Journal of Electronic Testing (2024) 40:707–721
https://doi.org/10.1007/s10836-024-06149-z

RESEARCH

A Novel Two-Stage Model Based SCA against secAES

XiaoNian Wu[1] · JinLin Li[1] · RunLian Zhang[1] · HaiLong Zhang[2,3]

Received: 15 January 2024 / Accepted: 23 October 2024 / Published online: 31 October 2024

Abstract
secAES v2 is a mask scheme for AES and it is implemented with affine and shuffle techniques. Existing attack against secAES uses template attack, which first locates multiplicative leakage points and establishes templates to recover the two byte random mask, and then recovers secret key with template attack. Template construction can be complex in practice, and locating leakage points can also be difficult. Therefore, template attack may not perform well if conditions are not satisfied. In order to optimize the attack efficiency against the secAES mask scheme, a two-stage side-channel attack is proposed. First, a dense connected network DenseNet-121 is established to learn the multiplicative leakage characteristics according to traces from secAES's loadAndMaskInput function, and one byte multiplicative mask used can be recovered with a high probability. Second, based on the recovered multiplicative mask, key recovery is achieved using deep-learning side-channel attacks. The experimental results in simulation scenario show that compared with template attack, the proposed method does not need to find leakage points in the process of recovering one byte multiplicative mask. Furthermore, it is easier to implement, has lower computational complexity and has a higher success rate. Indeed, the DenseNet-121 network can be used to recover the multiplicative mask used by each plaintext block with a probability of 84%. During key recovery, based on the DenseNet-121 model, the required number of traces is reduced, only 4 traces are required to recover key bytes.

Keywords Side-channel attacks · SecAES · Dense connection network · Multiplicative masking

1 Introduction

Side-channel attacks (SCA) [1] recover the secret key used by a cryptographic chip by exploiting its physical leakage such as power consumption, electromagnetic emission, and timing information. In practice, SCA can pose a serious threat on the physical security of cryptographic chips. The earliest SCA method is timing attack [2], and then various classical SCA methods have been proposed, such as Differential Power Analysis (DPA) [3], Simple Power Analysis (SPA) [4], Template Attack (TA) [5], and Correlation Power Analysis (CPA) [6]. In order to improve attack efficiency, deep-learning side-channel attack [7] was proposed in 2016.

In order to counteract SCA, countermeasures such as power balancing circuits, random perturbations, and masking techniques have been proposed. Among them, masking [8] uses random numbers to randomize sensitive intermediate values processed by cryptographic chips, which can eliminate the statistical relationship between the physical leakage of a cryptographic chip and the processed intermediate value. The merit of masking is that it can be proven secure against SCA in the algorithmic level. For example, Fig. 1 illustrates a portion of the trace for the masked AES software implementation, with the horizontal axis representing time and the vertical axis representing voltage amplitude. In Fig. 1, HW() denotes the Hamming weight of a value, $\propto$ denotes the correlation between voltage amplitude of trace and Hamming weight of sensitive intermediate value, $f(\cdot)$ denotes the masking function, and $z1$ and $z2$ are two sensitive intermediate values related to the secret key. In Fig. 1, T_1, T_2, and T_3 denote the power leakage related to mask generation m, masking $z1$ with m, and masking $z2$ with m,

Responsible Editor: N. Karimi.

✉ RunLian Zhang
zhangrl@guet.edu.cn

[1] Guangxi Key Laboratory of Cryptography and Information Security, Guilin University of Electronic Technology, Guilin 541004, China

[2] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100085, China

[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

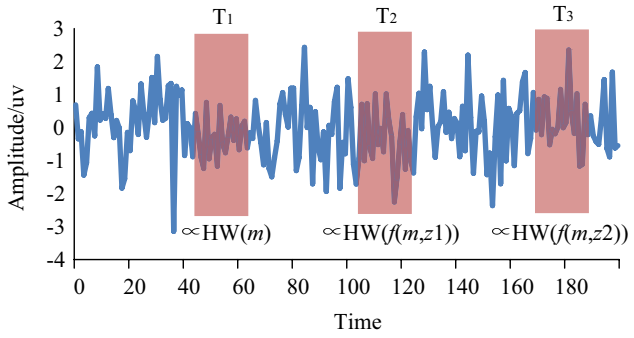

Fig. 1 A portion of the power trace of the masked AES software implementation

respectively. That is to say, the power leakages within these three regions can be correlated to the Hamming weight of m, $f(m, z1)$, and $f(m, z2)$. So, in order to compute the Hamming weight of m, $f(m, z1)$ and $f(m, z2)$, the random mask m should be known. Otherwise, the statistical relationship between the physical leakage and the processed intermediate value cannot be used anymore.

Even if masking can eliminate the statistical relationship between the instantaneous power consumption of a cryptographic chip and the processed sensitive intermediate value, by combining the power consumption of a cryptographic chip at multiple samples, one can rebuild this kind of statistical relationship, and we refer to this kind of attacks as higher-order SCA [9].

In the context of Fig. 1, in order to perform higher-order SCA, one should choose two samples in T1 and T2 and combine them with a certain technique. For example, one can multiply them. In this way, the effect of mask m may be eliminated and the statistical relationship between the Hamming weight of $z1$ and the combined power consumption can be used to recover the secret key. Similarly, one can choose two samples in T1 and T3, by combining them, one can build the statistical relationship between the combined power consumption and the Hamming weight of $z2$, which can be used to recover the secret key. The selection of samples and the methods used to combine leakages can be the crucial aspects of high-order SCA. In practice, many software masking schemes incorporate shuffling strategies, randomly altering the execution order of certain components, which make sample selection more challenging. Additionally, the spatial and temporal complexity of the attack will increase exponentially with the mask order, because more sample points from different regions need to be combined, which can make higher-order SCA more difficult to perform.

In 2020, Benadjila et al. [10] proposed an AES masking scheme called secAES V2, which combines multiplicative masking and shuffling strategies. The 1-byte multiplicative mask used in this scheme can be generated from 2-byte random masks. secAES V2 (which will be abbreviated as secAES later) is an outcome of the French National Cybersecurity Agency's (ANSSI) REASSURE project and it is primarily used for side-channel security research and the security of IoT devices. In 2021, Balli et al. discovered leakage in the multiplicative operation implementation of the secAES scheme and proposed an attack method that first recovers the masks and then recovers the secret key [11]. This method involves locating 16 leakage points related to the multiplicative operation and establishing 2^{16} templates for template matching to recover the 2-byte random masks. In the key recovery process, leakage related to the multiplicative operation between the mask and the secret key can be utilized, and template matching can be used to recover the secret key.

In practice, locating 16 leakage points is extremely challenging. Moreover, if the device experiences clock jitter, the noise level at 16 leakage points may increase, which would result in the construction of incorrect templates, rendering template attack infeasible. Furthermore, the computational complexity and the space complexity of building 2^{16} templates can be too high to be practically feasible. In order to simplify and optimize the efficiency of SCA against secAES, we propose a two-stage side-channel attack. The specific contributions are as follows:

(1) The attack process is divided into two stages, i.e. mask recovery and key recovery. In the mask recovery stage, a DenseNet-121 neural network model is established and trained using data labels. This model automatically learns the leakage characteristics of secAES during the execution of 16 multiplicative operations. In comparison to template attack, the proposed method employs a neural network model instead of constructing templates, which does not need to locate leakage points and create 2^{16} Hamming Weight Table. In light of this, the proposed method can be easier to be performed than template attack in practice. Besides, one only needs to recover 1-byte multiplicative mask rather than 2-byte random masks, which can be easier to be achieved. In the key recovery stage, deep-learning SCA can be used to recover the secret key. The output of loadAndMaskKey function in secAES can be chosen as the target intermediate value, and the recovered multiplicative masks can be used to compute the simulated power consumption.

(2) secAES is implemented and executed in an 8-bit Atmel AVR microcontroller ATmega2560, and the leakage generated are collected. Based on the implemented two-stage side-channel attack method and the collected side-channel traces, the evaluation experiments are carried out, and the experimental results indicate that in the mask recovery stage, the DenseNet-121 neural network can successfully recover the multiplicative mask with a success rate of 84%. Of course, it is not the highest success rate, but it is the best result in the learning rate we have attempted. In the key recovery stage, deep-learning SCA requires only 4 traces to recover the secret key.

The rest of this paper is organized as follows: Sect. 2 introduces the related work; Sect. 3 presents the secAES

masking scheme and existing attack methods; Sect. 4 provides a detailed description of the proposed two-stage side-channel attack; in Sect. 5, the efficiency of two-stage side-channel attack is evaluated in simulation scenario; finally, Sect. 6 concludes the paper.

2 Related Works

The first published work about SCA was timing attack, which was proposed by Kocher et al. in 1996 [2]. In this work, Kocher et al. used the timing information of a cryptographic chip to recover the secret key. Subsequently, DPA [3], SPA [4], TA [5], and CPA [6] have been proposed. Among them, CPA has shown good performance and it is one of the widely used attack styles in practice.

In order to counteract SCA, masking techniques are widely used. Depending on the type of operation applied to the sensitive intermediate value, different types of masking schemes, e.g., Boolean or multiplicative, are deployed. Boolean masking is suitable for linear operations like bit-wise XOR, and using Boolean masking for non-linear operations can increase the overhead. Multiplicative masking, on the other hand, is suitable for non-linear operations while not suitable for linear operations like bit-wise XOR. Masking can effectively eliminate the statistical relationship between the power consumption of a cryptographic chip and the processed sensitive intermediate value, and therefore makes SCA useless [12].

For cryptographic chips protected with masking, first-order SCA are no longer effective. However, second-order or higher-order SCA, which exploits the joint leakage of multiple intermediate values through data preprocessing, can remove the masking effect and successfully recover the secret key, albeit the data complexity of higher-order SCA can increase exponentially with the mask order. As early as 2000, Messerges et al. first proposed second-order SCA and the absolute difference preprocessing technique [13]. In 2006, Oswald et al. showed how to use second-order power analysis to attack DPA resistant software [14]. In 2010, Moradi et al. combined correlation power analysis with collision analysis and proposed the correlated collision power analysis, which successfully attacked AES hardware implementation protected with masking [15]. In 2014, BELGARRIC et al. [16] introduced an improved FFT second-order DPA processing technique, and they suggested that performing DPA in the frequency domain rather than in the time domain may yield better results. In 2016, Battistello et al. introduced Horizontal Side-Channel Attack (HSCA) [17], enabling attacks on masking that satisfy the ISW security framework. In 2021, Bouvet et al. [18] discovered that when a masked chip cannot run in constant time, a first-order CPA can break the protection of masking.

Compared with traditional SCA, deep-learning SCA leverages the powerful learning capabilities of deep-learning to discover vulnerabilities of cryptographic chips from a large number of traces. Currently, deep-learning SCA has become a powerful tool in the arsenal of SCA, especially when targeting cryptographic chips protected by masking. In 2016, Maghrebi [8] first used convolutional networks to attack AES implementation in both the unmasked and the masked scenarios, demonstrating the superior performance of deep-learning SCA. At CHES-2017, Cagli et al. combined convolutional neural networks with data augmentation to break a cryptographic chip protected by clock jitter [19]. In 2019, Timon introduced a new method to apply deep-learning techniques in a non-profiled context, which shows that by combining key guesses with observations of deep-learning metrics, it is possible to recover information about the secret key [20]. In the same year, Pfeifer et al. proposed a network structure called "Spread" and validated its effectiveness to implement first-order and second-order attacks against the ASCAD dataset [21]. In 2021, Kuroda et al. [22] studied the structure of multi-layer perceptrons and non-profiling deep- learning SCA against AES software implementation with two types of masking countermeasures including RSM. In 2022, Gohr et al. [23] proposed a side-channel attack against clyde-128 masking, using neural networks to predict intermediate value states after the first round S-box and subsequently deduce the real intermediate values. In 2023, Do et al. [24] introduced two multi-output regression models based on multi-layer perceptrons (MOR-MLP) and convolutional neural networks (MOR-CNN), which can significantly optimize the success rate of non-profiling deep-learning SCA against masked chip and reduce the computational complexity.

3 secAES Masking Scheme and Existing Attacks against secAES

3.1 secAES Masking Scheme

Two versions of secAES masking implementation are provided, tailored for STM32 and ATMega8515 respectively [11]. We focus on the security of secAES on ATMega8515 version2. In order to counteract SCA, this masking scheme disrupts the statistical characteristics of traces through multiplicative masking and shuffling. Each plaintext block encryption needs to be protected by 18 random bytes, denoted as $R[0] \sim R$ [17]. Among them, random masks $R[0]$ and R [1] are used to generate the multiplicative mask byte "a" and three arrays used for shuffling, which can be denoted as $\pi 0$, $\pi 1$, and $\pi 2$. In the initialization, the plaintext and the key are multiplied over GF(2^8) under the shuffling strategy of $\pi 0$, and the mask "a" is used to protect the real value. Subsequently, R [2] ~ R [17] are used to perform XOR operation with the masked plaintext to prevent the case that the multiplicative result is zero. This masking scheme is illustrated in Algorithm 1.

Algorithm 1 secAES masking scheme

Input: 16 byte plaintext group P, 16 byte key group K, 18 byte random number group R
Output: AES encryption result S

```
M = [R[2], ..., R[17]]; bin = R[16]; bout = R[17] ;
π0, π1, π2 = buildPermutation(R[0], R[1]);          //creating array
a = buildMultiplicativeMask(π0, π1);                //generating masking a
L = computeAndStoreMaskedSbox(a, R[15], bin, bout); //generating new S-boxes
S = loadAndMaskInput(P, M, a, π0);                  // masking plaintext
K = loadAndMaskKey(K, a, π0);                       //masking key
addRoundKey(S, K, π0);
for i = 0 to 9 do
    maskedSubBytes(S, L, M, bin, bout, π0);
    shiftRows(S, π0);
    shiftRows(M, π0);
    if i ≠ 9 then
        orderFromOneToTwo(S, M, π0);
        mixColumns(S, π0);
        mixColumns(M, π0);
        orderFromOneToTwo(S, M, π0);
    endif
    keyExpansion(K, a, bin, bout);
    addRoundKey(S, K, π0);
endfor
for i = 0 to 15 do
    S[i] = a^-1 × (S[i] ⊕ M[i]);
endfor
Return S;
```

In Algorithm 1, in order to ensure the correctness of the encryption result, one needs to use the multiplicative mask byte "a" to perform a multiplicative transformation on the original S-box to generate a new S-box. Then, the multiplicative mask byte "a" is used to mask all plaintext bytes, which is shown in Algorithm 2, and further protection is applied by XORing all plaintext bytes with 16 masking bytes $R[2]$~$R[17]$. In the loadAndMaskKey function, the multiplicative mask byte "a" is used to mask all secret key bytes, which is shown in Algorithm 3.

Algorithm 2 loadAndMaskInput

Input: Plaintext P, random masking block M, Shuffle sequence table $\pi 0$
Output: Mask status S
1: $M = R[2]$~$R[17]$
2: **for** $i = 0$ **to** 15 **do**
3: $j = \pi 0[i]$;
4: $S[j] = (a \times P[j]) \oplus M[j]$;
5: **endfor**
6: **Return** S;

Algorithm 3 loadAndMaskKey

Input: key K, Shuffle sequence table $\pi 0$
Output: Masked Key
1: **for** $i = 0$ **to** 15 **do**
2: $j = \pi 0[i]$;
3: $K[j] = a \times P[j]$;
4: **endfor**

Because the multiplication operations in loadAndMaskInput function and loadAndMaskKey function are performed over GF(2^8), in order to accelerate multiplicative operations, this scheme uses two arrays LOG and ALOG and each contains 256 elements to implement multiplication. The specific implementation algorithm for multiplication is as follows:

Algorithm 4 GF256_mul

Input: Multiplicative parameter $r16$ and $r17$
Output: Multiplicative Results on GF(2^8)
1: push $r0$, $r1$, $r18$, $r30$, $r31$ to stack
2: $r5 = 0$ **if** $r16 \cdot r17 = 0$, otherwise $r5 = 0$;
3: $r16 = LOG[r16]$;
4: $r18 = LOG[r17]$;
5: $r16 = (r16 + r18) \bmod 255$;
6: $r16 = ALOG[r16]$;
7: $r1 \,||\, r0 = r16 \cdot r5$;
8: $r16 = r0$;
9: pop $r31$, $r30$, $r18$, $r1$, $r0$
10: **Return** $r16$;

In Algorithm 4, the 2nd line and the 7th line are meant to ensure that regardless of the input values of $r16$ and $r17$, the multiplicative operation will terminate within the same amount of time, so as to prevent timing attacks.

In SCA, usually the S-box output of a block cipher implementation can be chosen as the target intermediate value. In order to counteract SCA, in the line 9 of Algorithm 1, one needs to use $\pi 0$ to shuffle the substitution process of 16 bytes of the plaintext and replace the original AES S-box with a new S-box table L calculated in the line 4 of Algorithm 1. The maskedSubBytes function is described in algorithm 5 as follows:

Algorithm 5 maskedSubBytes

Input: Encrypted data block S, new S-box table L, random masking block M, masking bin and out, Shuffle sequence table $\pi 0$
Output: maskedSubbBytes S
1: **for** $i = 0$ **to** 15 **do**
2: $j = \pi 0[i]$;
3: $z = S[j] \oplus M[j]$;
4: $z = L[z \oplus bin] \oplus M[j]$;
5: $S[j] = z \oplus bout$;
6: **endfor**
7: **Return** S;

In the maskedSubBytes function of Algorithm 5, the random mask M is first removed with the XOR operation in the line 3. Then, in the line 4, the SubBytes is performed using the S-box table L. Then, the random mask M is added back. Finally, in the line 5, the random *bout* is removed by the XOR operation. Based on the above implementation, the S-box output values can be protected by the mask "a" and *bout* or M.

According to Algorithm 2 to Algorithm 5, the secAES scheme described in Algorithm 1 uses independent random byte arrays R for each block encryption. Additionally, the S-box table L is recalculated using random byte arrays R before encrypting each block, which can ensure that the power consumption generated during the Subbytes computation is uncorrelated to the unprotected S-box output. Moreover, all intermediate values of secAES are protected by the mask byte "a" and the random array M. This design makes the secAES masking scheme secure against first-order SCA.

3.2 Existing Attacks on secAES

In the secAES masking scheme, the most important mask can be the multiplicative mask byte "a". All sensitive intermediate values are protected by "a", which is generated from random numbers $R[0]$ and R [1] (for convenience, these two random numbers will be referred to as $R_{0,1}$ in the following). In order to attack the secAES scheme, Balli et al. suggested to recover the random bytes $R_{0,1}$ first, and then recover the secret key [11].

First, the leakage generated in loadAndMaskInput function is exploited to recover $R_{0,1}$. In line 9 of Algorithm 4, during the process of popping the register, register $r0$ is set to 0. Since the current value in $r0$ is the result of multiplication, during the set-to-0 process, the register generates power consumption corresponding to the Hamming weight of the multiplicative result. In order to use this leakage, Balli et al. established nine Hamming weight templates with this leakage.

Then, a set of fixed plaintexts is selected, and all 2^{16} possible values of $R_{0,1}$ are traversed. For each value, they calculated the multiplicative mask byte "a" and the shuffle permutation table $\pi 0$ used in loadAndMaskInput function. Based on this, they computed a Hamming weight lookup table I_{hw} of size $2^{16} \times 16$, where each element represents the Hamming weight of the multiplicative result of the fixed plaintext with the generated multiplicative mask byte "a" and $\pi 0$ corresponding to each $R_{0,1}$ value. During the template matching process, for each target trace, they identified the leakage points in the trace generated by the 16 GF256_mul function called in loadAndMaskInput function where the pop operation was executed. Then, they computed sequentially the Bayesian probabilities for the 16 Hamming weights corresponding to each $R_{0,1}$ value in I_{hw} according to the established template. Using maximum likelihood estimation, they determined the most likely masking bytes $R_{0,1}$ in the trace, thus recovering $R_{0,1}$. Balli et al. stated that $R_{0,1}$ could be correctly recovered with a probability higher than 0.5.

Besides, in order to recover the secret key, Balli et al. utilized the leakage generated by the loadAndMaskKey function. They first calculated a probability table J of size $i \times j \times h$, where i represents the i-th trace, j represents the j-th leakage point in the trace (In total, there are 16 leakage points), and h represents the number of categories of Hamming weights. When recovering the k-th key byte, they calculated the probability $p_i = J[i, \pi_0^i, \mathrm{HW}(a^i \times g)]$ of all key guesses g at the i-th trace. Furthermore, using maximum likelihood estimation, they computed the probabilities of all key guesses g and select the key guess with the highest probability as the secret key.

The template attack proposed by Balli et al. requires one to precisely locate 16 leakage points in loadAndMaskInput function and create 2^{16} lookup table in order to recover the random masking bytes. Locating these leakage points accurately is challenging in practice. Indeed, if you want to construct an accurate probability density distribution function for 9-classes Hamming weight, you may need a large number of traces to obtain enough sample points.

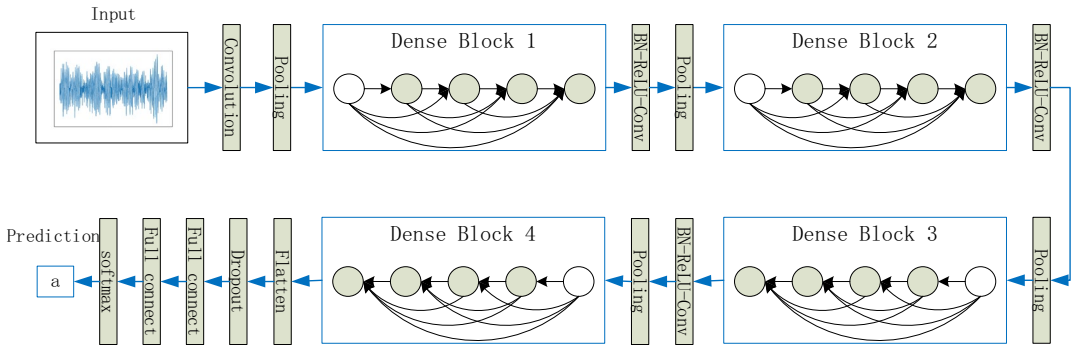

Fig. 2 the DenseNet-121 network structure

4 Two-Stage Side-Channel Attacks Method for secAES

In response to the issues exist in template attack [11], we propose a two-stage side-channel attack. Firstly, it employs deep-learning to recover the mask byte "*a*". Then, with the knowledge of the mask byte "*a*", deep-learning SCA is used to recover the secret key.

4.1 Recovering Masking Byte "*a*" Based on Deep Learning

In the proposed method, one can recover the mask byte "*a*" using the leakage generated during the execution of the GF256_mul function. In order to avoid difficulties in localization of interesting points, this paper employs a deep-learning side-channel attack to recover the mask byte "*a*". Widely used neural networks for deep-learning SCA are Multi-Layer Perceptrons (MLP) and Convolutional Neural Networks (CNN). Among them, CNN has been proven to be more effective in extracting features under shuffle protection. For instance, Benadjila et al. [25] successfully recovered the key of masked AES using traces in the ASCAD dataset with VGG-16 and ResNet-50 convolutional networks in 2020.

In order to obtain the data needed in deep learning, the secAES masking scheme was implemented into a microcontroller ATmega2560. As the secAES masking scheme exhibits leakage during the execution of the loadAndMaskInput function, both the traces in the training and testing datasets generated during the execution of the loadAndMaskInput function were used. The training dataset contains all the information related to the secAES masking operation, including 18 random masks, plaintext, ciphertext, power consumption and the runtime of the secAES masking scheme. The testing dataset, on the other hand, contains only the information of plaintext, ciphertext and the power consumption. Identity labeling was implemented for the data labels in training dataset. According to line 2 to line 3 of Algorithm 1, the mask byte "*a*" is generated from $R_{0,1}$, which consists of 2 bytes. Therefore, the data labeling computation needs to be consistent with the operation of mask byte generation. The computation of data labeling can be shown in Algorithm 6.

Algorithm 6 Calculate data labels

Input: Random bytes of an trace $R[0]$, $R[1]$
Output: The data label a of the trace

```
1:  Tb = [12, 5, 6, 11, 9, 0, 10, 13, 3, 14, 15, 8, 4, 7, 1, 2];
    //set constant
2:  m1 || m0 || m3 || m2 = R[0] || R[1];
3:  for i = 0 to 15 do      // create two tables using
    disordered operations
4:    π0[i] = Tb[Tb[Tb[Tb[i⊕m0]⊕m1]⊕m2]⊕m3];
5:    π1[i] = Tb[Tb[Tb[π0[i]⊕m1]⊕m2]⊕m3];
6:  endfor
7:  a0 = π0[2] || π0[2];
8:  a1 = π1[3] || π1[3];
9:  if a0 ≠ 0 then
10:   a = a0;
11: else
12:   a = a1;
13: endif
14: Return a;
```

Based on the data labeling method, the model can be constructed using the DenseNet-121 architecture. The model is trained to recover the mask byte "*a*". The DenseNet-121 network structure is illustrated in Fig. 2.

The DenseNet-121 structure is primarily consisted of an input layer, dense blocks, transition layers, and an output layer. The input layer receives external data and passes it to dense blocks for feature extraction. Since DenseNet-121 is originally designed for image classification, in this context, in order to utilize convolution operations to extract features from traces, it is necessary to reshape all one-dimensional trace data in the training and testing datasets into a two-dimensional image format. For example, transforming a trace with 10,000 points into a two-dimensional data of size $100 \times 100 \times 1$, like the array [0, 1,…,9999] reshape to [[0,1,.,99], [100,101,…,199], …, [9900,9901,…,9999]].

Before entering the dense blocks, a convolutional layer with a 7×7 kernel and a stride of 2 is applied to the input data to perform a convolution operation, extracting shallow-level features from the traces. Subsequently, a pooling layer is added to downsample the output of the convolutional layer, reducing the size of the data features. This ensures that the trace, before entering the dense blocks, has its features enhanced and compressed, allowing for deep feature learning in the subsequent dense blocks.

Each dense block is consisted of a specific number and size of BN-ReLU-Conv units. Each unit comprises a batch normalization layer, an activation layer with ReLU activation function, and a convolutional layer. The dense connectivity mechanism between units allows each unit to have direct access to all preceding units. The batch normalization layer normalizes the output of the convolutional layer, transforming it into features with zero mean and unit variance, which helps improve the convergence speed and the stability of the network. The activation layer sets negative input values to zero while keeping positive input values unchanged, introducing non-linearity that enables the network to learn non-linear features and decision boundaries. This non-linear characteristic selectively activates some neurons, aiding in feature selection and sparse representation, thereby enhancing the model's generalization and robustness. The convolutional layer uses 1×1 and 3×3 convolution kernels to apply filtering operations to input features, capturing important features in the input data. The spatial translational invariance of convolution can detect the same features scattered at different positions in traces, allowing convolutional networks to learn features in traces without the need for preprocessing, even when leakage points are misaligned.

The design of dense connections within dense blocks leads to a rapid increase in the number of features as the layers deepen, which can result in excessive information flow. Transition layers help control the flow and transmission of information by reducing the size and the number of channels in the data features, making the network more balanced and

Table 1 The methods generating two data sets

	Key	Plaintext
DATA-SET-1	$0 \times 0123456789abcdef123456789abcdef0$	$I_{j+1} = AES(K, I_j)$
DATA-SET-2		0xda39a3ee5e6b4b0d-3255bfef95601890

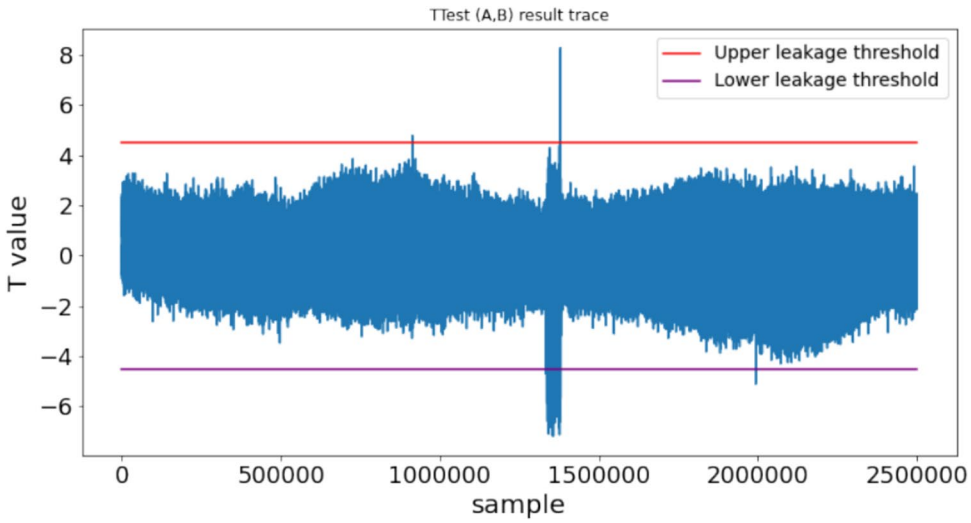

Fig. 3 The result of TVLA for secAES

stable. Transition layers are consisted of BN-ReLU-Conv units and average pooling layers. The convolutional layer in BN-ReLU-Conv units uses 1×1 convolution kernels to control the number of channels, while average pooling layers are used to reduce the size of data features. The data features processed through the transition layers are then input into the next dense block for deeper feature extraction.

The output module is consisted of a flatten layer, a dropout layer with a rate of 0.5, two fully connected layers with 4096 nodes each, and an output layer using the softmax function. The flatten layer transforms multidimensional data into a 1D vector while preserving the order of the data. The dropout layer with a rate of 0.5 randomly drops half of the features to prevent overfitting. The two fully connected layers each with 4096 nodes map the calculated feature space to the sample label space. Finally, the output layer uses the softmax function to output all features and map them to obtain the classification probability of each label, and the Maximum Likelihood Estimation (MLE) method is used to output the label with the highest probability.

During the model training process, the number of epochs is set to 40, the size of the batch is set to 64, the optimizer used is RMSprop, and the loss function used is cross-entropy. Cross-entropy is primarily used to measure the information between the differences of two probability distributions. The formula to calculate the cross-entropy $H(\cdot)$ of two probability distributions is as follows:

$$H(p,q) = -\sum_{i=1}^{k} p(x_i) \log q(x_i) \tag{1}$$

where k denotes the length of the discrete variable x_i, $p()$ and $q()$ are probability distributions of the discrete variable x_i. $p()$ represents the distribution of true labels, while $q()$ represents the predicted label distribution after model training. $H(p, q)$ is used to measure the similarity between $p()$ and $q()$.

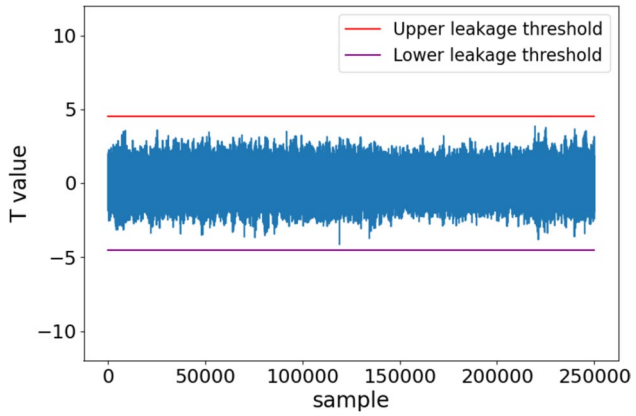

Fig. 4 TVLA result at loadAndMaskInput function for secAES

Based on the training set data and the data labeling method, the model is trained using the above-mentioned model.

During the testing phase, for each of the test set traces, a probability vector is computed to predict the probabilities that each multiplicative mask byte "a" candidate occurs. Among different values, the one with the highest probability can be selected as the actual multiplicative mask byte "a".

4.2 Recovery of the Key Using Deep-Learning SCA

According to Algorithm 3, there are 16 multiplicative operations, then, we would choose to recover the result of the multiplicative operation first by training the DenseNet-121 model. Furthermore, we can calculate the true key byte based on the mask byte "a" recovered.

Because convolutional networks can automatically handle misaligned traces without preprocessing, after recovering the multiplicative masked byte "a", the DenseNet-121 network model, which is used to recover the mask value, is used to recover the secret key.

During the training phase, the training dataset can be the same with the training dataset used to train the leakage property of the multiplicative mask byte "a", except that the leakage related to the calculation of maskedSubBytes function

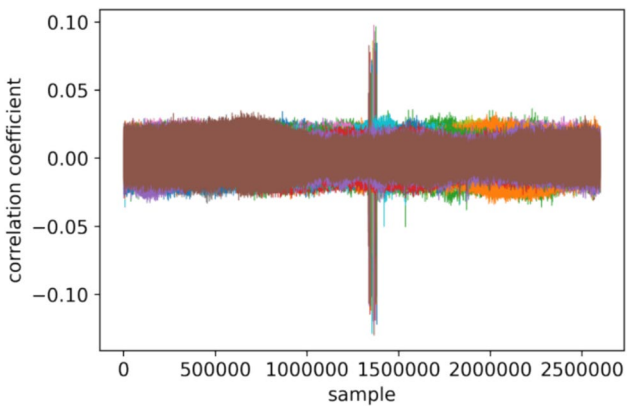

Fig. 5 The correlation coefficient of CPA against time samples

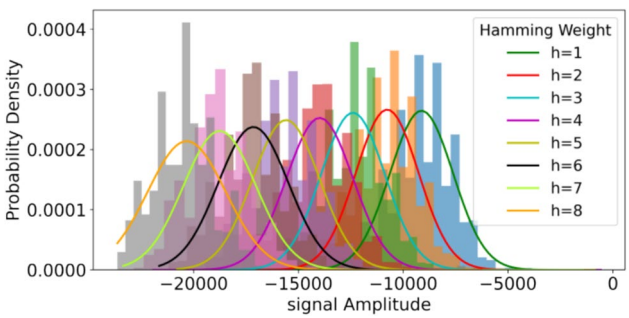

Fig. 6 The probability density result of 1th leakage location

is considered. Prior to leakage training, each trace should be labeled with formula (2) as follows:

$$Label = a \times K_i^j \tag{2}$$

where a denotes the multiplicative masked byte "a" in the trace, K_i^j is the j-th key byte of the key used in i-th trace. In the training phase of model, the parameter settings are the same with those used for training when recovering the mask, except for the learning rate, which is set to 0.00001. The testing phase of model for recovering the key is the same as the testing phase for mask recovery.

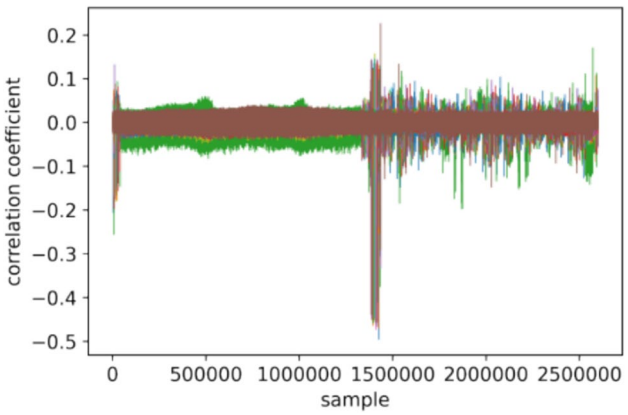

Fig. 7 The correlation coefficients on samples

5 Experimental Setups and Result Analysis

5.1 Experimental Environment

The experiments utilize an 8-bit Atmel AVR microcontroller ATmega2560. A sampling resistor is connected in series on the power supply path of the MCU. The oscilloscope probes are used to measure the voltage drop across the sampling resistor to capture the power consumption of the microcontroller. At the beginning of each encryption round, a pin is raised to trigger the PicoScope3206D oscilloscope for power consumption acquisition. Multiple sets of random plaintexts and random mask arrays are generated by a host computer and sent to the microcontroller. Encryption can be performed using both random and fixed keys, and the leakage generated during the execution of secAES can be collected. Additionally, the random plaintexts and random mask arrays are saved.

The computer hardware used for data collection is an i5-12500 processor with Windows 11 as the operating system and the size of RAM is 16GB. The computer hardware used for SCA is equipped with an Intel Xeon 6148 processor, with Ubuntu 20.04 as the operating system, the size of RAM is 256GB.

5.2 TVLA Experimental

The public implementation of secAES is on the ATMega8515 embedded device. We attempt to conduct TVLA (Test vector leakage assessment) [26] in order to evaluate the first-order security of our implementation on ATMega2560.

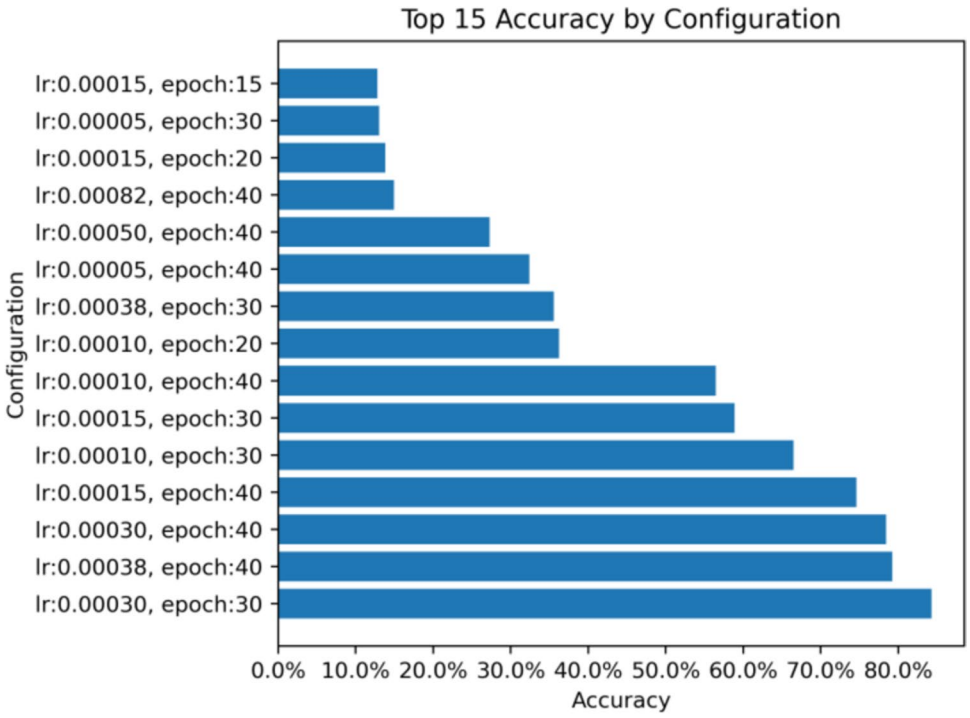

Fig. 8 Success rates of recovering mask byte with different learning rates

We collect two data sets DATA-SET-1 and DATA-SET-2 from the secAES encryption with a specific, published key and a set of data as follows in Table 1. For DATA-SET-2, the plaintext and the key are fixed. For DATA-SET-1, the key is fixed, the plaintext is dynamically changing according to AES encryption operation, and $I_0 = 0 \times 00000000000000000000000000000000$. The number of traces collected in two sets is 10,000, respectively. And the result of TVLA for secAES is as follows in Fig. 3.

In Fig. 3, except for the sample region around 1320000th of the loadAndMaskInput function, all other T values of samples are within the range of ± 4.5. But the loadAndMaskInput function is not related to the key, and the leakage cannot be used to complete a first-order attack against the secAES on the ATMega2560.

Furthermore, in order to analyze whether leakages at the loadAndMaskInput function affects the first-order security of secAES, we collect traces using fixed plaintext, random keys, and alternating fixed keys. Especially, aiming at leakages at the loadAndMaskInput function, we select 250,000 sample points starting from 1300000th sample point in Fig. 3, which includes operations such as loadAndMaskInput, loadAndMaskKey, addRoundKey, and the first round of encryption. Finally, 100,000 traces are collected for TVLA testing, and the result is shown in Fig. 4.

The result in Fig. 4 shows that all T values of samples are within the range of ± 4.5, which indicates that there is no leakage related with plaintexts at the loadAndMaskInput function under the fixed plaintext mode.

5.3 Replication Experimental

To compare the SNR (signal-to-noise ratio) levels of Atmega8151 and atmega2560, we try to reproduce attack experiment in [11], 501,000 traces related to the processing of loadAndMaskInput function are collected, plaintext is set to be fixed [1,2,..,16] and 18 masks are randomly generated. Among them, 500,000 traces (called D1) are used to establish probability density distribution models for 8 classes Hamming weights (based on the fixed plaintext, there is no case where the Hamming weight of the multiplicative result is 0), and the remaining 1000 traces (called D2) are used to attempt to recover two-byte mask $R_{0,1}$.

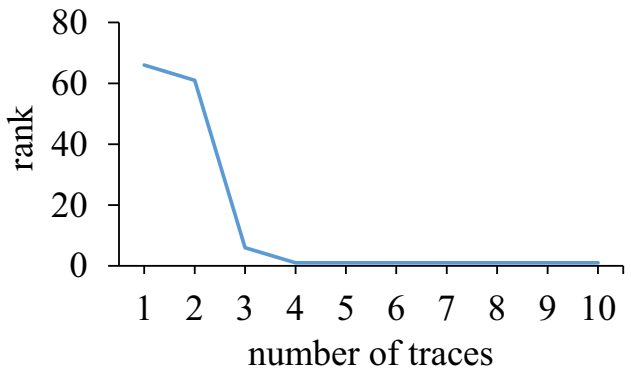

Fig. 9 The rank of deep-learning SCA against the first key byte

Table 2 Comparison on the number of traces of two methods against secAES scheme

Method	Mask recovery stage		Key recovery stage	
	Number of traces used to train the model	The probability to recover the mask	Number of traces used to train the model	Number of traces to recover the key
[11]	–	–	–	45
Our work	10 000	84%	10 000	4

In order to locate the 16 leakage points according to the loadAndMaskInput function when it is poccessing pop $r0$, CPA is used to do the work, because we cannot directly locate the points of the 60th clock as the original method in [11]. According to all known $R_{0,1}$ of D1, we calculate the result of $a \times P[j]$ in loadAndMaskInput function and the corresponding $\pi 0$, then multiplicative results are stored after sorted according to Table $\pi 0$. Furthermore, the correlation coefficient is computed by using CPA against every time sample, the result is indicated in Fig. 5.

The result in Fig. 5 shows that, the leakage (the region with the highest score) of the 16 multiplicative operations of pop r0 in GF256_mul of loadAndMaskInput function is obvious. The sample points in these regions will be used to recover the mask. In order to extract the sample points corresponding to the 16 leakage locations, we classify the power consumption at each leakage point according to the Hamming weight, and calculate the probability density function of each Hamming weight separately. The probability density result of the 1st leakage location is shown in Fig. 6, where, each histogram of different colors represents the number of power leakage values under a certain Hamming weight, and the curve graph is a probability density distribution function drawn based on the mean and standard deviation of power leakage classified according to the Hamming weight classes.

The results in Fig. 6 indicate that based on above 500,000 traces, there is a significant overlap among the probability density functions constructed based on power leakage classified by Hamming weight, which would have a significant impact on the effectiveness of recovering the mask bytes. Subsequent experiments have demonstrated this, we attempt to recover the 1000 masks $R_{0,1}$ by using Bayesian probability and maximum likelihood estimation on D2, but the success rate is only 16.7%. Furthermore, 1,000,000 traces are used to build the model, but the success rate is not increased. In this case, the following key recovery cannot be performed.

5.4 Recovery Experiment in Our Work

The DenseNet-121 network model is implemented based on tensorflow-gpu-2.2.0 using the Python language firstly. Then, we generate two data sets S1 and S2. S1 has 10,000 traces generated by random plaintext, random keys, and random mask bytes. S2 has also 10,000 traces generated by random plaintext, fix keys, and random mask bytes.

In order to recover the multiplicative mask byte "*a*", sample points corresponding to the the processing of loadAndMaskInput function are extracted. After extracting the sample points related to the processing of loadAndMaskInput function from dataset S1, the DenseNet-121 model is trained to recover the multiplicative masked byte "*a*". Subsequently, mask "*a*" is predicted with the same leakage of S2 in the same region.

But in the process of recovering the secret key, the leakage region is located at the specific location of the loadAndMaskKey function in the trace. So, we calculate the value of the loadAndMaskKey function given 18 random mask bytes and the true key, and arrange the values according to their respective $\pi 0$. Then, the correlation coefficients are computed with CPA, which is similar to the method shown in Sect. 5.3, and the result is shown in Fig. 7.

According to Fig. 7, the sample points corresponding to the leakage region related to loadAndMaskKey function are extracted from dataset S1, and the DenseNet-121 model is trained based on the sample data. Then, key recovery attack is conducted on the samples at the same leakage region in dataset S2.

5.4.1 Recover the Mask Byte with Different Learning Rates

Experiments are performed using the DenseNet-121 model according to the above environment. Figure 8 below illustrates bar charts depicting the success rates of recovering multiplicative mask byte "*a*" with the model under different learning rates and epoch.

The results in Fig. 8 indicate that the DenseNet-121 model consistently demonstrate a high probability of recovering multiplicative mask byte "*a*" under different learning rates, it is 84% in correctly recovering 10,000 unknown mask bytes when the learning rate is set to 0.0003 and epoch is set to 30.

5.4.2 Recover the Secret Key

The DenseNet-121 model is also used to recover each key byte according to the above environment. The number of traces needed to recover the secret key with deep-learning SCA against the first key byte is shown in Fig. 9.

Figure 9 shows that the secret key is successfully recovered with only 4 traces. In fact, with 4 traces, we can also successfully recover the other secret key bytes.

5.5 Comparison of Different Attacks against secAES

Balliet al. [11] suggested using template attack to target the secAES scheme by first recovering 2 random mask bytes and then recovering the secret key. In this study, a deep-learning SCA is used to first recover the multiplicative mask byte "*a*"; then, the model also can be used to recover the secret key. In detail, 10,000 traces are used to train the models, the sample points corresponding to the leakage region related to loadAndMaskInput function are extracted from 10, 000 traces to train the model during mask recovery stage, and the sample points corresponding to loadAndMaskKey function are extracted from the same 10,000 traces to train the model during key recovery stage. The comparison of the number of traces required for mask recovery and key recovery in different methods is summarized in Table 2, where "-" indicates an unknown value.

The results in Table 2 shows that [11], uses 45 traces to recover the secret key with template attack, but the number of traces built the model is unknown. Comparatively, the number of traces needed to training the model is 10,000 in our work. Of course, different sample points of 10,000 traces are used to train different models respectively, the probability recovering the mask byte is 84%, and the number of traces needed to recover the secret key with deep-learning SCA can be only 4.

Furthermore, the process of recovering 2 mask bytes with template attack involves locating 16 multiplicative leakage points and building 2^{16} lookup table. This can be complex to be achieved in practice. In contrast, the approach presented in this study uses deep-learning to recover the mask byte, which does not require the identification of leakage points and the building of 9 classes under the Hamming Weight model, making it feasible in practice.

6 Conclusion

In this study, a two-stage side-channel attack is proposed to recover the secret key used by the secAES scheme. It first uses deep-learning techniques to recover the mask bytes and then uses deep-learning SCA to recover the secret key. Compared to existing attacks against the secAES scheme, the proposed method has several advantages. Firstly, it is easier to be used in practice, as it does not require the identification of leakage points. Secondly, it has a higher success rate and lower computational complexity. Thirdly, the number of traces needed to recover the secret key is reduced.

Furthermore, compared to existing higher-order SCA that involve leakage combination, the proposed method does not require leakage combination. By first recovering the mask and then recovering the key, it effectively reduces

a second-order side-channel attack to a first-order one, significantly lowers the difficulty of the attack. In future work, the feasibility of applying the two-stage method to different types of mask schemes will be explored, and optimization methods for the proposed method will be researched to optimize the attack efficiency.

Funding This article is supported in part by the National Natural Science Foundation of China (62062026, 62272451), in part by the Key Research and Development Program of Guangxi in China (guike AB23026131), in part by the Innovation Project of Guangxi Graduate Education in China (YCSW2024347), in part by Beijing Natural Science Foundation (L234079).

Data Availability The datasets generated and analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing Interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Randolph M, Diehl W (2020) Power side-channel attack analysis: a review of 20 years of study for the layman. Cryptography 4(2):15
2. Kocher PC (1996) Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. Proc. Advances in Cryptology — CRYPTO '96. California, USA 104–113. https://doi.org/10.1007/3-540-68697-5_9
3. Kocher P, Jaffe J, Jun B (1999) Differential Power Analysis. Proc. Advances in Cryptology — CRYPTO' 99. California, USA 388–397. https://doi.org/10.1007/3-540-48405-1_25
4. Mangard S (2003) A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion. Proc. Information Security and Cryptology — ICISC 2002. Korea 343–358. https://doi.org/10.1007/3-540-36552-4_24
5. Chari S, Rao JR, Rohatgi P (2003) Template Attacks. Proc. Cryptographic Hardware and Embedded Systems - CHES 2002. CA, USA 13–28. https://doi.org/10.1007/3-540-36400-5_3
6. Brier E, Clavier C, Olivier F (2004) Correlation Power Analysis with a Leakage Model. Proc. Cryptographic Hardware and Embedded Systems - CHES 2004. MA, USA 16–29. https://doi.org/10.1007/978-3-540-28632-5_2
7. Maghrebi H, Portigliatti T, Prouff E (2016) Breaking Cryptographic Implementations Using Deep Learning Techniques. Proc. Security, Privacy, and Applied Cryptography Engineering. SPACE 2016. Hyderabad, India 3–26. https://doi.org/10.1007/978-3-319-49445-6_1
8. Prouff E, Rivain M (2013) Masking against Side-Channel Attacks: A Formal Security Proof. Proc. Advances in Cryptology – EUROCRYPT 2013. Berlin, Heidelberg 142–159. https://doi.org/10.1007/978-3-642-38348-9_9
9. Peeters E, Standaert FX, Donckers N, Quisquater JJ (2005) Improved Higher-Order Side-Channel Attacks with FPGA Experiments. Proc. Cryptographic Hardware and Embedded Systems – CHES 2005. Berlin, Heidelberg 309–323. https://doi.org/10.1007/11545262_23
10. Benadjila R, Lomné V, Prouff E, Roche T Secure AES128 for ATMega8515. 2020-03-27. https://github.com/ANSSI-FR/secAES-ATmega8515
11. Balli F, Caforio A, Banik S (2021) Some applications of hamming weight correlations. Cryptol ePrint Archive. https://eprint.iacr.org/2021/611.pdf
12. Akkar ML, Giraud C (2001) An implementation of DES and AES, secure against some attacks. Proc. International Workshop on Cryptographic Hardware and Embedded Systems, Berlin, Heidelberg 309–318. https://doi.org/10.1007/3-540-44709-1_26
13. Messerges TS (2000) Using Second-Order Power Analysis to Attack DPA Resistant Software. Proc. Cryptographic Hardware and Embedded Systems — CHES 2000. Berlin, Heidelberg 238–251. https://doi.org/10.1007/3-540-44499-8_19
14. Oswald E, Mangard S, Herbst C, Tillich S (2006) Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. Proc. Topics in Cryptology – CT-RSA 2006. Berlin, Heidelberg 192–207. https://doi.org/10.1007/11605805_13
15. Moradi A, Mischke O, Eisenbarth T (2010) Correlation-Enhanced Power Analysis Collision Attack. Proc. Cryptographic Hardware and Embedded Systems, CHES 2010. Santa Barbara, USA 125–139. https://doi.org/10.1007/978-3-642-15031-9_9
16. Belgarric P, Bhasin S, Bruneau N, Danger JL, Debande N, Guilley S, Heuser A, Najm Z, Rioul O (2014) Time-Frequency Analysis for Second-Order Attacks. Proc. Smart Card Research and Advanced Applications. CARDIS 2013. Berlin, Germany 108–122. https://doi.org/10.1007/978-3-319-08302-5_8
17. Battistello A, Coron JS, Prouff E, Zeitoun R (2016) Horizontal Side-Channel Attacks and Countermeasures on the ISW Masking Scheme. Proc. Cryptographic Hardware and Embedded Systems – CHES 2016. Berlin, Heidelberg 23–39. https://doi.org/10.1007/978-3-662-53140-2_2
18. Bouvet A, Guilley S, Vlasak L (2021) First-Order Side-Channel Leakage Analysis of Masked but Asynchronous AES. Proc. Security and Privacy. ICSP 2021. Communications in Computer and Information Science. 16–29. https://doi.org/10.1007/978-3-030-90553-8_2
19. Cagli E, Dumas C, Prouff E (2017) Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures. Proc. Cryptographic Hardware and Embedded Systems – CHES 2017. Taipei, Taiwan 46–68. https://doi.org/10.1007/978-3-319-66787-4_3
20. TIMON B (2019) Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Trans Cryptographic Hardw Embedded Syst 107–131. https://doi.org/10.13154/tches.v2019.i2.107-131
21. Pfeifer C, Haddad P, Spread (2018) a new layer for profiled deep-learning side-channel attacks. IACR Cryptology ePrint Archive 2018:880
22. Kuroda K, Fukuda Y, Yoshida K, Fujino T (2021) Practical Aspects on Non-profiled Deep-learning Side-channel Attacks against AES Software Implementation with Two Types of Masking Countermeasures including RSM. Proc. 5th Workshop on Attacks and Solutions in Hardware Security, Virtual Event, Republic of Korea 29–40. https://doi.org/10.1007/s13389-023-00312-6
23. Gohr A, Laus F, Schindler W (2022) Breaking masked implementations of the clyde-cipher by means of side-channel analysis: a report on the ches challenge side-channel contest 2020. IACR Trans Cryptographic Hardw Embedded Syst 397–437. https://doi.org/10.46586/tches.v2022.i4.397-437
24. Do NT, Hoang VP, Doan VS (2023) A novel non-profiled side channel attack based on multi-output regression neural network. J Cryptographic Eng 14(3):1–13. https://doi.org/10.1007/s13389-023-00314-4

25. Benadjila R, Prouff E, Strullu R, Cagli E, Dumas C (2020) Deep learning for side-channel analysis and introduction to ASCAD database. J Cryptographic Eng 10(2):163–188. https://doi.org/10.1007/s13389-019-00220-8
26. SCHNEIDER T, MORADI A (2016) Leakage assessment methodology: extended version. J Cryptographic Eng 6:85–99. https://doi.org/10.1007/s13389-016-0120-y

XiaoNian Wu XiaoNian Wu, born in 1972, holds an M.S. degree. He is a professor and his research interests include information security and distributed computing.

JinLin Li JinLin Li, born in 1997, is a Masters degree candidate. His research interests include side channel analysis.

RunLian Zhang RunLian Zhang, born in 1974, holds a Ph.D. She is an associate professor and her research interests include information security.

HaiLong Zhang HaiLong Zhang, born in 1986, holds a Ph.D. He is an associate professor and his research interests include side channel cryptanalysis.