Convolutional Neural Networks for Profiled Side-Channel Analysis

Shourong HOU, Yujie ZHOU, Hongming LIU

Dept. of Electronic Engineering, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai, China

ruihou@sjtu.edu.cn, mlscagroup@163.com, hongming.liu@aisinochip.com

Submitted May 2, 2019 / Accepted August 1, 2019

Abstract. *Recent studies have shown that deep learning algorithms are very effective for evaluating the security of embedded systems. The deep learning technique represented by Convolutional Neural Networks (CNNs) has proven to be a promising paradigm in the profiled side-channel analysis attacks. In this paper, we first proposed a novel CNNs architecture called DeepSCA. Considering that this work may be reproduced by other researchers, we conduct all experiments on the public ASCAD dataset, which provides electromagnetic traces of a masked 128-bit AES implementation. Our work confirms that DeepSCA significantly reduces the number of side-channel traces required to perform successful attacks on highly desynchronized datasets, which even outperforms the published optimized CNNs model. Additionally, we find that DeepSCA pre-trained from the synchronous traces works well in presence of desynchronization or jittering after a slight fine-tuning.*

Keywords

Side-channel analysis, deep learning, convolutional neural networks, DeepSCA

1. Introduction

Side-channel analysis (SCA) is a powerful cryptanalytic attack that exploits the information leaked from the physical implementations of cryptographic algorithms to break the secret key [1]. The profiled SCA attacks based on supervised learning techniques have recently received significant attention in the SCA community. Researchers in the security field explore different machine learning techniques to assess their effectiveness in the SCA context. As a consequence, there are several papers on the intersection of machine learning techniques and profiled SCA attacks [2–9]. While different scenarios usually require different machine learning techniques, almost all work demonstrates that Support Vector Machines (SVM) and Random Forests (RF) are good baseline algorithms for profiled SCA attacks.

In recent years, deep learning has begun to demonstrate its powerful efficiency in profiled SCA attacks because it almost perfectly approximates arbitrary functions. Several studies have already investigated the performance of deep neural networks in profiled SCA attacks. Maghrebi et al. [10] first compare the SCA-efficiency of deep learning and machine learning in terms of the number of side-channel traces. The work [11] evaluates the performance of convolutional neural networks (CNNs) in scenarios where power consumption traces are misaligned due to countermeasures or hardware-related effects (clock jitter). Their research shows that CNNs combined with data augmentation technique can effectively suppress the misalignment effects. Prouff et al. [12] give an empirical solution to the question of the choice of hyper-parameters for CNNs and multi-layer perceptron (MLP), further confirming the power of deep learning applied to profiled SCA attacks. The other important contribution is the release of the public ASCAD database, which provides side-channel traces of a masked 128-bit AES implementation. The ASCAD dataset makes it easy for researchers to improve existing models or compare new deep neural network architectures.

Based on the CNNs model proposed in the article [12], this paper introduces several mechanisms to achieve high performance in profiled SCA attacks. In particular, we evaluate the effect of the batch normalization layer on improving the SCA-efficiency of CNNs and we present that the pre-trained CNNs model can also help us make more appropriate decisions. Briefly, the main contributions of this paper are as follows:

- We propose a deep learning architecture called DeepSCA that can reach higher SCA-efficiency than the CNNs architecture designed in [12].
- The batch normalization layer is necessary to get good performances in our case. Furthermore, batch normalization acts as a regularization in deep neural networks, which makes it more robust and easier to generalize.
- After a slight fine-tuning, DeepSCA pre-trained from the synchronous side-channel traces works well in presence of desynchronization or jittering.

The rest of this article is organized as follows: Section 2 introduces the basic knowledge of profiled SCA attacks and deep learning. In Sec. 3, we provide the details about datasets used in this article, the DeepSCA architecture and evaluation metrics. The designed experiments and results obtained are presented in Sec. 4. We conclude this article in Sec. 5.

2. Preliminaries

This section briefly introduces profiled SCA attacks and deep learning algorithms, especially convolutional neural networks.

2.1 The Profiled SCA Attacks

For profiled SCA attacks, the adversary is assumed to have a pair of identical devices: a profiling device and a target device. In this scenario, the target device runs a symmetric cryptographic algorithm with the fixed secret key. The attacker has access to control the input and the key of the profiling device, so he has the ability to characterize the leaked information very precisely by manipulating some statistical techniques. The profiled SCA attacks are performed in two steps: the profiling (training) phase and the attack (test) phase.

In the profiling phase, a customized leakage model can then be built by using the key-related information of the profiling device. The attacker calculates an estimation e_k of the probability distribution function (pdf) for each possible $k \in \mathrm{K}$ as follows:

$$e_k = \zeta[T = t|(P, K) = (p, k)] \tag{1}$$

where the dataset of side-channel traces is denoted by T and the estimation $\hat{e}_k$ is processed from a profiling set $D_p = \{t_i, p_i, k_i\},\ i = 1, ..., S_p$. The side-channel trace t_i is acquired under known plaintext p_i and secret key k_i of a cryptographic algorithm.

During the attack step, the attack set $D_a = \{t_i, p_i\}$, $i = 1, ..., S_a$ is used to test and the correct secret key is fixed but unknown. The final goal is to obtain the latter secret key k^* by using side-channel traces of the attack set. For this purpose, an attacker who builds the statistical model must decide which of the pdf estimations $\hat{e}_k,\ k \in \mathrm{K}$, is the most likely for the attack set. As we all know, maximum likelihood estimation is the most efficient way to make such decisions about the nature of the data distribution in real-world scenarios. This strategy is equivalent to estimating the following log-likelihood $L[k]$ for each key candidate $k \in K$:

$$\log L[k] = \sum_{i=1}^{S_a} \log \zeta[(P, K) = (p_i, k)|T = t_i] \tag{2}$$

where $L[k]$ is the log-likelihood probability corresponding to key candidate k. The most appropriate model for side-channel traces collected from the target device is used to reveal the correct key that maximizes $L[k]$.

2.2 Deep Learning

Deep learning is a branch of machine learning, which has been applied to image classification, speech recognition, and other fields [13]. Machine learning usually requires manual feature engineering while CNNs learn the automatic features directly from raw data [14]. Furthermore, the features extracted by convolutional layer are independent of their position in the data and dense layers can identify the features related to the labeled traces. Therefore, convolutional neural networks should be robust to the jitter effects from unstable clock domains or even desynchronization.

The basic structure of CNNs is a convolutional layer *Conv* followed by a nonlinear activation layer *Act*. The convolutional layer extracts local information from the input through a bunch of filters and the subsequent nonlinearity activation function can implement more complex learning strategies. After a series of $(Act \circ Conv)$ modules, a pooling layer *Pool* is usually added to reduce the input size of the next module: $Pool \circ [Act \circ Conv]^{l_2}$. This module is stacked in the neural network until a reasonable size of the output is obtained. Subsequently, several fully-connected dense layers are used to map the learned features to the label space of the samples. The well-known translation-invariant feature is implemented through local connections and shared weights, as well as a specific form of pooling operations (e.g, MaxPooling, AveragePooling, etc.). In addition, compared with fully-connected networks with the same number of hidden layers, convolutional neural networks greatly reduces the size of parameters so that it works well in practice. In brief, the general CNNs architecture can be characterized as follows:

$$s \circ [FC]^{l_1} \circ \left[Pool \circ [Act \circ Conv]^{l_2}\right]^{l_3} \tag{3}$$

where s denotes a softmax layer and FC represents a fully-connected layer.

In the article [15], batch normalization is introduced to remedy the defect of internal covariate shift in deep neural networks, ultimately allowing for higher learning rates. The input data distribution of a machine learning system varies when the covariate shift happens. The internal covariate shifting means that this change occurs at the input of neurons inside hidden layers of deep neural networks. We can use a higher learning rate and don't care about initialization due to batch normalization. The superior performance of batch normalization is demonstrated by comparing the VGG networks with and without batch normalization in [16]. Thus, batch normalization acts as a regularization that eliminates the dependency on dropout in the neural network.

For the AES algorithm, the attacker can adopt a divide-and-conquer strategy to break only one secret key byte at a time. Generally, deep learning classifies the side-channel traces based on the labels derived from the assumed leakage model. However, an attacker cannot reveal the secret key directly from the classification results of side-channel traces, so the key enumeration process is necessary. The output of

the softmax layer in deep neural networks is the classification probability vector for one side-channel trace. These probabilities are then used to calculate the log-likelihood of each key byte candidate through (2).

3. Methodologies

First of all, it is important to understand the overall framework for applying deep learning to profiled SCA attacks. The scenario described here is that the input and key are known, which is a supervised learning problem. The input and key are used to label each side-channel trace according to the hypothetical leakage model. The dataset of side-channel traces is then split into a training set and a test set. Ideally, the testing set is measured from another identical device that contains an unknown key. The training phase consumes a lot of computing resources while the test phase is usually very fast.

3.1 DeepSCA

As far as we know, the most popular CNNs architecture reported in the field of SCA attacks can be seen as a variant of the VGG network [17] under a specific design strategy. The VGG network was originally designed for image classification. Therefore, the input signal is a two-dimensional image, while the spatial dimension of our side-channel traces is only one. Fortunately, convolution and pooling operations are also feasible for one-dimensional signals. Based on literature and experimental results, we created hyper-parameter constraints and then selected the final CNNs architecture. Additionally, the hyper-parameter values of the CNNs architecture can be obtained by random search. The hyper-parameters that need to be optimized are: the number of convolutional/pooling layers, the number of epochs, the learning rate, batch normalization, batch size, dropout, convolutional activation functions, stochastic gradient descent, the kernel size of a convolutional filter, stride, padding, and the number of neurons in each fully-connected layer.

Our deep neural network is stacked by 6 convolutional/pooling layers and two fully-connected layers. All convolutional layers use filters with kernel size 11 and stride 1, which create many activation mappings for each layer. The activation function of convolutional layers is Leaky Rectified Linear Unit (Leaky ReLU). The Leaky ReLU fixes the problem of dying ReLU since it doesn't have zero-slope parts. Therefore, it is feasible to make the network learn more and more abstract representations from the training set by stacking many convolutional/pooling layers [18]. The form of padding is set to the same, which means that the output size is the same as the original input for each convolutional layer. The number of filters per layer follows a geometric progression (where the initial value is 32 and the ratio is 2) until the maximum value 512. The average pooling is configured as the kernel of size 2 and stride 2. Two fully-connected layers have 2048 units respectively and the softmax function is used to obtain a probability distribution from the input in the classification layer. Note that dropout is not used in the convolutional and fully-connected layers.

Besides, batch normalization is a widely used technique to train deep neural networks faster and more stable. Here, we only applied the batch normalization for the first and last convolutional layers to standardize the output values after the Leaky ReLU activation function. In order to facilitate writing and comparison, we denoted this deep neural network architecture as DeepSCA. Figure 1 provides a visual DeepSCA network architecture and Table 1 lists the following hyper-parameter values of DeepSCA: the input and output shapes, filter size, sampling and activation function. Compared with the CNN_best architecture in [12], our DeepSCA is added one convolutional/pooling layers and batch normalization operation is performed after the first and last convolutional layers. In order to reduce the number of trainable parameters, we reduced the number of neurons in the last two fully-connected layers from 4096 to 2048.

There are a large number of hyper-parameters to consider when training a deep neural network for profiled SCA attacks. Due to the limited space of the article, we will not discuss these hyper-parameters one by one in the next section. More detailed descriptions of the choice of hyper-parameters and rationality analysis are given in [12].

3.2 Evaluation Metrics

The common evaluation metrics are accuracy and loss in the machine learning community. The training accuracy refers to the successful classification rate over the training set, and training loss is the error rate over the training set. As the training accuracy continues to increase and the training loss decreases after many epochs, we know that the deep neural network is learning from the training set. The test accuracy indicates how good the trained model predicts side-channel traces of the test set that have never been learned before. However, the accuracy is not adequate in the key recovery phase for profiled SCA attacks.

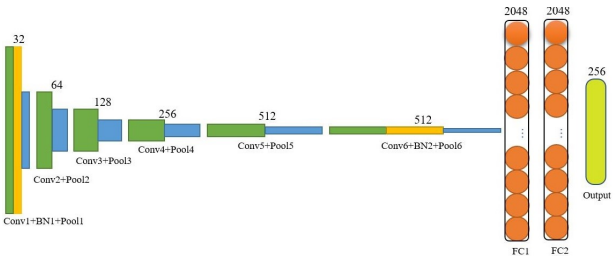

Fig. 1. The DeepSCA architecture diagram. The 1-dimensional convolutional layer is represented as a green rectangular block, the orange blocks indicate batch normalization layers, and the blue blocks indicate average pooling layers. The output of the final average pooling layer is flattened into feature vectors of a single dimension. The fully-connected layers are used to identify features for classification and the softmax layer gives the categories to which the input belongs. The number on the top of each block indicates the number of filters or neurons in the corresponding layer.

Layer	Output Shape	Filter Size	Sub-Sampling	Activation
Conv1	700×32	$1 \times 32 \times 11$	-	Leaky ReLU
BN1	700×32	-	-	-
Pool1	350×32	-	2	-
Conv2	350×64	$1 \times 64 \times 11$	-	Leaky ReLU
Pool2	175×64	-	2	-
Conv3	175×128	$1 \times 128 \times 11$	-	Leaky ReLU
Pool3	87×128	-	2	-
Conv4	87×256	$1 \times 256 \times 11$	-	Leaky ReLU
Pool4	43×256	-	2	-
Conv5	43×512	$1 \times 512 \times 11$	-	Leaky ReLU
Pool5	21×512	-	2	-
Conv6	21×512	$1 \times 512 \times 11$	-	Leaky ReLU
BN2	21×512	-	-	-
Pool6	10×512	-	2	-
FC1	2048	-	-	-
FC2	2048	-		-
Output	256	-	-	Softmax

Tab. 1. The DeepSCA architecture parameters.

The rank function is widely used to evaluate the effects of attacks in multi-trace experimental scenarios. When using maximum likelihood estimation to recover the secret key, we pay more attention to the final probability output of each side-channel trace. The output probability of each key candidate is considered as the score in rank function. The rank function then ranks the scores of all key candidates. In a successful attack, the rank function of the correct secret key k^* is zero. The rank function is defined as follows:

$$\text{rank}(\hat{e}_k, S_a) = \left|\left\{ k_i \in K | \, \text{score}(k_i) > \text{score}(k^*)\right\}\right| \qquad (4)$$

where $\text{score}(k)$ denotes the score of a key candidate k_i, $i = 0, 1, 2, ..., 255$.

More precisely, we perform a 10-fold cross validation in the following attack experiments to get a better measure of rank function. The average rank of the correct key is denoted as mean rank function. Thus, the mean rank function is used to be an evaluation metric for assessing the performance of profiled SCA attacks.

4. Experiments

We use Keras with TensorFlow backend [19], [20] to construct the DeepSCA architecture. The training phase is performed on a desktop computer equipped with an Intel Xeon E5-2698 v4 @2.2 GHz processor, 16 GB of RAM and an NVIDIA TESLA P100 card. In all experiments, we use a set of randomly selected profiled traces from ASCAD datasets with 256 classes.

4.1 ASCAD Dataset

The DeepSCA architecture is tested on the ASCAD dataset, which is available at https://github.com/ANSSI-FR/ASCAD. The target platform is an 8-bit AVR microcontroller implemented a masked AES-128 algorithm and side-channel traces are captured by using electromagnetic emanation. The dataset follows the format of the MNIST dataset, of which 50000 side-channel traces were used for training and the remaining 10000 for the test phase. The hypothetical leakage model is the unprotected S-Box output byte, i.e. $SBox[p_i \oplus k_i]$. The pre-selected window of 700 relevant samples per side-channel trace corresponds to the masked S-Box. Our attack experiments performed in this paper only aim at the output of the third S-Box processed in the first round of AES.

For the ASCAD dataset, a parametrized desynchronization is added to side-channel traces to test the SCA-efficiency of different attack algorithms against jittering. A desynchronization amount of $Desync_{\max}$ indicates that each side-channel trace has been randomly shifted by $\delta \in [0, Desync_{\max}]$ points to the left. A new side-channel trace is generated from the original trace after the desynchronization. Interested readers can get more useful information about the ASCAD dataset in [12]. Theoretically, the CNNs architecture is naturally able to extract information from desynchronized side-channel traces thanks to its translation-invariance property.

4.2 Results

In fact, we do not make a comprehensive analysis of the impact of hyper-parameters on the SCA-efficiency of DeepSCA in this paper. The main reason for this decision is that Prouff et al. [12] discusses in detail the question of the choice of the hyper-parameters for convolutional neural networks. Consequently, this article does not discuss the effects of hyper-parameters such as the learning rate, kernel size, batch size, padding, the optimization of loss function, etc. We explored the impact of the number of epochs, batch normalization layer, and the pre-training mechanism on improving the SCA-efficiency of DeepSCA. At the same time, our experiments compared the SCA-efficiency of DeepSCA and CNN_best.

The first experiment was obtained by training DeepSCA with a different number of epochs. The epoch value is a hyper-parameter, which defines the number times a learning algorithm traverses the whole training set. In an epoch, each sample in the training set is likely to be used to update the internal parameters of the model. By increasing the number of epochs, DeepSCA obtains a set of appropriate parameters, thereby improving the ability to fit side-channel traces of the training set. Figures 2, 3 and 4 report respectively the mean rank of DeepSCA as a function of the number of epochs when the amount of desynchronization is 0, 50 and 100. We can see that the SCA-efficiency of DeepSCA also continues to increase as the number of epochs increases in terms of the mean rank. These results imply that there is no overfitting phenomenon (relatively to our rank function) when the number of epochs increases.

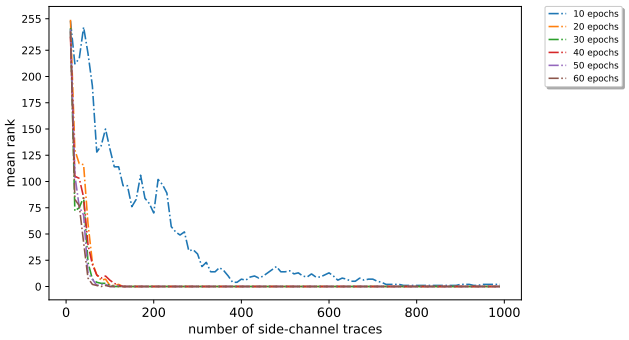

Fig. 2. Mean ranks of DeepSCA with different numbers of epochs when the desynchronization amount is 0.

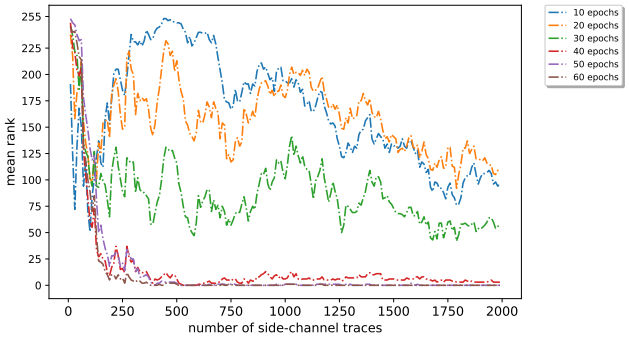

Fig. 3. Mean ranks of DeepSCA with different numbers of epochs when the desynchronization amount is 50.

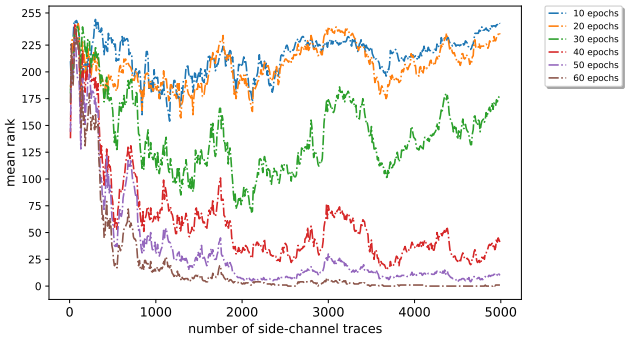

Fig. 4. Mean ranks of DeepSCA with different numbers of epochs when the desynchronization amount is 100.

It can be seen that the mean rank is also close to 0 as long as the number of epochs is sufficient in the training phase. When the maximal desynchronization value increases, we need more time to train parameters of DeepSCA and more side-channel traces in the test (key recovery) phase. With the increase of desynchronization, the number of epochs has more and more influence on the SCA-efficiency of DeepSCA. An epoch value of 50 is sufficient for our datasets in view of the mean rank, but it does not yield to the satisfactory performance when side-channel traces are desynchronized. For the sake of robustness and acceptable training time, the number of epochs is set to 60 in this case. The DeepSCA acquires less than 100 side-channel traces when the maximal desynchronization value $Desync_{\max}$ is 0. However, for the case where the maximal desynchronization value is 100, the number of required traces is about 3000 when the mean rank is equal to 0. These results highlight the success of DeepSCA in the context of desynchronized traces, but DeepSCA still needs a lot of side-channel traces to recover the secret key in the test phase.

Figure 5 depicts the training loss varies with the number of epochs when the desynchronization value is 0, 50 or 100. The training loss is decreasing continuously as the number of epochs increases. Correspondingly, the SCA-efficiency of DeepSCA is also increasing from the perspective of the training accuracy in Fig. 6. Moreover, the emergence of desynchronization accelerates the training speed of DeepSCA. The DeepSCA is robust to desynchronization, which can be explained in terms of local connection characteristics of the convolutional layer [13].

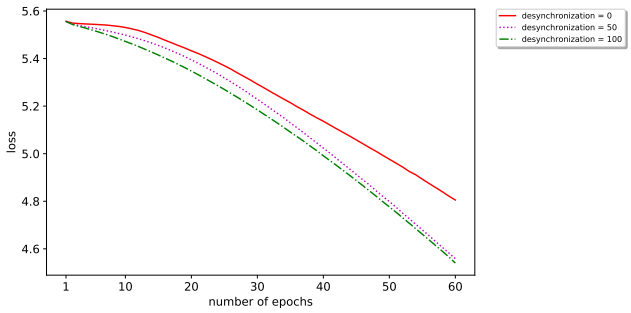

Fig. 5. The training loss of DeepSCA with different numbers of epochs when the desynchronization amount is 0, 50, 100.

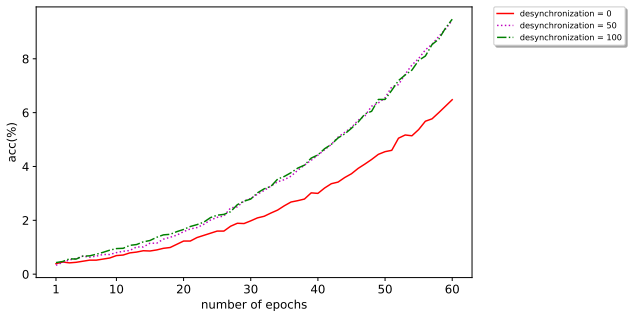

Fig. 6. The training accuracy of DeepSCA with different numbers of epochs when the desynchronization amount is 0, 50, 100.

Interestingly, the training loss and accuracy of DeepSCA are almost the same when the desynchronization amount is 50 or 100. Remarkably, the accuracy of random guessing accounts for about 0.39% ($\frac{1}{256}$). Moreover, we do not further increase the number of epochs to reduce loss and improve test accuracy. The main reason for this choice is that the SCA-efficiency of DeepSCA after training 60 epochs is sufficient when we use the mean rank function as a measure.

The second experiment was used to discuss the effect of batch normalization on the SCA-efficiency of DeepSCA. Therefore, we removed batch normalization layers of DeepSCA and then retrained this network. Figures 7, 8 and 9 show the mean rank of DeepSCA without batch normalization layers on the test set as the training phase progresses.

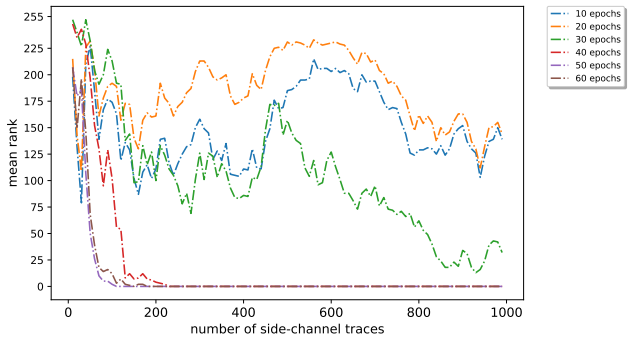

Fig. 7. Mean ranks of DeepSCA (without batch normalization layers) with different numbers of epochs when the desynchronization amount is 0.

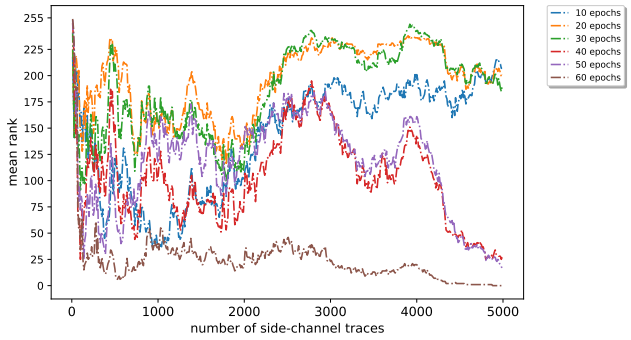

Fig. 8. Mean ranks of DeepSCA (without batch normalization layers) with different numbers of epochs when the desynchronization amount is 50.

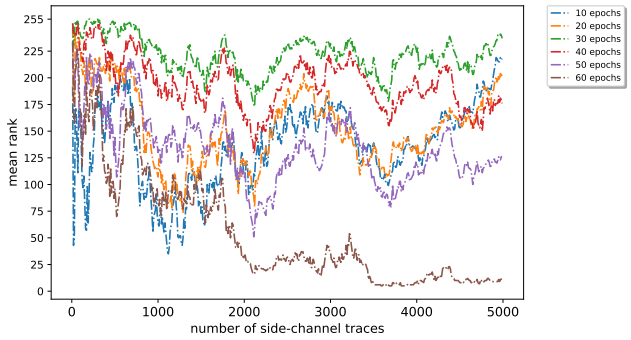

Fig. 9. Mean ranks of DeepSCA (without batch normalization layers) with different numbers of epochs when the desynchronization amount is 100.

The DeepSCA with batch normalization layers has the higher test accuracy. Merely adding batch normalization yields a substantial speedup in training. During the training phase, the DeepSCA with batch normalization layers takes extra time to normalize the output, thus increasing the training time of each epoch from 10 seconds to 11 seconds. The time required for each epoch can be further reduced by increasing the learning rate, using Dropout, and applying other modifications supported by batch normalization [16]. However, we are more interested in validating the effectiveness of batch normalization, rather than finding the optimal parameters to achieve the state of the art performance. Therefore, we have not specifically discussed the above details.

The mean ranks obtained from DeepSCA and CNN_best models are compared in Figs. 10 and 11. DeepSCA outperforms significantly CNN_best on desynchronized side-channel traces with only 60 epochs, although these two models are essentially instantiated from the VGG architecture. DeepSCA greatly reduces the number of side-channel traces required to reveal the secret key on highly desynchronized datasets. DeepSCA can learn more complex and more abstract representations from the training set due to stacking more convolutional/pooling layers. Furthermore, the batch normalization operation makes the network more robust and easier to generalize. Interestingly, the test accuracy of DeepSCA and CNN_best models is obviously at a low level (0.8%, random guess accuracy is 0.39%) in our experiments. This can be explained by the fact that both DeepSCA and CNN_best models approximate the distribution of multiple side-channel traces rather than single traces.

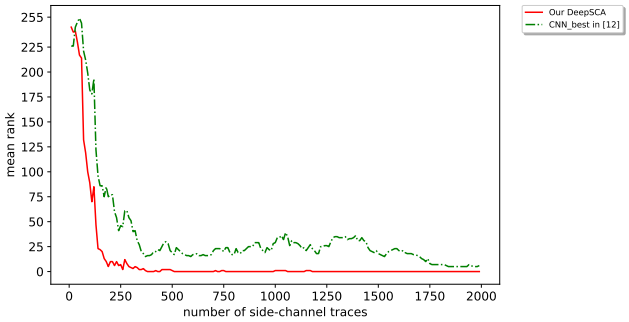

Fig. 10. Mean ranks of DeepSCA with different numbers of epochs when the desynchronization amount is 50.

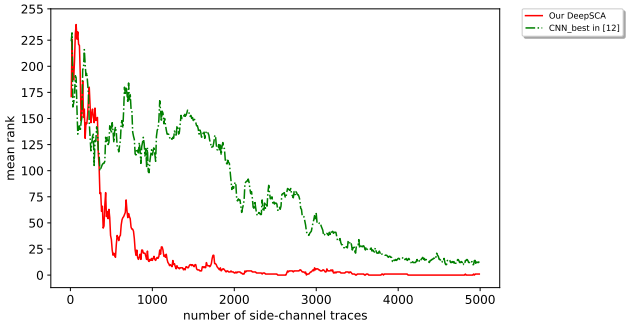

Fig. 11. Mean ranks of DeepSCA with different numbers of epochs when the desynchronization amount is 100.

In this case, we introduced the idea of the pre-trained mechanism to simplify our training process, which allowed us to get more accurate parameters in a time-saving manner [21]. Generally, it is feasible to use a pre-trained network if both datasets have something in common. The pre-trained model contains weights and biases that represent features learned from other datasets through an extensive training process. These learned features are often transferable to different datasets. With the pre-trained mechanism, instead of implementing the learning process from scratch, we can start with features that have been learned when dealing with similar problems.

We first trained DeepSCA for 40 epochs on the dataset with a desynchronization amount of 0 as the pre-trained model. For convenience, the pre-trained network is called $DeepSCA^{0}_{40}$, Next, the $DeepSCA^{0}_{40}$ network was trained on the datasets with the desynchronization amount of 50 or 100. The $DeepSCA^{0}_{40}$ network has already learned rich features, but it can learn more patterns that are specific to the new dataset when the network parameters are fine-tuned. Figures 12 and 13 report the results of DeepSCA with the pre-trained network when the desynchronization amounts are 50 and 100. The "0 epochs" in the legend represents the $DeepSCA^{0}_{40}$ network is used directly to make predictions the case where the desynchronization amount is 50 or 100.

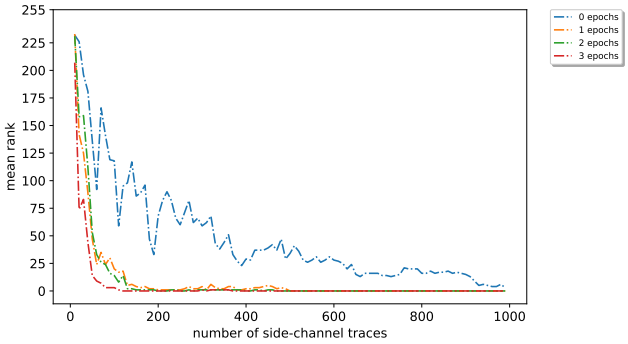

Fig. 12. Mean ranks of DeepSCA with the pre-train network when the desynchronization amount is 50.

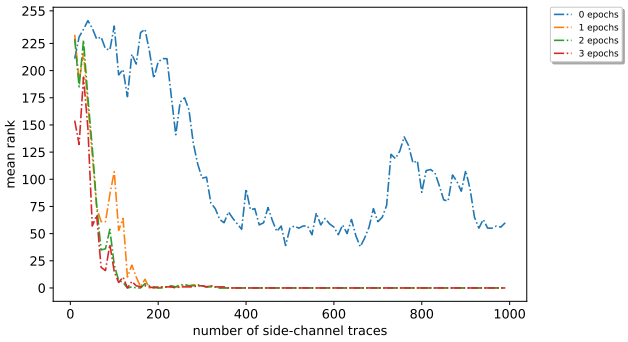

Fig. 13. Mean ranks of DeepSCA with the pre-train network when the desynchronization amount is 100.

As we expected, the SCA-efficiency of the fine-tuned $DeepSCA^{0}_{40}$ network is far superior to the DeepSCA network trained from scratch (see Figs. 3 and 4). For the pre-trained $DeepSCA^{0}_{40}$ network, we only need to execute 3 epochs to fine-tune parameters, which spend very little computing resources. The pre-trained DeepSCA network is able to reveal the secret key by using less than 200 traces during the attack phase. The presence of desynchronization or jittering in the synchronous traces (data augmentation) is equivalent to adding regularization term to the objective function [22]. This is beneficial in the classification process as it enhances the robustness of DeepSCA.

5. Conclusions

The application of deep learning to profiled SCA attacks requires careful configuration of the hyper-parameters of deep neural networks. The profiled SCA attacks require deep neural networks to identify leaked information about side-channel traces. Deep neural networks can approximate highly complex functions by stacking different layers. As a consequence, deep learning might be able to break widely adopted countermeasures like masking, desynchronization or shuffling.

In this study, we have demonstrated that the proposed DeepSCA can efficiently recover the key byte in presence of desynchronization. The main reason that DeepSCA is robust to the desynchronized side-channel traces is that the convolutional layer extracts features independently of their locations in one trace. Therefore, DeepSCA can bypass jitter-based effects from unstable clock domains. The results are in agreement with the conclusions in the article [12]. The difference is that batch normalization significantly improves the SCA-efficiency of DeepSCA in this paper. Moreover, the pre-trained DeepSCA model simplifies our training process and allows us to get more accurate parameters in a time-saving manner.

The CNN_best model proposed in paper [12] has $66,652,544$ parameters that need to be trained, which is too large for the general applications. In comparison, our DeepSCA is more lightweight, but there are still $20,011,200$ trainable parameters. The last two fully-connected layers account for most of the parameters. In the initial attempt to design the DeepSCA architecture, we want to reduce the number of neurons or even remove the fully-connected layer. Unfortunately, our experimental results do not support this view. Nevertheless, further investigations on the application of the lightweight neural network architectures (e.g., SqueezeNet [23], MobileNets [24], and ShuffleNet [25]) are necessary in profiled SCA attacks.

References

[1] KOCHER, P. C., JAFFE, J., JUN, B. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*. London (UK), 1999, p. 388–397. DOI: 10.1007/3-540-48405-1_25

[2] MARTINASEK, Z., HAJNY, J., MALINA, L. Optimization of power analysis using neural network. In *Proceedings of the 12th International Conference Smart Card Research and Advanced Applications (CARDIS)*. Berlin (Germany), 2013, p. 94–107. DOI: 10.1007/978-3-319-08302-5_7

[3] WHITNALL, C., OSWALD, E. Robust profiling for DPA-style attacks. In *Proceedings of the 17th International Workshop Cryptographic Hardware and Embedded Systems (CHES)*. Saint-Malo (France), 2015, p. 3–21. DOI: 10.1007/978-3-662-48324-4_1

[4] MARTINASEK, Z., ZEMAN, V., MALINA, L., et al. k-Nearest neighbors algorithm in profiling power analysis attack. *Radioengineering*, 2016, vol. 25, no. 2, p. 365–382. DOI: 10.13164/re.2016.0365

[5] HOSPODAR, G., GIERLICHS, B., MULDER, D. E., et al. Machine learning in side-channel analysis: A first study. *Journal of Cryptographic Engineering*, 2011, vol. 1, no. 4, p. 293–302. DOI: 10.1007/s13389-011-0023

[6] HEUSER, A., ZOHNER, M. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In *Proceedings of the Constructive Side-Channel Analysis and Secure Design: Third International Workshop (COSADE)*. Darmstadt (Germany), 2012, p. 249–264. DOI: 10.1007/978-3-642-29912-4_18

[7] BARTKEWITZ, T., LEMKE-RUST, K. Efficient template attacks based on probabilistic multi-class support vector machines. In *Proceedings of Smart Card Research and Advanced Applications*. Graz (Austria), 2013, p. 263–276. DOI: 10.1007/978-3-642-37288-9_18

[8] HOU, S. R., ZHOU, Y. J., LIU, H. M., et al. Wavelet support vector machine algorithm in power analysis attacks. *Radioengineering*. 2017, vol. 26, no. 3, p. 890–902. DOI: 10.13164/re.2017.0890

[9] HOU, S. R., ZHOU, Y. J., LIU, H. M., et al. Exploiting support vector machine algorithm to break the secret key. *Radioengineering*, 2018, vol. 27, no. 1, p. 289–298. DOI: 10.13164/re.2018.0289

[10] MAGHREBI, H., PORTIGLIATTI, T., PROUFF, E. Breaking cryptographic implementations using deep learning techniques. *Cryptology ePrint Archive*, Report 2016/921, 2016, p. 1–25. Available at: https://eprint.iacr.org/2016/921

[11] CAGLI, E., DUMAS, C., PROUFF, E. Convolutional neural networks with data augmentation against jitterbased countermeasures - profiling attacks without pre-processing. In *Proceedings of the 19th International Conference on Cryptographic Hardware and Embedded Systems (CHES)*. Taipei (Taiwan), 2017, p. 45–68. DOI: 10.1007/978-3-319-66787-4

[12] PROUFF, E., STRULLU, R., BENADJILA, R., et al. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *Cryptology ePrint Archive*, Report 2018/053, 2018, p. 1–45. Available at: https://eprint.iacr.org/2018/053.

[13] GOODFELLOW, I., BENGIO, Y., COURVILLE, A. *Deep Learning*. Cambridge (USA): MIT Press, 2016. Available at: http://www.deeplearningbook.org. ISBN: 0262035618

[14] BENGIO, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2009, vol. 2, no. 1, p. 1–127. DOI: 10.1561/2200000006

[15] IOFFE, S., SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. Lille (France), 2015, p. 448–456.

[16] SANTURKAR, S., TSIPRAS, D., ILYAS, A., et al. How does batch normalization help optimization? *arXiv*, 2018, p. 1–26. Available at: https://arxiv.org/abs/1805.11604

[17] SIMONYAN, K., Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2014, p. 1–14. Available at: https://arxiv.org/abs/1409.1556

[18] JARRETT, K., KAVUKCOUOGLU, K., RANZATO, M., et al. What is the best multistage architecture for object recognition? In *Proceedings of the IEEE 12th International Conference on Computer Vision (ICCV)*. Kyoto (Japan), 2009, p. 2146–2153. DOI: 10.1109/ICCV.2009.5459469

[19] CHOLLET, F., et al. *Keras: Deep Learning for Humans*. Available at: https://github.com/fchollet/keras

[20] ABADI, M., AGARWAL, A., BARHAM, P., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems. *arXiv*, 2015, p. 1–19. Available at: https://arxiv.org/abs/1603.04467. Software available at: https://tensorflow.org

[21] RAWAT, W., WANG, Z., Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 2017, vol. 29, no. 9, p. 2352–2449. DOI: 10.1162/neco_a_00990

[22] VAN DYK, D. A., MENG, X.-L. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 2001, vol. 10, no. 1, p. 1–50. DOI: 10.1198/10618600152418584

[23] IANDOLA, F. N., HAN, S., MOSKEWICZ, M. W., et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv*, 2016, p. 1–13. Available at: https://arxiv.org/abs/1602.07360

[24] HOWARD, A. G., ZHU, M., CHEN, B., et al. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv*, 2017, p. 1–9. Available at: https://arxiv.org/abs/1704.04861

[25] ZHANG, X., ZHOU, X., LIN, M., et al. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City (USA), 2018, p. 1–9. DOI: 10.1109/CVPR.2018.00716

About the Authors . . .

Shourong HOU was born in Shandong, China. He received his B.Eng. from Xidian University in 2015. His research interests include machine learning and side channel attack. He is now a Ph.D. candidate at Shanghai Jiao Tong University.

Yujie ZHOU was born in Zhejiang, China. She received his Ph.D. from University of Science and Technology of China in 1997. Her research interests include digital integrated circuit design, embedded system and digital copyright protection.

Hongming LIU was born in Jiangxi, China. He received his Ph.D. from Shanghai Jiao Tong University in 2014. His research interests include chip design and machine learning.