DLFA: Deep Learning-based Fault Analysis against Block Ciphers

Yukun Cheng[1], Changhai Ou[1], Fan Zhang[2], Shihui Zheng[3], Shengmin Xu[2] and Jiangshan Long[1]

[1] School of Cyber Science & Engineering, Wuhan University, Wuhan, HuBei, China,
{kuin33,ouchanghai,longjiangshan}@whu.edu.cn;
[2] College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang, China,
fanzhang@zju.edu.cn
[3] School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China
shihuizh@bupt.edu.cn
[4] Fujian Provincial Key Laboratory of Network Security and Cryptology, College of Computer and Cyber Security, Fujian Normal University, Fuzhou, China
smxu1989@gmail.com

Abstract. Previous studies on fault analysis have demonstrated promising potential in compromising cryptographic security. However, these fault analysis methods are limited in practical impact due to methodological constraints and the substantial requirement of faulty information such as correct and faulty ciphertexts. Additionally, while deep learning techniques have been widely applied to side-channel analysis (SCA) in recent years and have shown superior performance compared with traditional methods, there has been minimal research focusing on the application of deep learning techniques in fault analysis to date. This paper proposes an innovative approach named deep learning-based fault analysis (DLFA) by incorporating deep learning techniques into fault analysis, which enhances the efficiency and versatility of the analysis. DLFA is equipped with a novel feature selection method to extract valuable information for neural networks from faulty ciphertexts. Besides, optimized hyper-parameters for MLP and CNN are presented as the benchmarks. Experimental results on advanced encryption standard (AES) reveal that DLFA achieves outstanding performance. Notably, DLFA requires only 683 minimum and 1488 average ciphertexts with an average analysis time of 0.12s, surpassing previous works in terms of efficiency. IACR Transactions.

Keywords: Deep Learning · Multi-Layer Perceptron · Convolutional Neural Network · Fault Analysis · Advanced Encryption Standard

1 Introduction

To provide a reliable and efficient computing environment, cryptographic algorithms are implemented in particular cryptographic devices. Security in such devices relies not only on the mathematical security of the cryptographic algorithm, but also on the physical security of the cryptographic implementation. Nowadays, implementation-based attacks have attracted intensive attention. Such attacks pose a serious threat to cryptographic security, as they can exploit leakage of physical information that is neglected in conventional cryptanalysis.

Depending on the leakage of physical information, implementation-based attacks can be classified into passive and active attacks. For passive attacks, the adversary does

not interfere with the work of the cryptographic device. The adversary plays the role of an observer and collects key-related information from the power consumption [KJJ99], electromagnetic emanation [GMO01] and other leakages of physical information while the cryptographic device is in operation. As a representative of passive attacks, side-channel analysis (SCA) attracted wide attention. Different methods, such as simple power analysis (SPA) [Man02], correlation power analysis (CPA) [BCO04] and template analysis (TA) [CRR02], can be launched to recover the key. On the other hand, fault analysis (FA), which is the representative of active attacks, aims to induce faults on the cryptographic device using some physical methods, such as changing the power supply voltage [BMM00], varying the temperature [GA03] and using the pulsed laser [CLFT14]. The adversary analyzes the faulty outputs of the cryptographic device to recover the secret key. Differential fault analysis (DFA) [BS97] and statistical fault analysis (SFA) [FJLT13] are the most prominent fault analysis over the years. Additionally, statistical ineffective fault analysis (SIFA) [DEK+18] and persistent fault analysis (PFA) [ZLZ+18] also show their tremendous potential as new analysis methods in recent years.

In the traditional FA, the exploitation of fault leakage is solely dependent on the chosen analysis method, which is primarily focused on successfully recovering the key under specific restrictions. Moreover, these methods struggle to fully harness the potential of fault leakage and consequently necessitate a substantial number of faulty encryptions.

In this study, we apply deep learning techniques to fault analysis to improve the efficiency and versatility of the attack. Our primary objective is to recover the internally processed secret key by utilizing feature information obtained from the faulty cryptographic implementation. To address this intricate problem, we propose a novel approach called deep learning-based fault analysis (DLFA) using a divide-and-conquer strategy.

1.1 Related Works

In 1991, Rivest first formulated how provable machine learning techniques and cryptography related to one another in [Riv91]. The combination of machine learning techniques and SCA was first proposed by Backes et al. [BDG+10], who recovered the printed text from a printer using an acoustic side channel. Hospodar et al. proposed the first study on the application of machine learning techniques in SCA [HGM+11]. From here, a large number of works have been proposed to break both unprotected [BL12, LPB+15] and protected cryptographic implementations [GHO15, LBM15] using machine learning techniques.

In recent years, more researchers have focused on deep learning techniques, such as multilayer perceptron networks (MLP) [MZVT15, MDM16] and convolutional neural networks (CNN) [PSK+18] to go with the trend in the machine learning community. Of the various SCA, profiled attacks, represented by TA, are the most powerful ones and in some cases can recover the key with only one power trace. These attacks can be composed of two phases: profiling and matching, which are similar to the training phase and test phase in deep learning. On this basis, deep learning techniques have been introduced into SCA as an alternative to TA [KPH+19, CDP17, BPS+20]. Compared to TA, deep learning-based SCA is more robust to noise and shows comparable or better performance [Tim19].

On the other hand, there is less research on the combination of machine learning or deep learning techniques with FA. Several researchers have introduced FA into deep neural networks for misclassification [LWLX17, BHJ+18]. Moreover, Saha et al. [SAB+20] have applied deep learning techniques to the leakage assessment of FA. However, to the best of our knowledge, the investigation of the application of machine learning or deep learning techniques in the key recovery of FA is a void in the field of cryptography.

Table 1: Comparison with Related Works on AES-128

Analysis Method	Average N_c	N_f	Analysis Time(s)	Ciphertext Access	Middle Round Attack
PFA[ZLZ+18]	2273	1	10.48	✓	×
PFA-MLE[ZZJ+20]	1641	1	11.03	✓	×
EPFA [XZY+21]	970	1	8.64	✓	✓
SPFA [ESP20]	1643	16	66.22	✓	×
SFA [PCNM15]	304	633	2649.83	✓	×
FTA [SBR+20]	-	4112	0.96	×	✓
DLFA	1488	1	0.12	✓	×

1.2 Contributions

Our contributions in this paper can be summarized as follows:

- To improve the efficiency and versatility of FA, we undertake a comprehensive investigation into the application of deep learning-based key recovery within the framework of fault analysis and propose DLFA as an innovative FA method. We select MLP and CNN as neural network models and comb the attack process. The intricate problem of key recovery is decomposed into two tractable sub-problems: feature selection and classification.

- We analyze the impact of faults and propose an effective method for selecting features. The probability distributions of the faulty outputs of the S-box, which contain a substantial amount of pertinent information regarding the key, are chosen as the feature set. Additionally, the dataset is prepared for subsequent training and testing steps of the neural network model.

- To comprehensively assess the effectiveness of our attack, we conduct DLFA on advanced encryption standard (AES). We explore various parameterization options in our experiments and present optimized hyper-parameters for MLP and CNN models, which can serve as a valuable reference for future researchers when designing novel deep learning models.

One of the most significant contributions of DLFA is that it can efficiently recover the key with only a few required ciphertexts N_c and only one fault, within a short analysis time. Table 1 gives a summary of our results on AES-128 in comparison with related works, including the original PFA [ZLZ+18], the improved PFA with Maximum Likelihood Estimation (PFA-MLE) [ZZJ+20], the extended PFA with GPU acceleration (EPFA) [XZY+21], the statistical PFA (SPFA) [ESP20], SFA[PCNM15] and FTA[SBR+20]. Compared with previous work, DLFA achieves a suitable trade-off among N_c, the required number of faults N_f and analysis time.

1.3 Organizations

The remainder of the paper is organized as follows: We begin with the selection of the fault model and feature set in Section 2. Section 3 introduces the neural network models employed in this work and provides a comprehensive description of DLFA. The experimental results are presented in Section 4. Additionally, within this section, we discuss the impact of hyper-parameters on the results and highlight the superiority of our attack. Some brief discussions on the performance of machine learning techniques and the impact of the position and value of the fault are presented in Section 5. Finally, we conclude this paper with a summary of our attack and an outlook for future work in Section 6.

2 Fault Model and Feature Selection

In this section, we initially give a brief introduction to the fault model selected in this article. Then, we provide a brief overview of the characteristics of the fault. Subsequently, we elucidate the methodology for feature selection, which serves as an essential preliminary step in deep learning. Finally, we present the dataset to facilitate resolution of the classification problem.

2.1 Fault Model

For a systematic study of FA, faults are categorized by effects, such as modifying the control flow, skipping commands, or altering storage information. The most typical one in each category is chosen as the representative to build the fault model. The fault model explains the position (e.g. before or after a specific operation), the property (e.g. transient, permanent and persistent), the effect of the fault and the ability of the adversary.

In this article, we focus on the fault model of PFA in [ZLZ+18], which can be described as follows:

1. The faults are injected into the cryptographic device before the encryption of the block cipher.
2. The injected faults randomly corrupt the stored constants of the algorithm (i.e. the elements in the S-box) in the storage of the device.
3. The injected faults are persistent, meaning that the faulty constants are used during the encryption until the storage is refreshed or the device is rebooted.
4. The adversary is able to feed plaintexts into the faulty device and collect ciphertexts for subsequent fault analysis.

Such a choice of fault model has several advantages. In contrast to traditional fault models, which are mostly transient, the fault model of PFA places minor restrictions on the adversary's capabilities, such as his ability to inject faults within a relatively loose time window. Then, according to the persistence property, the fault can be exploited multiple times with only one injection. Also, note that the faults lead to a non-uniform distribution of the outputs of the S-box. This non-uniform distribution favors feature selection, as will be explained in detail below.

2.2 Characteristic of the Fault

While the fault categories in FA may vary, they all share a common characteristic: from the circuit point of view, every fault can be perceived as a transition of a specific intermediate variable from its correct value x to a faulty value x'. This transition disrupts the distribution of the intermediate variable by introducing an abnormal bias.

Indeed, this characteristic has already been leveraged in the case of SFA [FJLT13] and SIFA [DEK+18]. In both studies, the adversary evaluates the disparity between the expected uniform distribution in the fault-free scenario and the practical distribution affected by byte-level faults using a metric named Squared Euclidean Imbalance (SEI), which can be described as follows:

$$SEI(\hat{k}) = \sum_{\delta=0}^{2^s-1} \left(\frac{\#\left\{ i \mid x^{(\hat{k})} = \delta \right\}}{N} - \frac{1}{2^s} \right)^2, \tag{1}$$

where $\hat{k}$ denotes the hypothesis of the key and s denotes the bit length of the intermediate variable $x^{(\hat{k})}$ which correlates to $\hat{k}$. Thus the number of different values of $x^{(\hat{k})}$ can be

expressed as 2^s. We use $\frac{\#\left\{i|x^{(\hat{k})}=\delta\right\}}{N}$ and $\frac{1}{2^s}$ to denote the practical distribution of x and the uniform distribution, respectively. The SEI of each $\hat{k}$ is computed as the sum of the squared difference between the practical and the uniform distribution in all values of x. In the fault-free scenario, with the increase of encryption, the two distributions are almost equal thus the SEI of each $\hat{k}$ is close to 0. In the faulty scenario, for the wrong hypothesis of the key, each value of $x^{(\hat{k})}$ occurs randomly in the same probability. However, for the correct hypothesis of the key, with the bias caused by the fault, the distribution of $x^{(\hat{k})}$ becomes non-uniform, which leads to the increase of the SEI. Hence, the hypothesis that the key has the highest SEI can be considered correct.

The aforementioned key recovery approach based on the SEI demonstrates that the disparity of the intermediate variable induced by the fault encompasses accessible information regarding the key. A detailed exposition of feature selection based on the characteristic of the fault will be presented subsequently.

2.3 Feature Selection

Before solving the classification problem, feature selection is used as a preliminary step in deep learning, which filters out and preprocesses the given raw data. The significance of feature selection can mainly be represented in two aspects. On one hand, this step filters out erroneous features and thus improves the accuracy of the neural network. On the other hand, the features containing larger relevant pieces of information are extracted, which reduce the computational cost of the neural network.

Unfortunately, in the field of FA, the problem of feature selection is more intricate compared to SCA, where features can be directly selected from power traces. The effects of faults are diverse, resulting in a wide range of data used in different fault analysis methods. For instance, DFA conducts an analysis based on the differential values between the correct and faulty ciphertexts generated from the same plaintext. SFA exploits biases caused by the fault to recover the key. Moreover, the adversary of PFA pays more attention to the distribution of ciphertexts after numerous encryptions. Consequently, finding a uniform criterion for feature selection becomes challenging.

In this paper, we focus on the SPN-based block ciphers (e.g. AES and PRESENT) and the fault is injected into the S-box of the cipher. Reviewing the characteristic of the fault, the output of the S-box can be considered as the intermediate variable affected by the fault. Due to the avalanche effect, for the correct encryption, the distribution of the output of the S-box converges to a uniform distribution, which is consistent with the SEI requirement. Unfortunately, the SEI is merely a calculated value that is too brief to be directly selected as the feature. However, the feature selection problem can be solved with the help of the concept of SEI. Note that the distribution of the intermediate variable in the fault-free scenario is a fixed value (i.e. $\frac{1}{2^s}$), which is associated only with the bit length of the intermediate variable. Thus, from the information-theoretic point of view, the key-related information is contained in the practical distribution of the intermediate variable. That is, the adversary can select the practical distribution of the intermediate variables as the features.

2.4 Details of the Dataset

At the completion of feature selection, the dataset is prepared for the training and testing of the neural network in the next stage. As mentioned above, we select the distribution of the output of the S-box of the SPN-based block cipher as the features. More specifically, the distribution of the output of the S-box in the last round is selected to reduce the computational overhead. The key of the last round is selected as the label for each group of features. Due to the nature of the block cipher, the features and labels are processed

in bytes or nibbles. For the sake of simplicity, the term sample is used to denote a set of features. The examples of the sample on each cipher are given in Fig. 1.

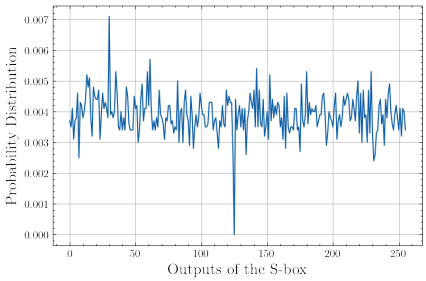


Figure 1: Examples of the samples on AES.

In addition to this, other relevant information, such as the plaintext, the ciphertext and the master key, can also be selected as the metadata. The efficiency of the neural network in deep learning techniques can be confirmed by the mere use of the labels. However, since these data are necessary to check for the correctness of the features and labels, the metadata are helpful for the adversary.

3 Deep Learning-Based Fault Analysis

In this section, we first introduce the basic information about MLP and CNN, which are the two neural network models used in this paper. The method of how to implement FA using deep learning techniques is shown later. Finally, we provide a general description of DLFA in detail.

3.1 MLP and CNN

MLP and CNN are two widely used neural network models in deep learning techniques. Both of them fall into supervised learning, where a neural network is trained on data labeled with the corresponding labels. When the training phase is complete, the neural network is able to give the prediction for another set of inputs based on the rank of some indicators, such as probability or likelihood score.

MLP is a classical artificial neural network, inspired by the human nervous system. The artificial neuron (i.e. perceptron), whose basic structure is shown in Fig. 2(a), is the basic element of the network. In a mathematical sense, an artificial neuron consists of an affine transformation nested in an activation function, which can be defined as follows:

$$y = \alpha(\vec{w} \cdot \vec{x} + b), \tag{2}$$

where $\vec{x}$, y, α denote the input, output, and activation function of the artificial neuron, respectively. The weight and bias parameters of the affine transformation are denoted as $\vec{w}$ and b. Naturally, a single artificial neuron is powerless to deal with complex problems. Therefore, MLP is designed by joining artificial neurons together to adapt to complicated situations. As shown in Fig. 2(b), the output of one layer is connected to the following layer as input. Let λ denote the fully-connected layer, which is the set of the affine transformation with the same input, then the model $\mathcal{M}$ of MLP can be defined as follows:

$$\mathcal{M}(\vec{x}) = s \circ \lambda_n \circ \alpha_{n-1} \circ \lambda_{n-1} \circ ... \circ \alpha_1 \circ \lambda_1(\vec{x}). \tag{3}$$

Note that s denotes the last activation function which is the special one. The activation functions from α_1 to α_{n-1} have a variety of options, such as sigmoid or ReLU. However, s is invariably chosen from the softmax function or its variants, which can be written as:

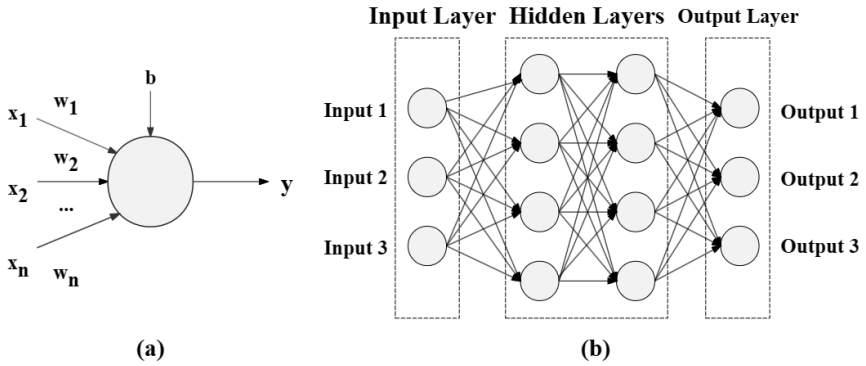


Figure 2: Structure of the artificial neuron and MLP. (a) Structure of single artificial neuron. (b) Structure of a four-layer MLP.

$$s(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{C} e^{z_i}}, \tag{4}$$

where C denotes the number of classes of the network's output. The softmax function renders the output of the network as a probability distribution of all classes. The classification problem can be solved by choosing the class with the highest one or several probabilities as the final output of the neural network model.

Compared to MLP, CNN is better at learning the internal representations of two-dimensional images. On one hand, CNN inherits the hierarchical structure from MLP. On the other hand, CNN broadens the type of layers to improve the efficiency of the network. In general, a CNN starts from the convolutional layer followed by the activation function. The convolutional layer extracts high-level abstract features from the input through the filters, and the activation function introduces non-linearities to increase the goodness-of-fit of the network. Then, the pooling layer is introduced to restrict the number of neurons. The convolutional layer, the activation function and the pooling layer form the main block of the CNN, which is reused in the network until desired output. Moreover, the batch normalization layer is sometimes used as an option to improve the generalization ability of the network. After that, the flatten layer compresses the data into one dimension, and some fully-connected layers followed by a softmax function at the end of the network are added to summarize the global information and produce the final output. The structure of CNN is given in Fig. 3.

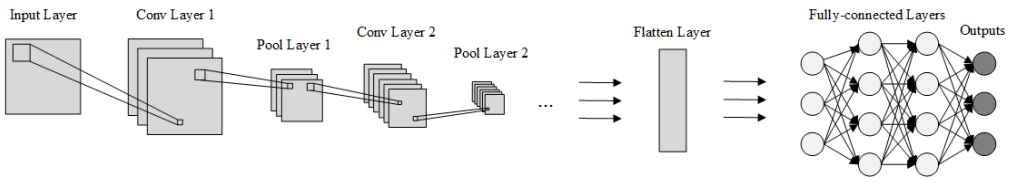


Figure 3: Structure of CNN.

Similarly, the model $\mathcal{C}$ of CNN can be characterized as follows:

$$\mathcal{C} : s \circ [\lambda]^{n_1} \circ [\rho \circ \alpha \circ \iota]^{n_2}, \tag{5}$$

where ι denotes the convolutional layer and ρ denotes the pooling layer. We recall that α, λ and s denote the activation function, the fully-connected layer and the softmax function, respectively. Moreover, n_1 and n_2 denote the repetition times of the fully-connected layer and the main block, respectively.

3.2 Combination of Fault Analysis and Deep Learning

To combine fault analysis and deep learning, our attack can be composed of two phases: the training phase and the attack phase. Thus, two independent datasets, the training set and the attack set, are prepared for the analysis. As mentioned above, the training set $\mathcal{S}_T$ of size N_T consists of a quantity of independent identically distributed samples and the corresponding labels, which can be described as:

$$\mathcal{S}_T : \{(\vec{s_i}, k) | 1 \leq i \leq N_T\}, \tag{6}$$

where $\vec{s_i}$ denotes the sample, and k denotes the label as it represents the last round key in our dataset. In the training phase, the adversary aims to construct a generative model. The parameters of the network, denoted by θ, will be updated iteratively to fit the estimation g_k of every k from the key space $\mathcal{K}$. The model M_θ can be described with the following conditional probability distribution function:

$$M_\theta : \{g_k : (\vec{s}, k) \rightarrow \Pr[\vec{S} = \vec{s} | K = k]\}_{k \in \mathcal{K}}, \tag{7}$$

where the upper-case letter $\vec{S}$ and K denote the random variables over the set of $\vec{s}$ and k, respectively.

After the training phase, the adversary is able to perform the fault analysis in the attack phase with the attack set $\mathcal{S}_A$, which can be described as:

$$\mathcal{S}_A : \{(\vec{s_i}) | 1 \leq i \leq N_A\}. \tag{8}$$

Unlike the training set, the label (i.e. the key) of the attack set is unknown to the adversary. The goal of the attack phase is to recover the key from the dataset. Since the generative model has been constructed, the adversary needs to determine which one of the key candidates is the most likely correct key. This problem is normally solved via the maximum likelihood method, which calculates a likelihood score $d_{\mathcal{S}_A}[k]$ for every key candidate $k \in \mathcal{K}$ with the help of Bayes' theorem as follows:

$$\begin{aligned} d_{\mathcal{S}_A}[k] &= \prod_{i=1}^{N_A} \Pr[K = k | \vec{S} = \vec{s_i}] \\ &= \prod_{i=1}^{N_A} \frac{\Pr[\vec{S} = \vec{s_i} | K = k]}{f_{\vec{S}}(\vec{s_i})} \times f_K(k), \end{aligned} \tag{9}$$

where $f_{\vec{S}}$ and f_K denote the probability distribution function of $\vec{S}$ and K respectively. The key candidate with the highest $d_{\mathcal{S}_A}[k]$ can be selected as the correct key.

3.3 General Description of DLFA

Consistent with the previous analysis, we can give a general description of DLFA as shown in Fig. 4. To perform DLFA, the following stages are required for the adversary:

Stage 1: Injection. In the beginning, the adversary injects the fault into the S-box of the block cipher prior to the encryption. The faulty S-box will be used in the following encryption. The fault does not only affect a single output of the S-box, but all outputs of the S-box in all rounds. However, we only focus on the effect of the fault in the last round.

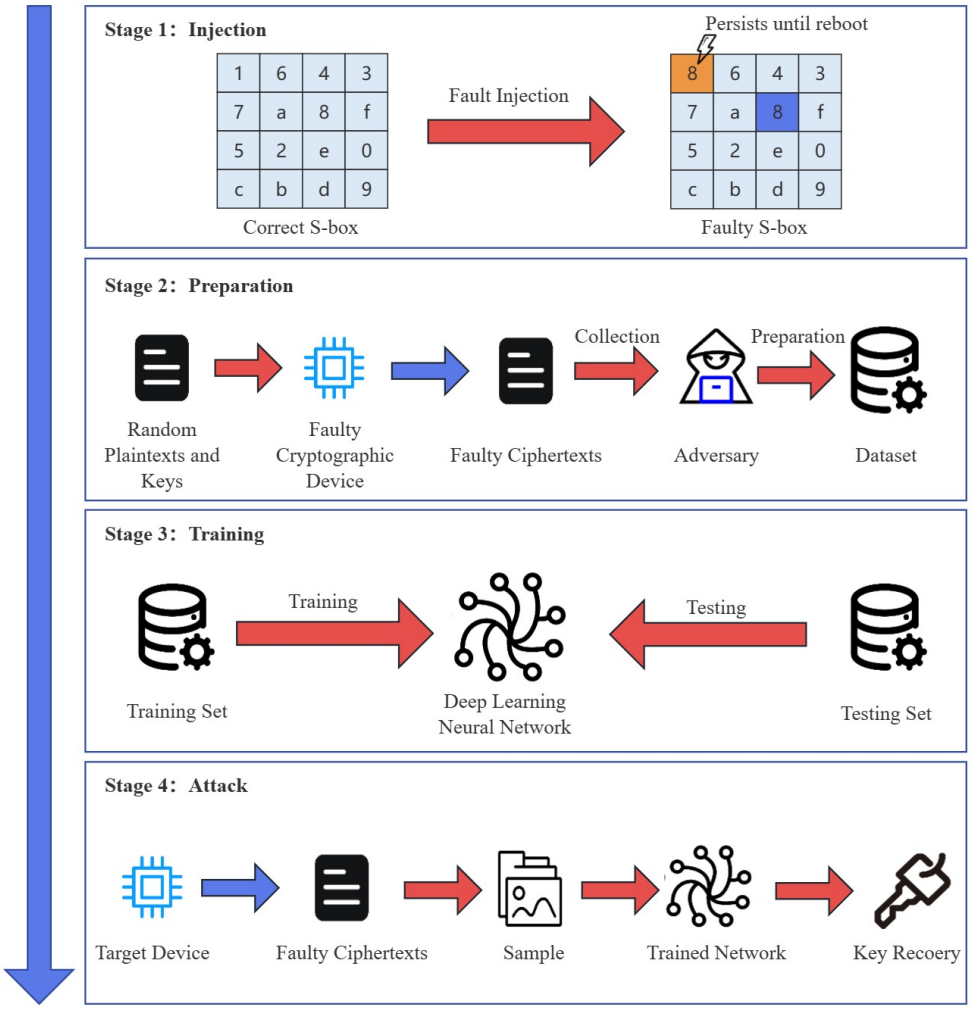


Figure 4: Overview of DLFA.

Similar to TA, we assume that the adversary possesses the capability to manipulate the target cryptographic device (or a cloned device) for encryption. Therefore, both the target and cloned device should be subjected to fault injection.

Stage 2: Preparation. When the injection is complete, the next task of the adversary is to prepare the training dataset. For one sample in the dataset, the adversary performs encryption several times with the same key and random plaintexts. The metadata are collected to compute the distribution of the output of the S-box in the last round as features and the key of the last round as the label.

Stage 3: Training. After that, the adversary needs to select the model and train the neural network offline. The parameters of the network are initialized with random values. These parameters are then updated to ensure that the predictions of the network are close to the labels. The loss function and the optimizer are usually used as hyper-parameters to aid training. The former measures the difference between the prediction and the label, and the latter determines the range of parameters to update based on the difference. The choice of the hyper-parameters will be discussed later, as it has a huge impact on the

performance of the network.

Stage 4: Attack. Eventually, the adversary is able to use the trained network to mount a practical attack. The attack set is prepared from the target device using the same way as that of the training set. One sample of the attack set is input into the network at a time, and the network computes a score vector containing the likelihood scores of all the key candidates. The candidate achieving the highest score is deemed to be the correct key. The adversary can evaluate the performance of the attack based on the metric, which is similar to the loss function but has no effect on the update of the parameters.

4 Evaluation

The experiments of DLFA on AES are presented in this section. Firstly, we introduce the experimental setups employed in our study. Subsequently, a concise overview of AES is provided. We investigate the impact of the choice of the hyper-parameters on the training phase and present the corresponding experimental results. Moreover, we evaluate the superiority of DLFA compared to state-of-the-art FA methods.

4.1 Experimental Setups

The experiments are implemented on an ordinary computer with the following configuration: IntelCore i5-12400 CPU at 3.20 GHz, 32-GB DDR5 memory, 1-TB Samsung SSD-980-PRO drive and Nvidia GeForce RTX 3060 GPU. The development language is Python (version 3.9.7) and the development environment is Jupyter Notebook (version 6.4.5).

Our implementations of the deep learning neural network are developed with Keras library (version 2.9.0) as the frontend and Tensorflow library (version 2.9.1) as the backend. CUDA (version 11.7) is used to accelerate the training process. The entire computation of the tuning process takes approximately 300 hours.

The fault injection in our experiments is achieved through the utilization of hardware trojan (HT). For a more comprehensive understanding of HT, we refer the reader to [PP23]. We assume that the persistent faults are injected into the S-box during the outsourcing fabrication process of the cryptographic circuit.

For simplicity, a serial cryptographic implementation is considered in our experiments. The initial size of dataset is 50000, where the training set size is 45000 and the attack set size is 5000.

4.2 AES Block Cipher

AES is a widely used block cipher, which was published by NIST in 2001 as a standard for symmetric encryption. It comes with a block size of 128 bits and three standard key sizes of 128, 192 and 256 bits. In this paper, we focus on the 128-bit version AES-128, which consists of 10 rounds.

The round function of AES-128 operates on a 4×4-byte matrix (i.e. state matrix) and includes four byte-oriented operations: AddRoundKey (AK), SubBytes (SB), ShiftRows (SR) and MixColumns (MC). AK combines the state with the round key via bitwise exclusive-or. SB substitutes the state byte by byte with the S-box. SR cyclically shifts each row of the state by a different offset. MC mixes each column of the state by multiplying it with a fixed matrix. Note that the MC operation is missing in the last round based on the specification. Moreover, a KeyExpansions operation is applied to generate round keys from the master key using a key generation schedule.

4.3 Choice of the Hyper-Parameters

The discussion of tuning the hyper-parameters can be divided into two parts. We first focus on the parameters that define the network architecture (i.e. architecture parameters). Then the parameters that affect the training phase (i.e. training parameters) are discussed in the second part.

In the experiments of MLP model, three parameters are selected to characterize the architecture of MLP: the number of layers, the number of units in each layer and the activation function. The parameterization of an MLP architecture can be described as $\text{MLP}_{arch}(n_{layers}, n_{units}, function)$. In the same way, we select the number of epochs, the batch size and the optimizer as the three training parameters and use $\text{MLP}_{train}(n_{epochs}, batch\_size, optimizer)$ to denote the parameterization of a training procedure.

The strategy of selecting the hyper-parameters for CNN model is inherited from the research in MLP context for fair comparison. However, the number of hyper-parameters of CNN model is too large for an exhaustive test of all the possible configurations. Hence, some representative parameters are selected to be tuned and the others are set up utilizing the work of Benadjila et al. in [BPS+20]. We consider a convolutional layer followed by a pooling layer as a block. Then the number of blocks, the number of fully-connected layers and the activation functions are selected as architecture parameters, which can be described as $\text{CNN}_{arch}(n_{blocks}, n_{FC\_layers}, function)$. Moreover, we fix the optimizer to Adam and select the number of epochs and the batch size as training parameters with the description of $\text{CNN}_{train}(n_{epochs}, batch\_size)$.

4.4 Evaluation Metrics

To evaluate the performance of our attack, we choose guessing entropy (GE) [SMY09] as the metric in this paper. GE is a widely used metric in SCA, which can give information about the trend of the attack and the size of data required for a successful attack. Compared with traditional deep learning metrics like success rate (SR), GE can give a fairer evaluation of the attack as it assesses the attack by the mean rank of the secret key, which gives a more objective assessment of the failed attack.

As mentioned above, for an attack set $\mathcal{S}_A$ of size N_T with the correct key k^*, the adversary is able to calculate the likelihood score $d_{\mathcal{S}_A}[k]$ for every key candidate k. Then the adversary can sort the scores from the largest to the smallest to get a rank vector. GE calculates the mean position of the correct key k^* in the rank vector, which can be described as:

$$\mathbf{GE} = \mathop{\mathbb{E}}_{\mathcal{S}_A}[|\{k \in \mathcal{K} | d_{\mathcal{S}_A}[k] > d_{\mathcal{S}_A}[k^*]\}|] \tag{10}$$

4.5 Experiments of MLP Model

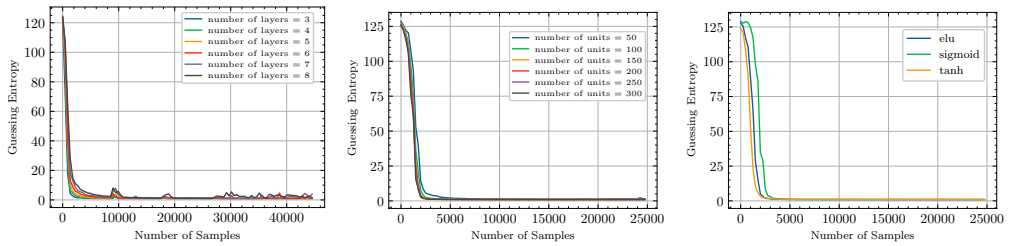
For the sake of fairness, we tune the architecture parameters with the fixed training parameters $\text{MLP}_{train}(40, 150, Adam)$, which denotes 40 epochs, batch size 150 and the Adam optimizer. Then in the second part of the experiment, different training parameters are tested with the tuned architecture parameters.

Choice of Architecture Parameters: We start the experiment with an evaluation of the number of layers. We fix the number of units at 200 and select the eLU as the activation function. Then MLP model is trained with different $n_{layers} \in \{3, \cdots 8\}$. As shown in Fig. 5(a), the 3-layer MLP has the lowest GE.

The number of units in each layer is then evaluated. We train a 3-layer MLP model with the eLU activation function and different $n_{units} \in \{50, 100, \cdots, 250, 300\}$. As Fig. 5(b) shows, our attack has a better performance with the increase of n_{units}. However, the

larger value of n_{units} will lead to higher computational complexity of the model. Taking the trade-off between efficiency and complexity into account, a compromise of n_{units} is 150.

The last architecture parameter we tune is the activation function. We select three activation functions as candidates: sigmoid, tanh and eLU. The first two are common deep learning activation functions and the last one is a modified version of ReLU, which has achieved remarkable performance in the field of image recognition. Experimental results depicted in Fig. 5(c) indicate that all three candidates exhibit strong performance, with tanh exhibiting a slight advantage over the other two activation functions.



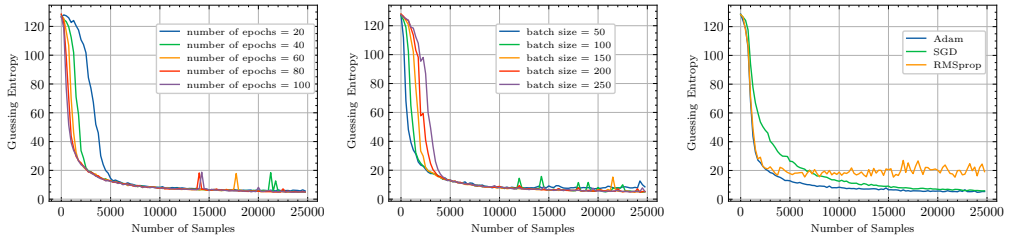
(a) Experiments of n_{layers}. (b) Experiments of n_{units}. (c) Experiments of activation function.

Figure 5: GE of $\text{MLP}_{arch}(n_{layers}, n_{units}, function)$ trained with $\text{MLP}_{train}(40, 150, Adam)$ for different n_{layers}, n_{FC_layers} and activation function.

Choice of Training Parameters: When the choice of architecture parameters is finished, the training parameters are tuned with the determined architecture parameters $\text{MLP}_{arch}(3, 150, tanh)$. Initially, we fix the batch size at 150 and the optimizer at Adam to select the best value for the number of epochs. Based on the experimental results in Fig. 6(a), we choose $n_{epochs} = 40$ as a good trade-off between efficiency and computation time.

The next training parameter we tune is the batch size. As shown in Fig. 6(b), 100 is a sound choice of batch size due to its efficiency in small data size and stability in large data size.

Finally, we evaluate the effect of the optimizer on the performance of the neural network. We provide three options: RMSprop, SGD and Adam. The learning rate for the first two is fixed to 10^{-5} and the last one is an adaptive optimizer that can dynamically adjust the learning rate by itself. The experimental results in Fig. 6(c) show that the choice of optimizer has a significant effect on GE. We manage to obtain good results with the Adam optimizer.



(a) Experiments of n_{epochs}. (b) Experiments of batch size. (c) Experiments of optimizer.

Figure 6: GE of $\text{MLP}_{train}(n_{epochs}, batch\_size, optimizer)$ trained with $\text{MLP}_{arch}(3, 150, tanh)$ for different n_{epochs}, batch size and optimizer.

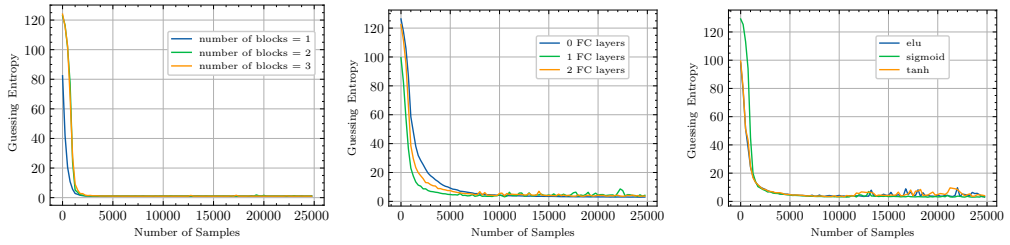
4.6 Experiments of CNN Model

Due to the large search space of the hyper-parameters of CNN model, the goal of the following experiments is not to find the optimal value, but rather to obtain a better value that leads to satisfactory results. Before starting the experiment, some initial configurations should be set up following the experience of the predecessors. We initially fix the kernel size of the convolutional layer at 11. Then the number of filters in the i-th convolutional layer is set to $64 \times 2^{i-1}$ with an upper limit of 512. We choose the average pooling as the pooling method and set the padding option to the same padding. Ultimately, the number of units in a fully-connected layer is fixed at 4096. We start the experiment with the architecture parameters $\text{CNN}_{arch}(1, 1, eLU)$ and the training parameters $\text{CNN}_{train}(40, 100)$.

Choice of Architecture Parameters: We first evaluate the impact of the number of blocks. As Fig. 7(a) shows, the efficiency of the model is dimly impacted by n_{blocks}. One block is sufficient for CNN model to mount a successful attack. Hence, we set the number of blocks to 1 and fix the number of filters in the convolutional layers at 64.

We then investigate the effect of the number of fully-connected layers. Unexpectedly, Fig. 7(b) shows that the network without fully-connected layers performs equally well. This fact can be explained by the functionality of the fully-connected layer. The fully-connected layer integrates and reduces the dimensionality of the characteristics extracted by the network. Since the dataset is lower dimensional and the network model is compact, the fully-connected layer is unnecessary for our experiments. However, the best results are obtained with one fully-connected layer. We recommend keeping the value $n_{FC\_layers} = 1$ to improve the performance of our CNN model.

Finally, the effect of the activation function is evaluated. Results given in Fig. 7(c) show that all the activation functions are suitable for our attack. We select eLU as it has less overhead than the other two functions.



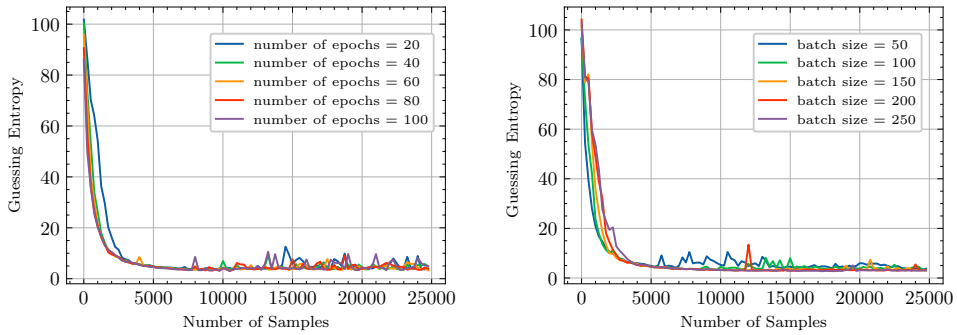
(a) Experiments of n_{blocks}. (b) Experiments of n_{FC_layers}. (c) Experiments of activation function.

Figure 7: GE of $\text{CNN}_{arch}(n_{blocks}, n_{FC\_layers}, function)$ trained with $\text{CNN}_{train}(40, 100)$ for different n_{blocks}, n_{FC_layers} and activation function.

Choice of Training Parameters: We first investigate the impact of the number of epochs on the model efficiency. The experimental results are plotted in Fig. 8(a). Similar to MLP model, the efficiency of the model increases when the number of epochs increases. However, the experiments need to be performed in a reasonable amount of time. We select 40 as an appropriate value of n_{epochs}.

Then we evaluate the performance of CNN model with different values of the batch size. Fig. 8(b) shows that each batch size can perform well with a large training set size. However, when the size of the training set is smaller than 5000, GE decreases more steeply for smaller batch size. Therefore, we recommend fixing the batch size to 50 to obtain decent global results.

The tuned hyper-parameters of MLP and CNN models are summarized in Tables 2 and 3. However, since the optimization of the hyper-parameters is not our goal, we believe



(a) Experiments of n_{epochs}.

(b) Experiments of batch size.

Figure 8: GE of $\text{CNN}_{train}(n_{epochs}, batch\_size)$ trained with $\text{CNN}_{arch}(1, 1, eLU)$ for different n_{epochs} and batch size.

that more appropriate parameters may be found to obtain better results in the following studies. Additionally, we recommend employing more sophisticated models to enhance training phase efficiency.

Table 2: Summary of the Tuned Hyper-Parameters of MLP Model

Type	Hyper-Parameters	Reference	Range	Choice
Architecture Parameters	Layers	n_{layer}	{3,4, ⋯ ,7,8}	3
	Units	n_{units}	{50,100, ⋯ ,250,300}	150
	Activation Function	-	eLU, Sigmoid, tanh	tanh
Training Parameters	Epochs	n_{epochs}	{20 ,40, ⋯ ,80,100}	40
	Batch Size	-	{50,100, ⋯ ,200,250}	100
	Optimizer	-	Adam, SGD, RMSprop	Adam

Table 3: Summary of the Tuned Hyper-Parameters of CNN Model

Type	Hyper-Parameters	Reference	Range	Choice
Architecture Parameters	Blocks	n_{blocks}	{1,2,3}	1
	FC Layers	n_{FC_layers}	{0,1,2}	1
	Activation Function	-	eLU, Sigmoid, Tanh	eLU
Training Parameters	Epochs	n_{epochs}	{20,40, ⋯ ,80,100}	40
	Batch Size	-	{50,100, ⋯ ,200,250}	50

4.7 Comparisons with State-of-the-art Methods

After completing the offline training with tuned hyper-parameters, we evaluate the attack performance of DLFA by comparing it with some state-of-the-art FA methods. The experiments are repeated 1000 times to find the average number of ciphertexts required and the analysis time needed for key recovery. Note that the number of ciphertexts required in our attack refers to the number of ciphertexts utilized for generating the sample that is fed into the trained deep learning model. Specifically, we initially compare our attack approach to the persistent fault-based methods due to their similar chosen fault model. Three counterparts are selected for comparison: the original PFA [ZLZ+18], the improved

PFA with Maximum Likelihood Estimation [ZZJ+20], and the extended PFA with GPU acceleration [XZY+21]. The results are shown in Table 4, where N_c denotes the number of ciphertexts that are required for fully recovering the master key. We can see that DLFA can crack ciphers using the least ciphertexts and spending the least analysis time.

Table 4: Comparison with Persistent Fault-Based Methods

Method	Minimum N_c	Average N_c	Analysis Time(s)
PFA	1493	2273	10.48
PFA-MLE	926	1641	11.03
EPFA	896	970	8.64
DLFA	683	1488	0.12

Subsequently, DLFA is compared with SFA[PCNM15] and statistical PFA [ESP20] due to their shared utilization of SEI. Table 5 shows the results, where N_f denotes the number of faults the attack required. While SFA exhibits the lowest average N_c, it necessitates a significantly longer time, nearly 1 hour, due to the laborious fault injection process involving the large number of faults. In contrast, DLFA only requires one fault and achieves a significant trade-off between N_c and analysis time.

Table 5: Comparison with SFA and SPFA

Method	Average N_c	N_f	Analysis Time(s)
SFA	304	633	2649.83
SPFA	1643	16	66.22
DLFA	1488	1	0.12

Furthermore, we explore the comparison between DLFA and Fault Template Attacks [SBR+20], which is another prominent profiled fault attack constructed from fault activation and propagation. The strongest feature of FTA is that it can enable attacks in the middle round of a cipher without requiring any explicit access to the ciphertexts. However, as shown in Table 6, an extensive number of 4112 faults are required to recover the master key of AES, imposing a substantial burden on the adversary.

Table 6: Comparison with FTA

Method	N_f	Ciphertext Access	Middle Round Attack
FTA	4112	×	✓
DLFA	1	✓	×

5 Discussions

The applicability and efficiency of DLFA have been validated in section 4. In this section, we further expand our study by discussing the performance of machine learning techniques and investigating the impact of fault position and value.

5.1 Performance of Machine Learning Techniques

Given the validated success of applying deep learning techniques to FA key recovery, it is worthwhile to investigate whether simpler machine learning techniques can achieve comparable performance. The performance of two classical machine learning techniques, Support Vector Machine (SVM) and Random Forest, have been chosen for evaluation in the context of FA key recovery.

The experimental results indicate that, utilizing a dataset comprising 2000 ciphertexts, which is deemed sufficient for our DLFA method, the accuracy scores of SVM and Random Forest models are 0.008 and 0.601 respectively. When the number of ciphertexts increases to 4500, which is nearly three times greater than our proposed method's requirement, the Random Forest model demonstrates a satisfactory accuracy score of 0.847. However, the SVM model exhibits a limited ability to recover the key with an accuracy score of 0.054. The poor performance can be attributed primarily to the limited applicability of SVM, which is mainly designed for binary classification problems. However, its effectiveness diminishes when applied to multi-class classification tasks like ours.

5.2 Impact of the Position and Value of the Fault

The position and value of the fault are the two requirements of the original PFA. In DLFA, we ignore these requirements and assume that the adversary has no knowledge of these information. However, the position and value of the fault can affect our attack in another way. The randomization of the position and value implies a variety of faults in the dataset. Given a fixed-size dataset, the more types of faults there are in the dataset, the less information each fault provides for the training of the neural network.

To evaluate our attack in stages, we prepare four different datasets representing four levels of restriction on the position and value of the fault. The first dataset, named FPFV dataset, assumes that the position and value of the fault are fixed. That is, all samples in the dataset are collected under a fixed fault condition, which aligns with the underlying assumption shared by other persistent fault-based analyses. The FPRV and RPFV datasets are then prepared for the case where the position or value of the fault is random, respectively. Finally, in the RPRV dataset, we assume that the position and value of the fault are random, which is the highest level of restriction.

The experiments demonstrate that MLP and CNN models can mount successful attacks as expected on the FPFV, FPRV and RPFV datasets. GEs of the three experiments eventually approximately converge to 1. Unfortunately, these models perform poorly on the RPRV dataset. The experimental results indicate that the oscillation of GE converges around 128, implying a lack of learning capability in the network when exposed to the given dataset. We attribute the failure to the increased randomness of the dataset. The randomness can be roughly quantified in terms of the possibility of the fault. Reviewing the configuration of the dataset, there is only one possibility of the fault in the FPFV dataset as the position and value of the fault are fixed. According to the characteristics of AES, the faults of the S-box have 256 random values and 256 random positions. Thus, there are 256 possible faults in the FPRV and RPFV datasets and 65536 possible faults in the RPRV dataset.

To validate our point of view, we extend our attack to the lightweight SPN-based cipher PRESENT[BKL+07] due to its pint-sized S-box. We perform the experiments of MLP and CNN models with the hyper-parameters tuned in the previous section. The experimental results demonstrate that both models exhibit satisfactory performance on the RPRV dataset. GE eventually approximately converges to 3, which implies the existence of an average of three possible key candidates. The adversary can easily recover the key with a brute-force search.

For future work, it might be interesting to enhance the performance of DLFA on

the RPRV dataset on AES. This issue bears resemblance to the cross-device scenario encountered in SCA, where training and attack data are acquired from disparate devices.

6 Conclusion

In this paper, we conduct a thorough study on the application of deep learning-based key recovery in the field of fault analysis. We propose DLFA as an innovative FA method and mount successful attacks with MLP and CNN models. In particular, we discuss the choice of hyper-parameters and present the tuned parameterization options as a reference for the design of new models. The comparisons with state-of-the-art methods demonstrate the superiority of DLFA. Given that current deep learning theory does not yet provide a clear foundation for such analysis, we believe that establishing a methodology is an essential initial step that paves the way for further research in this domain.

References

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156, pages 16–29. Springer, 2004.

[BDG+10] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 307–322. USENIX Association, 2010.

[BHJ+18] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 2204–2206. ACM, 2018.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727, pages 450–466. Springer, 2007.

[BL12] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient template attacks based on probabilistic multi-class support vector machines. In *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, volume 7771, pages 263–276. Springer, 2012.

[BMM00] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880, pages 131–146. Springer, 2000.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.*, 10(2):163–188, 2020.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294, pages 513–525. Springer, 1997.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529, pages 45–68. Springer, 2017.

[CLFT14] Franck Courbon, Philippe Loubet-Moundi, Jacques J. A. Fournier, and Assia Tria. Adjusting laser injections for fully controlled faults. In *Constructive Side-Channel Analysis and Secure Design - 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, volume 8622, pages 229–242. Springer, 2014.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523, pages 13–28. Springer, 2002.

[DEK+18] Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):547–572, 2018.

[ESP20] Susanne Engels, Falk Schellenberg, and Christof Paar. SPFA: SFA on multiple persistent faults. In *17th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2020, Milan, Italy, September 13, 2020*, pages 49–56. IEEE, 2020.

[FJLT13] Thomas Fuhr, Éliane Jaulmes, Victor Lomné, and Adrian Thillard. Fault attacks on AES with faulty ciphertexts only. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 108–118. IEEE Computer Society, 2013.

[GA03] Sudhakar Govindavajhala and Andrew W. Appel. Using memory errors to attack a virtual machine. In *2003 IEEE Symposium on Security and Privacy (S&P 2003), 11-14 May 2003, Berkeley, CA, USA*, pages 154–165. IEEE Computer Society, 2003.

[GHO15] Richard Gilmore, Neil Hanley, and Máire O'Neill. Neural network based attack on a masked implementation of AES. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015*, pages 106–111. IEEE Computer Society, 2015.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162, pages 251–261. Springer, 2001.

[HGM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptogr. Eng.*, 1(4):293–302, 2011.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666, pages 388–397. Springer, 1999.

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):148–179, 2019.

[LBM15] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked AES - reaching the limit of side-channel attacks with a learning model. *J. Cryptogr. Eng.*, 5(2):123–139, 2015.

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064, pages 20–33. Springer, 2015.

[LWLX17] Yannan Liu, Lingxiao Wei, Bo Luo, and Qiang Xu. Fault injection attack on deep neural network. In *2017 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2017, Irvine, CA, USA, November 13-16, 2017*, pages 131–138. IEEE, 2017.

[Man02] Stefan Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In *Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers*, volume 2587, pages 343–358. Springer, 2002.

[MDM16] Zdenek Martinasek, Petr Dzurenda, and Lukas Malina. Profiling power analysis attack based on MLP in DPA contest V4.2. In *39th International Conference on Telecommunications and Signal Processing, TSP 2016, Vienna, Austria, June 27-29, 2016*, pages 223–226. IEEE, 2016.

[MZVT15] Zdenek Martinasek, Ondrej Zapletal, Kamil Vrba, and Krisztina Trasy. Power analysis attack based on the MLP in DPA contest v4. In *38th International Conference on Telecommunications and Signal Processing, TSP 2015, Prague, Czech Republic, July 9-11, 2015*, pages 154–158. IEEE, 2015.

[PCNM15] Sikhar Patranabis, Abhishek Chakraborty, Phuong Ha Nguyen, and Debdeep Mukhopadhyay. A biased fault attack on the time redundancy countermeasure for AES. In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 2015.

[PP23] Tiago D. Perez and Samuel Pagliarini. Hardware trojan insertion in finalized layouts: From methodology to a silicon demonstration. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 42(7):2094–2107, 2023.

[PSK+18] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In *Security, Privacy, and Applied Cryptography*

Engineering - 8th International Conference, SPACE 2018, Kanpur, India, December 15-19, 2018, Proceedings, volume 11348, pages 157–176. Springer, 2018.

[Riv91] Ronald L. Rivest. Cryptography and machine learning. In *Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings*, volume 739, pages 427–439. Springer, 1991.

[SAB+20] Sayandeep Saha, Manaar Alam, Arnab Bag, Debdeep Mukhopadhyay, and Pallab Dasgupta. Leakage assessment in fault attacks: A deep learning perspective. *IACR Cryptol. ePrint Arch.*, page 306, 2020.

[SBR+20] Sayandeep Saha, Arnab Bag, Debapriya Basu Roy, Sikhar Patranabis, and Debdeep Mukhopadhyay. Fault template attacks on block ciphers exploiting fault propagation. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 612–643. Springer, 2020.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479, pages 443–461. Springer, 2009.

[Tim19] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):107–131, 2019.

[XZY+21] Guorui Xu, Fan Zhang, Bolin Yang, Xinjie Zhao, Wei He, and Kui Ren. Pushing the limit of PFA: enhanced persistent fault analysis on block ciphers. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 40(6):1102–1116, 2021.

[ZLZ+18] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. Persistent fault analysis on block ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):150–172, 2018.

[ZZJ+20] Fan Zhang, Yiran Zhang, Huilong Jiang, Xiang Zhu, Shivam Bhasin, Xinjie Zhao, Zhe Liu, Dawu Gu, and Kui Ren. Persistent fault attack in practice. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):172–195, 2020.