).

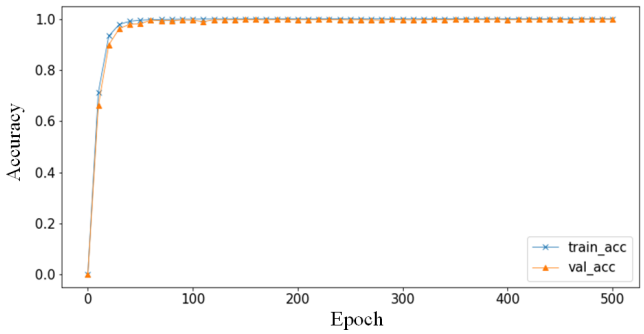

Fig. 10. Presentation of the results of the model trained with the data power traces (Model_data).

B. Comparative experiments of the two models

In order to investigate the performance of models trained on data power traces in side-channel attacks, three further sets of comparison experiments are done in this paper.

(1) Model training time comparison As can be seen in Fig.9 and Fig.10, the model trained based on data power traces has the advantage of fast convergence and robustness. To quantify the efficiency gains of the model on side channel attacks, we compared the two models in terms of model training time. The time required to train each model when both models achieved 99.00% accuracy on the validation set is represented in Fig.11. The model trained using the power consumption trace dataset takes only 47 seconds to achieve 99.00% accuracy on the validation set, however the model trained using the original power consumption trace takes 171 seconds. When both models reached 99.00% accuracy on the validation set, the model using the data power traces was 3.64 times more efficient than the model trained using the original power traces. The model trained using the data power traces is denoted as Model_data and the model trained using the raw power traces is denoted as Model_raw.

(2) Two models of cross-byte recovery key comparison experiments Previous studies have used a partitioned approach in recovering the key, where it is necessary to recover $k_i (i =$

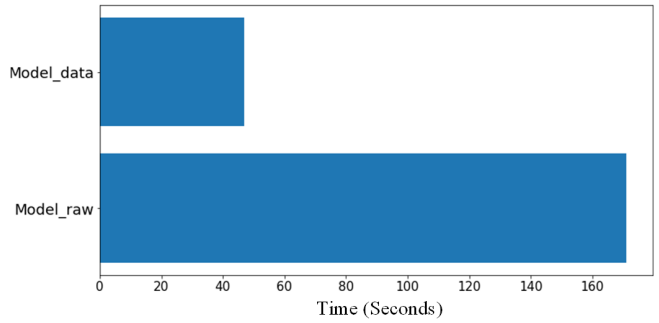

Fig. 11. Comparison of the time taken by the two models when the validation set is 99.00% accurate.

$1, 2, ..., 16$), then use the k_i corresponding interest interval and the corresponding sensitive value to build a network to recover the byte key, this byte is called the target byte by the attacker. In fact, it is not difficult to find that the shape of the power trace of the target byte's interest interval is similar to the shape of the power trace of the interest interval of other bytes, as shown in Fig.12. So the network model trained using the target byte can also classify the power traces of other non-target bytes.

Next, we perform comparative experiments between the two models using the accuracy of the models across bytes and mean_rank. Fig.13 shows the results for the accuracy of the two models on a test set of 16 bytes. Fig.14 shows the number of power traces (i.e. mean_rank) required to recover the full key for both models.

(3) Cross-device recovery key comparison experiment between two models Based on the fact that both XMEGA_A and XMEGA_B use 8-bit ATMEL microcontrollers, the power traces obtained when running AES-128 on both target boards are similar. Based on this fact, we conducted comparative experiments using both models on the power traces taken when running AES-128 on the XMEGA_B target board. Figure 14 shows the accuracy of the two models for classification on the XMEGA_B target board. Figure 15 shows the number of power consumption traces (i.e. mean_rank) required by both models to recover the key for the XMEGA_B target board.

C. Analysis of experimental results

We have recorded the accuracy of both Model_raw and Model_data models during the training process, as shown in Fig.9 and Fig.10, and it can be seen that the Model_data model converges faster in accuracy during the training process.

To illustrate the effectiveness of Model_data, we used both models for cross-byte and cross-device attacks as a comparison test. From Fig.13 and Fig.15, it can be concluded that Model_data is more accurate than Model_raw in classifying both non-target byte power traces and cross-device power traces. In the following we quantify the efficiency of Model_data for recovering device key improvement using mean_rank. 73 power traces are required for Model_data to recover 16 byte keys and 1067 power traces are required for Model_raw to recover 16 byte keys, and the model trained with data power traces recovers the key, compared to the model trained with original power traces to recover key using data power traces is 14.62 times more efficient than recovering the key using the original power traces. For cross-device key recovery, Model_data requires 511 power traces to recover a 16-byte key, however, Model_raw is unable to recover the key.

Comparison with Experiment 1 demonstrates that the efficiency as well as the robustness of the model training can be improved by removing the power consumption trace components that are not correlated with the sensitive values.

Comparison Experiment 2 demonstrates that the model can improve the energy of the model to classify power traces across bytes by learning the components P_{data} and $P_{el.noise}$ in the power traces, but since the power traces are not simply quantifiable, this leads to a lower classification accuracy for non-target bytes.

Comparison experiment three also further demonstrates that by learning the components P_{data} and $P_{el.noise}$, the P_{data} components in the power consumption traces are similar even though the devices are different, and the same classification purpose can be accomplished, again because the components in the power consumption traces cannot be quantified by mathematical formulas, so the classification accuracy is further reduced across devices. In addition, the P_{const} varies from device to device, which also contributes to the further reduction in classification accuracy. Overall, models trained on the basis of data power consumption traces will be more efficient and have better overall performance on side channel attacks.

VII. Conclusion

This paper proposes a hardware cryptographic chip-side channel attack method based on power consumption trace decomposition, and tests and evaluates the method through the ChipWhisperer experimental platform. The experimental results show that: (1) the model is 3.64 times more efficient and has better robustness than the model trained with the original power traces in achieving the same classification accuracy of 99.00%; (2) the model is 14.62 times more efficient in recovering keys for cross-byte attacks than the model trained with the original power traces; (3) the model only needs 511 (3) the model only needs 511 power traces to recover all the keys when recovering keys across devices, while key recovery cannot be completed using the original power traces. Thus, the comprehensive performance of the model training method based on power trace decomposition is optimal.

Future work includes testing the approach of power trace decomposition presented in this paper on other implementations of cryptographic algorithms, as well as performing similar attacks on AES-enabled devices. In addition, we plan to further analyse the benefits of power trace decomposition for side-channel attacks by experimenting with power trace decomposition on other attack points. Of course, the most important future work should be to design defensive countermeasures against power trace decomposition-based side channel attacks.

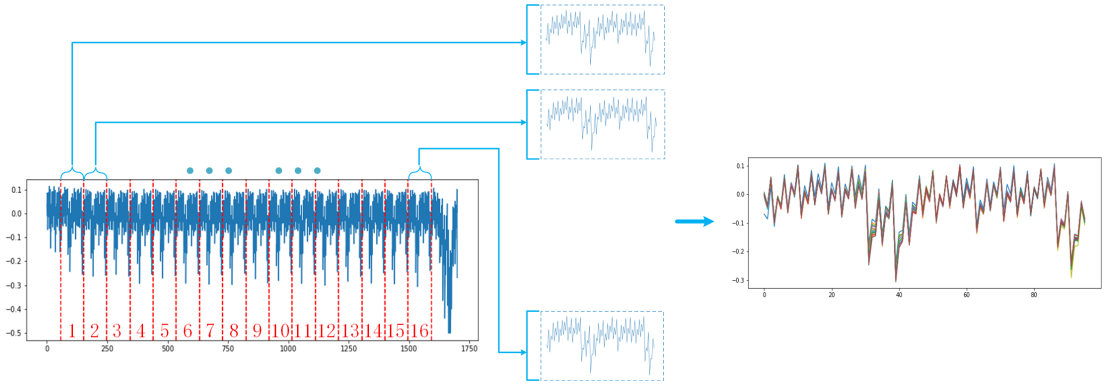

Fig. 12. 16 byte power trace comparison.

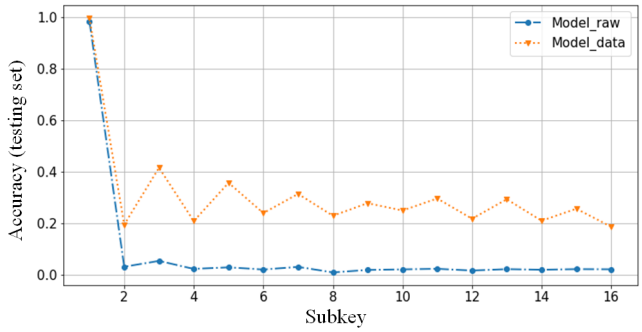

Fig. 13. Classification accuracy of the two models on the full byte test set.

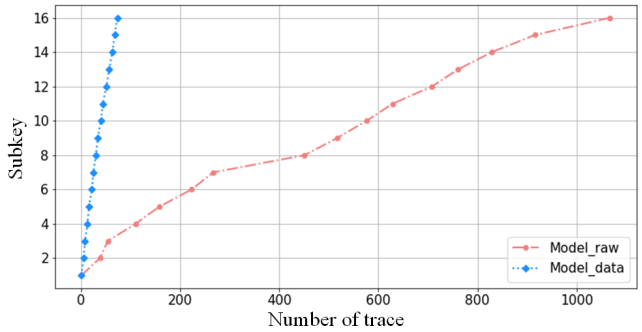

Fig. 14. Number of power traces used by both models to recover all keys.

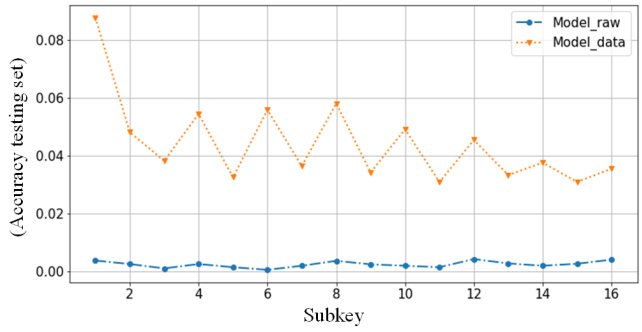

Fig. 15. Classification accuracy of the two models on all bytes of the target board XMEGA_B.

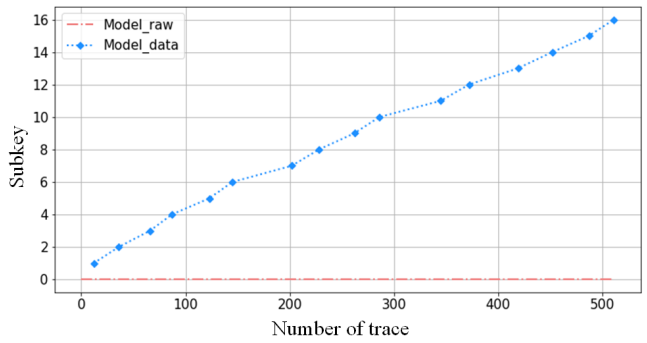

Fig. 16. Number of power traces used by both models to recover the full key of the target board XMEGA_B.

VIII. Acknowledgements

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] C. Herbst, E. Oswald, and S. Mangard, "An aes smart card implementation resistant to power analysis attacks," in *International conference on applied cryptography and network security*. Springer, 2006, pp. 239–252.

[3] M. Zhao and G. E. Suh, "Fpga-based remote power side-channel attacks," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 229–244.

[4] S. B. Ors, F. Gurkaynak, E. Oswald, and B. Preneel, "Power-analysis attack on an asic aes implementation," in *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, vol. 2. IEEE, 2004, pp. 546–552.

[5] A. A. Pammu, K.-S. Chong, W.-G. Ho, and B.-H. Gwee, "Interceptive side channel attack on aes-128 wireless communications for iot applications," in *2016 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 2016, pp. 650–653.

[6] 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.

[7] R. Wang, H. Wang, E. Dubrova, and M. Brisfors, "Advanced far field em side-channel attack on aes," in *Proceedings of the 7th ACM on Cyber-Physical System Security Workshop*, 2021, pp. 29–39.

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

[9] 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.

[10] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2009, pp. 443–461.

[11] 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.

[12] 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.

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

[14] 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] D. Kwon, H. Kim, and S. Hong, "Improving non-profiled side-channel attacks using autoencoder based preprocessing," *Cryptology ePrint Archive*, 2020.

[16] 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.

[17] 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.

[18] F. Hu, J. Wang, W. Wang, and F. Ni, "Software implementation of aes-128: Cross-subkey side channel attack," *Open Access Library Journal*, vol. 9, no. 1, pp. 1–15, 2022.

[19] Z. Liu, Z. Wang, and M. Ling, "Side-channel attack using word embedding and long short term memories," *Journal of Web Engineering*, pp. 285–306, 2022.

[20] F. Hu, H. Wang, and J. Wang, "Multi-leak deep-learning side-channel analysis," *IEEE Access*, 2022.