Open Access ISSN: 3007-1437 (Online), ISSN: 3007-1429 (Print)

IRAQI Academic Scientific Journals

Alkadhim Journal for Computer Science (KJCS)

Journal Homepage: https://alkadhum-col.edu.iq/JKCEAS

KJCS ALKADHIM JOURNAL FOR COMPUTER SCIENCE

Design of Deep Learning Techniques for Side-Channel Attacks on Masked 128-bit AES Implementations

[1]Mohammed Saeb Nahi, [2]Ahmed Imran Fattah, [3]Hassan Jameel Mutashar & [4] Opeyemi Lateef Usman

[1]Department of Computer Techniques Engineering, Al-Kadhum College (IKC), Baghdad, Iraq;
mohammed.saib@alkadhum-col.edu.iq

[2]Department of Computer Techniques Engineering, Al-Kadhum College (IKC), Baghdad, Iraq;
ahmed.omran@alkadhum-col.edu.iq

[3]Department of Computer Techniques Engineering, Al-Kadhum College (IKC), Baghdad, Iraq;
hasan.jamel@alkadhum-col.edu.iq

[4]Department of Computer Science, Tai Solarin University of Education Ogun, Nigeria
p99943@siswa.ukm.edu.my

Article information

Article history:
Received: February, 11, 2024
Accepted: March, 11, 2024
Available online: March, 14, 2024

Keywords:
AES, Deep Learning, VGG, Side Channel Analysis and Attacks

Corresponding Author:
Mohammed Saeb Nahi
mohammed.saib@alkadhum-col.edu.iq

DOI:
https://doi.org/10.53523/ijoirVolxIxIDxx

Abstract

Researchers are exploring the use of convolutional neural networks (CNNs) in side-channel attacks to understand the weaknesses in cryptographic implementation. CNNs can learn hierarchical characteristics automatically from electromagnetic radiation or power usage during cryptographic processes. Researchers train CNNs on side-channel data to extract meaningful representations and deduce secret keys. Deep learning algorithms are helpful in evaluating the security of embedded systems, and CNNs are a feasible paradigm for profiling side-channel analysis attacks. In this paper, it has been introduced a VGG (Visual Geometry Group)-Net architecture, which is a typical deep convolutional neural network design with numerous layers. It uses the ASCAD dataset to conduct experiments. They found that VGG-Net architecture Side Channel Attacks (SCA) provides better results than the previously optimized CNN model by significantly reducing the number of side-channel traces required for successful attacks on desynchronized datasets. The researchers also discovered that synchronous traces serve as the pre-training source for VGG-Net architecture, functioning successfully in terms of jittering with minimal fine-adjusting after training.

1. Introduction

It is possible to safeguard the confidentiality of messages that are sent between peers by using the most used block cipher [1], which is known as AES (Advanced Encryption Standard)[2]. On the other hand, a variety of side-channel attacks have been proposed as a straightforward method for an adversary to discover secret keys that are

stored in hardware [3],[4]. In the realm of side-channel analyses, one of the most prevalent attacks is the power analysis attack, which considers the utilization of power as information that has been disclosed. In particular, side-channel attacks[5],[6] against the implementation of naïve block cipher algorithms are considered to be significant hurdles when it comes to ensuring information confidentiality. The problem of side-channel analysis has been addressed by several different strategies that have been proposed. It is common practice to use masking countermeasures to obstruct side-channel analysis, which is a process in which we randomly construct the intermediate computation values. The masking strategy is another method that may be used to ensure the safety of the AES implementation[7].

To protect the compromised AES implementation, we use a Boolean masking approach. A Boolean function, such as the XOR operation, is used in this context to create the output of the S-Box randomly[8]. It is possible to determine the amount of security provided by the masking approach by using the number of secret shares. A d-masking is a countermeasure[9] that applies d-masks to a single variable. This countermeasure is known as d-masking, and it has the potential to give resistance against d-masking order attacks[10].

To obtain the secret key, this paper proposes a variety of power analysis attacks that make use of either profiling or nonprofiling techniques that are based on Convolutional Neural Network (CNN)[11] models. Since Z originally presented the concept of side-channel attack, there have been several fascinating advancements in the actual research that has been conducted in this sector[12].

1.1 Review of Works

According to Martinasek et al.[13], it was shown that MLP can crack naïve AES. In addition, R. Gilmore et. al., demonstrated that a neural network-based profile attack has the potential to make the masked countermeasure of AES. The researchers achieved an accuracy of 91.8% when extracting the mask values and 88.4% when extracting the secret keys, respectively. Through the use of Differential Deep Learning Analysis (DDLA)[14], it can be circumvent that the first-order masking countermeasure when the circumstance does not include profiling. An attack method that does not use profiling and is based on deep learning is referred to as the DDLA[15]. It can train the deep learning models for each hypothetical key during the attack phase, which does not include the profiling phase. This phase does not contain data collection. In the case that a verified key was available, it would be possible to acquire the correct key by comparing parameters such as accuracy and cost (loss). When it comes to establishing the efficacy of non-profiling attacks, one of the most important components is the identification of the label for deep learning models. Examples of such labels include binary and HW (Hamming Weight) labeling. The authors proposed a HW-based binary labeling approach based on the idea that efficiency and performance should be taken into consideration[16].

In the article [17], the authors report on many tests that were conducted on the implementation of masked AES. These tests used real power traces that were measured from the ASCAD dataset for attacks that did not include profiling and the Chip Whisperer platform was used for attacks that involved profiling. The deep learning-based analysis of the profiling attacks was divided into two steps: the mask value recovery phase and the masked S-box output recovery phase [18]. Both phases were segregated into their distinct phases. To directly implement AES-128 encryption, the authors make use of the XMEGA128 microcontroller. Additionally, they use CNN and MLP models to carry out power analysis attacks [19] To break the masked AES using the MLP-based DDLA, the authors do not need to split the attack phases or do any preprocessing. As a result, they were able to confirm that the secret key of masked AES may be used to produce an accuracy of 81.0% with MLP and 75.4% with CNN. Furthermore, the DDLA attack is fully capable of extracting the secret key of the disguised AES implementation when the epoch value is 10 or higher. This is particularly true when the epoch value is larger than 10 [20].

2. Methodology

Understanding the basics is crucial to using deep learning to characterize SCA attacks. Supervised learning with known input and key is the problem. Using the hypothetical leakage model, input and key label side-channel traces were collected. The side-channel trace dataset may be split into training and test sets. An ideal testing set is measured using a device comparable to the one being tested but with an unknown key. Training requires a lot of processing power, whereas testing is quick. VGG networks with unique design strategies are the convolutional neural network architecture employed in SCA attacks, we believe. Photo classification network VGG was created. Our side-channel traces are one-dimensional, whereas the incoming signal is two-dimensional. Convolution and pooling can be done on one-dimensional data, fortunately. We chose the VGG architecture after creating hyper-parameter limitations from literature and testing. Random search may provide VGG architecture hyper-parameter values. The amount of pooling and convolutional substrate, the number of epochs and learning speed, normalization collection, batch proportion, abandonment, convolutional functions for activation, number of neurons in each fully-connected layer, padding, stride length, kernel dimension of a convolutional filter, and stochastic descent gradient must be tuned.

Six convolutional, pooling, and two fully-connected layers make up our deep neural network. Filters with a kernel size of 11 and a stride of 1 generate numerous activation mappings for each convolutional layer. Convolutional layers activate using Leaky Rectified Linear Units. Zero-slope sections are eliminated in the Leaky ReLU to solve the dying ReLU issue. As convolutional and pooling layers are added, the network may learn more abstract representations from training input. Each convolutional layer has the same padding, so the output is the same size as the input. The number of filters each layer increases geometrically from 32 to 512. The average pooling kernel doubles in size and stride. Both layers are connected and h SoftMax operates 2048 units. Based on input parameters, the classification layer computes a distribution of possibilities using SoftMax. Fully connected and convolutional substrate layers have no dropout. Normalizing batches is a common way to train deep neural networks quickly and reliably. We batch normalized the output values using the main and final convolutional layers after utilizing Leaky ReLU activation. For this reason, we called our deep neural network architecture DLSCA in writing and comparison. Figure 1 shows the DLSCA network design, and Table 1 lists its hyper-parameters. Input/output forms, filter size, sampling, and activation function are hyper-parameters. Unlike VGG best, our DLSCA adds one convolutional/pooling layer and batch normalization after the first and last convolutional layers to complete the neural network. We reduced trainable parameters by reducing the number of neurons in the last two fully connected layers from 4096 to 2048. Several hyper-parameters were considered while training a deep neural network to characterize SCA attacks. Due to space constraints, we cannot study each hyper-parameter independently in the following section.

3. Evaluation Metrics

Accuracy and loss are two measure types that are often used in machine learning. A training set's successful classification rate is referred to as its training accuracy, whereas the training set's error rate is referred to as its training loss. Over the course of many epochs, we can determine Test precision is a measurement of how accurately the model that has been trained predicts side-channel examines from the test set that has not been taught; on the other hand, the key recovery precision for profiled SCA attacks is rather inadequate. It is observed that the deep neural network is learning about the training set while the training accuracy and the training loss decreases. The mathematical equations for accuracy[17] and loss[18] functions are shown in equations (1) and (2) respectively.

$$\text{accuracy} = (\text{True}_{\text{Positive}} + \text{True}_{\text{Negative}})/(\text{True}_{\text{Positive}} + \text{True}_{\text{Negative}} + \text{False}_{\text{Positive}} + \text{False}_{\text{Negative}}) \quad (1)$$

$$Loss\ function\ \lfloor(\theta) = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)\text{^}2 \quad (2)$$

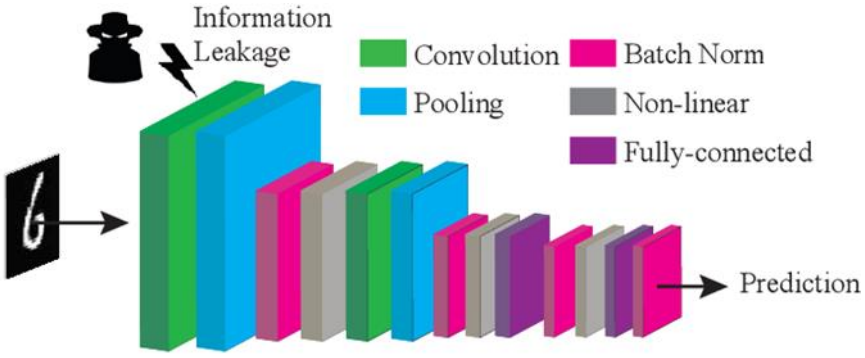

Figure (1): The architecture of DLSCA. It is a green rectangular block that represents the one-dimensional convolutional layer, orange layers represent the batch normalization layers, and blue layers represent the average pooling layers. The last average pooling lay

Table (1): The DLSCA architectural specifications.

Layers	Trace Input	Convolution 1D	Drop out	Batch Normalization	Convolution 1D	Drop out	Convolution 1D	Drop out	Input	Output
Hyperparameters	-	F=16, F_l=8, ReLU, MaxPL=2	P_{Drop} = 0.3	-	F=32, F_l=8, ReLU, MaxPL=2	P_{Drop} = 0.3	F=64, F_l=8, ReLU, MaxPL=2	P_{Drop} = 0.3	256 neurons	256 neurons

During multi-trace investigations, utilized the function of rank, rather often to evaluate strikes. When we are using that greatest likelihood estimate to obtain the secret key, we concentrate on all side channel traces' ultimate probability output. Rank function scores take into consideration the likelihood of key candidate output. Ranking all of the key candidate scores is what the rank function does. If the attack is successful, the function for the correct secret key k∗ will possess a rank of zero. The definition of the rank function is given in eq. [19]:

$$\text{rank(ek, Sa)} = ||\{k_i \in K | score(k_i) > score(k*)\}|| \quad (3)$$

Where i = 0, 1, 2,..., 255 and score (k) stands for a crucial candidate's score. For improved rank function measurements, we execute Ten times cross-validation in subsequent attack trials. The mean rank function is the proper key's average rank. Thus, profiled SCA attacks are evaluated using the mean rank function.

3.2 Algorithm for Side-Channel Attacks on Masked 128-bit AES:

Presented below in algorithm 1 is the approach for side-channel attacks against masked 128-bit AES. The first step in launching an attack on the target device is to collect input power traces and matching plaintexts from a copy of the target device. A final prediction about the rank of the suitable candidate key is made by the model. As part of the ID classification model, we first sort the list of components produced by the model, which includes all of the probabilities linked to each possible value. The key rank is the index of the key byte being targeted within the list.

Algorithm 1:

Entry: *Plaintexts; 128-bit key Key, tr: traces*
Step1: *Key as the outputRank: One byte's average "key rank"*
Step2: *Set up maxepoch and SOP for epoch= 1 as follows: maximal epoch*

Step3: *Divide the input data (tr and Pl) into attack (tra and Pla) and profile (trp and Plp) subsets.*
Step4: *Train Deep K-TSVM using Algorithm 1 for num = 0.*
Step5: *Size(Pla) does the following to forecast tra[num] for k=0: 255 do L=SBox(Pla[num] ⊕ k);*
Step6: *Add class L probability prediction to SOP[k]*
Step6: *To calculate the average key rank throughout all epochs, save the key*
Step7: *end.*

4. Experiments and Results

The DLSCA framework is constructed using Kera's and TensorFlow as the backend. The training is carried out on a personal computer configured with an Intel Processor Xeon E5-2698 v4 processing device operating at 2.2 GHz, 16 gigabytes of random-access memory RAM) as well as an NVIDIA Tesla P100 graphics card. With every single trial, profiled traces from ASCAD datasets were selected, which randomly included 256 different classes.

4.1 Dataset of ASCAD

The architecture of DLSCA was put through its paces by a dataset from ASCAD, that may be accessed via ANSSI-FR/ASCAD's GitHub page. An 8-bit AVR microcontroller that operates with a masked AES-128 algorithm can acquire side-channel traces via the use of electromagnetic emission. By the MNIST format, the dataset contains a total of 50000 side-channel traces for training purposes and 10000 for testing purposes. The S-Box output byte without protection, denoted as The S-Box[pi ⊕ ki] model can leak information. The Masked S-Box is the pre-selected window that contains 700 samples that are important to the side-channel trace. The only S-Box that is targeted by our attack testing in this research is the third one that is processed during the first round of AES.

Through the process of parametrically desynchronizing side-channel traces in the ASCAD dataset, the SCA efficiency of different attack techniques against jittering is analyzed and evaluated. Desyncmax desynchronization allows for the random movement of each side-channel trace to the left, with δ belonging to that range [0, Desyncmax]. The creation of a fresh side-channel trace serves as the result of desynchronizing that trace. Because of its translation invariance, the VGG side-channel traces can be used by architecture to retrieve information that has been desynchronized.

4.2 Results

Within the scope of this study, the impacts of hyper-parameters on DLSCA's efficiency have not been thoroughly investigated. This is because Proof et al. [20] address VGG hyper-parameter selection in great detail. This is the primary explanation for this. As a result, hyper-parameters including loss function, batch size, padding, kernel size, learning rate optimization, and so on are not investigated in this article. In this study, we investigated how DLSCA's efficiency is affected by epochs, batch normalization layer, and pre-training method. Additionally, we compared the effectiveness of DLSCA with VGG's best SCA in our testing.

To prepare DLSCA for the first experiment, many epochs were used for training. The hyper-parameter epoch is responsible for determining the frequency at which a learning algorithm revisits the training set. Every training set instance has the potential to occur within each epoch to change the model's internal parameters. DLSCA can better match training set side-channel traces by acquiring optimum parameters when the number of epochs is increased. The average rank of DLSCA is shown in Figure 2 as a function of epochs with desynchronization values of 100. The SCA efficiency of DLSCA also rises in tandem with the mean rank's growth during the course of epochs. According to these data, our rank function does not seem to overfit as the number of epochs increases.

As long as the training phase includes a sufficient number of epochs, the mean rank will be very close to 0. As maximum desynchronization measures rise, more time must be spent during the testing phase training DLSCA parameters settings and side-channel traces, which is also known as the crucial recovery phase. Since desynchronization is rising, epochs are having an increasingly negative impact on DLSCA's efficiency.

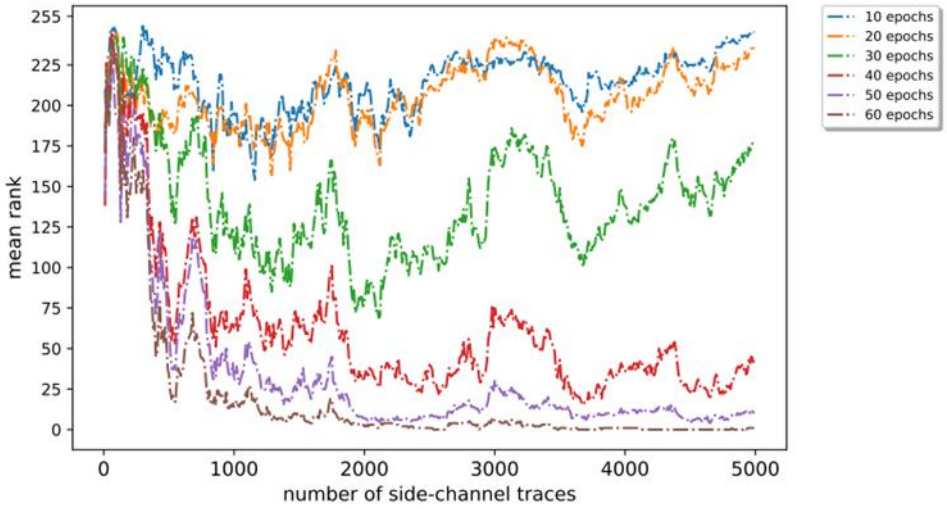

Figure (2): DLSCA mean ranks for different epochs when desynchronization is equal to zero

For our datasets, an epoch measurement of fifty is adequate due to median rank.; nevertheless, desynchronizing side-channel traces does not increase performance. For training duration and robustness's sake, this example makes use of sixty epochs. Whenever Desyncmax is set to zero, DLSCA can record less than one hundred side-channel traces. There is a need for three thousand traces in the scenario where the highest desynchronization value is one hundred and the mean rank is zero. DLSCA is effective when working with desynchronized traces; nevertheless, to retrieve the secret key during the test phase, it needs several side-channel traces to be completed.

Table (2): The DLSCA training loss is shown in with 0, 50, and 100 epochs of desynchronization present.

No. of epochs	Loss		
	Desynchronization=0	**Desynchronization=50**	**Desynchronization=100**
1	5.56	5.56	5.56
20	5.48	5.4	5.32
40	5.18	5.04	5.0
60	4.82	4.52	4.5

Table 2 shows the DLSCA training loss with 0, 50, and 100 epochs of desynchronization present. When desynchronization is 0, 50, or 100, Figure 3 illustrates how the loss of training differs depending on number of epochs it takes. Through the use of epochs, training loss is reduced. Data shown in Figure 4 demonstrates that DLSCA's SCA efficiency continues to improve due to training accuracy improves. The speed of DLSCA training is increased by desynchronization. The desynchronization resilience of the DLSCA may be explained by the elements of the convolutional layer that are associated with local connections.

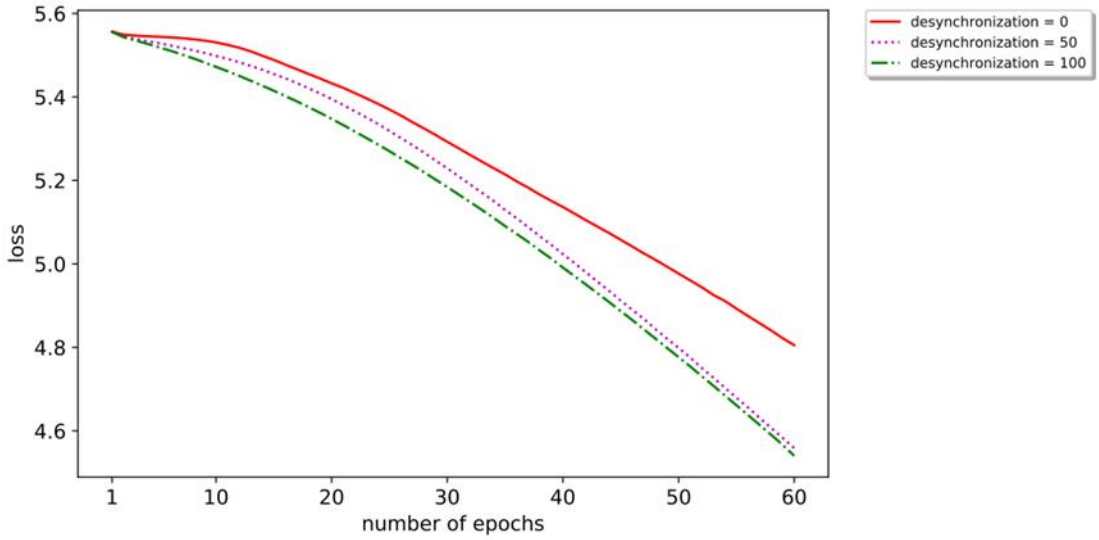

Figure (3): The DLSCA training loss is shown in with 0, 50, and 100 epochs of desynchronization present.

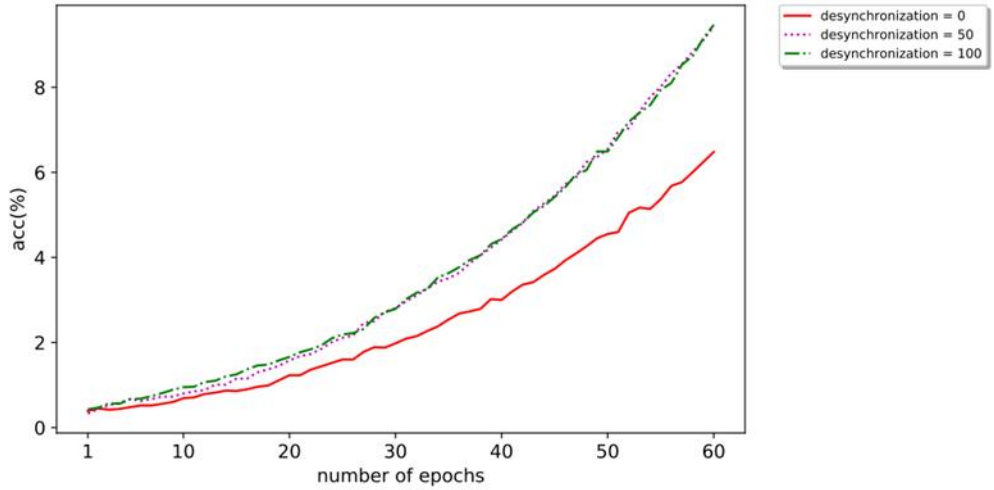

Figure (4): Desynchronization epochs of 0, 50, and 100 are shown to illustrate the accuracy of DLSCA training.

Both desynchronization 50 and desynchronization 100 have a similar impact on the amount of loss and accuracy that DLSCA training experiences. An interesting fact is that random guessing has a 0.39% accuracy rate (1 256). Furthermore, we do not employ more epochs in an attempt to decrease loss and raise test accuracy. The main argument in favor of this option is that DLSCA's SCA efficiency at sixty epochs is enough when a median rank mechanism is applied and becomes sufficient. The secondary proceeding investigated the impact that batch normalization has on the efficiency of DLSCA SCA computations. To do this, we removed the Deep-SCA batch normalization layers and then retrained it. On the test set, the DLSCA median rank in the absence of normalization across batch stages is shown in Figure 5. This indicates that the training process was successful.

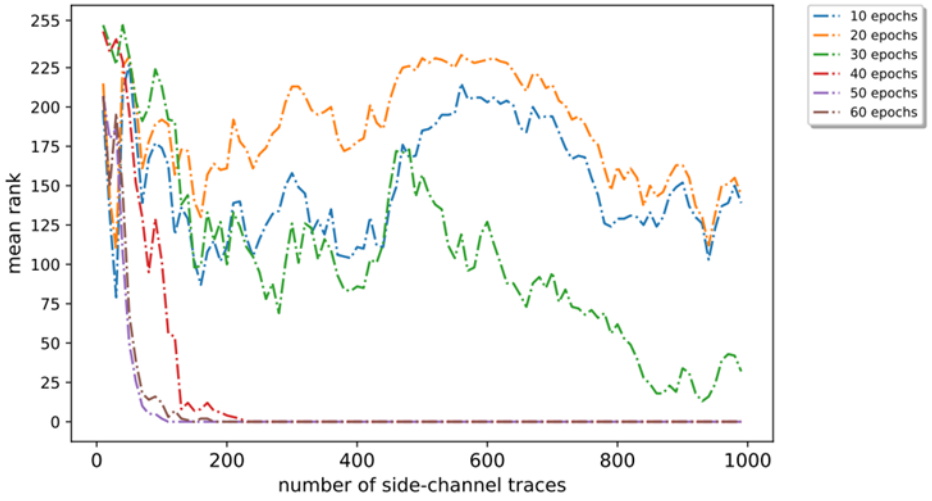

Figure (5): The mean DLSCA ranks for several epochs where desynchronization is zero.

DLSCA, when used with batch normalization layers, results in tests that are more accurate. The process of training is substantially sped up by batch normalization. When using DLSCA with batch normalization layers, the process of normalizing output during training takes longer, which increases in training time of each period was between 10 and 11 seconds. One way to reduce the amount of time spent on epochs is to increase the learning rate, implement Dropout, and make other adjustments that are enabled by batch normalization. Instead of focusing on finding the optimal configurations for cutting-edge performance, our primary objective is to verify batch normalization. As a result, we have not addressed the details that were discussed before.

Figures 6 compare VGG's best mean ranks and DLSCA mean rankings, respectively. For desynchronized side-channel records having 60 epochs, DLSCA performs better than VGG, even though it was constructed with the architecture of VGG. DLSCA minimizes the quantity of side-channel traces needed to disclose secret keys for significantly desynchronized datasets. The layers of convolution and pooling enable DLSCA to extract increasingly abstract and sophisticated interpretations from the training set. Batch normalization strengthens and expands the network's capacity for generalization. Our results show that the average guess accuracy is substantially higher at 0.39%, whereas DLSCA as well as VGG best models have poor test accuracy of 0.8%. Both DLSCA and VGG best models attempt to estimate the distribution of several side-channel traces rather than a single trace from each side channel.

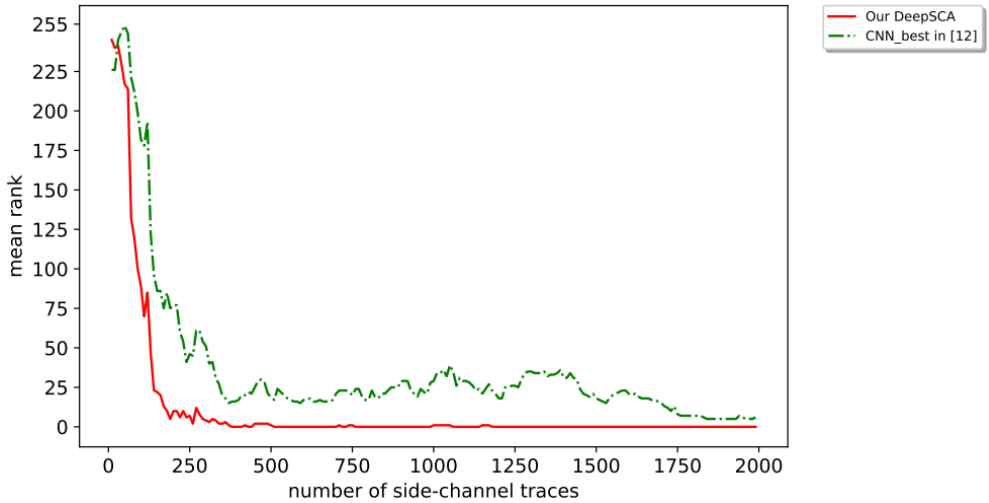

Figure (6): The mean ranks of DLSCA with a variety of epochs and 50 desynchronizations

The pre-trained mechanism was developed to streamline and obtain more precise parameters in a training

procedure of that shorter amount of duration. When datasets are comparable, it is possible to employ pre-trained neural networks. Both the weights and biases of the pre-trained model are reflective of the properties that were acquired from the prior datasets via the process of comprehensive training. It is possible to transfer learned properties to different datasets. Rather than beginning from zero, the pre-trained approach enables us to take advantage of features that have been learned from problems that are also similar.

The DLSCA algorithm was used to train the pre-trained model on the dataset for a total of forty epochs with no desynchronization. For the sake of convenience, the DLSCA040 network was trained on datasets that had either fifty or one hundred desynchronizations after the pre-training phase. By fine-tuning its parameters, the DLSCA040 network may be able to identify patterns that are specific to the dataset that it is currently working with, even though it has learned numerous features. Two examples of DLSCA using the pre-trained network are shown in Figure 7 at 50 desynchronizations. When referring to the DLSCA040 network, which is used to anticipate desynchronization quantities of fifty or one hundred, the phrase "0 epochs" is utilized in the caption.

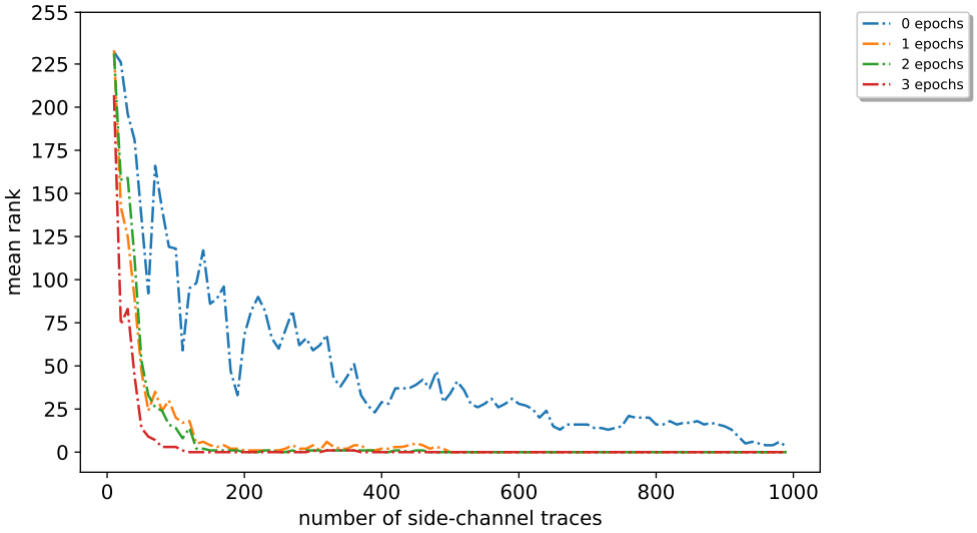

Figure (7): The average DLSCA ranks on a 50 desynchronization level using the pre-train network.

As was anticipated, the DLSCA040 network that has been fine-tuned has a much greater SCA efficiency than the DLSCA network that was first developed. To fine-tune the parameters of the pre-trained DLSCA040 network, which requires less CPU resources, we just need three epochs to complete the process. The DLSCA network that has been pre-trained makes use of less than 200 traces to reveal the secret key during the attack phase. Regularization is added to the target function by using when synchronous traces exhibit jittering or desynchronization, which is referred to as data augmentation. This results in DLSCA being more robust throughout the classification process.

5. Conclusion

To perform profiled SCA attacks using deep learning, it is necessary to do exact hyper-parameter tuning of deep neural networks. Deep neural networks are required to locate stolen information about side-channel traces in attacks that are profiled as SCA. The ability to approximate complicated functions using deep neural networks is achieved through the overlaying of levels. Therefore, deep learning has the potential to interfere with concealment, switching, and desynchronization. In this inquiry, we proved that DLSCA is capable of retrieving the key byte efficiently during the desynchronization process. This is the reason why the convolutional layer of DLSCA collects features independent of their placement inside a single trace, making it resistant to desynchronized side-channel recordings. Considering that jitter is caused by unstable clock domains, DLSCA can prevent it. This article demonstrates how batch normalization may improve the SCA efficiency of DLSCA. Through the provision of more exact parameters, the pre-trained DLSCA model makes training easier and saves time. Since the VGG best

model requires 66,652, 544 characteristics to train, its extreme complexity prevents it from being useful in most applications. Even though our DLSCA is lighter, it yet has 20, 011, 200 trainable parameters. The final two completely connected layers regulate most of the settings. While the DLSCA architecture was being designed initially, we intend to either remove the fully linked layer or decrease the total number of neurons. Our test results, however, run counter to this. However, further research is needed on lightweight neural network architectures for profiled SCA attacks.

References

[1] K. Kuroda, Y. Fukuda, K. Yoshida, and T. Fujino, "Practical aspects on non-profiled deep-learning side-channel attacks against AES software implementation with two types of masking countermeasures including RSM," in *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, 2021, pp. 29–40.

[2] A. A. Ahmed, M. K. Hasan, N. S. Nafi, A. H. Aman, S. Islam, and M. S. Nahi, "Optimization Technique for Deep Learning Methodology on Power Side Channel Attacks," in *2023 33rd International Telecommunication Networks and Applications Conference*, 2023, pp. 80–83.

[3] D. Bae, J. Hwang, and J. Ha, "Breaking a masked aes implementation using a deep learning-based attack," in *Proceedings of the 2020 ACM International Conference on Intelligent Computing and its Emerging Applications*, 2020, pp. 1–5.

[4] A. A. Ahmed, M. K. Hasan, N. S. Nafi, A. H. Aman, S. Islam, and S. A. Fadhil, "Design of Lightweight Cryptography based Deep Learning Model for Side Channel Attacks," in *2023 33rd International Telecommunication Networks and Applications Conference*, 2023, pp. 325–328.

[5] S. Picek, G. Perin, L. Mariot, L. Wu, and L. Batina, "Sok: Deep learning-based physical side-channel analysis," *ACM Comput. Surv.*, vol. 55, no. 11, pp. 1–35, 2023.

[6] A. A. Ahmed *et al.*, "Detection of Crucial Power Side Channel Data Leakage in Neural Networks," in *2023 33rd International Telecommunication Networks and Applications Conference*, 2023, pp. 57–62.

[7] A. Rădulescu and M. O. Choudary, "Side-Channel Attacks on Masked Bitsliced Implementations of AES," *Cryptography*, vol. 6, no. 3, p. 31, 2022.

[8] A. A. Ahmed and M. K. Hasan, "Design and Implementation of Side Channel Attack Based on Deep Learning LSTM," in *2023 IEEE Region 10 Symposium (TENSYMP)*, 2023, pp. 1–6.

[9] R. Gilmore, N. Hanley, and M. O'Neill, "Neural network based attack on a masked implementation of AES," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2015, pp. 106–111.

[10] A. A. Ahmed, R. A. Salim, and M. K. Hasan, "Deep Learning Method for Power Side-Channel Analysis on Chip Leakages," *Elektron. ir Elektrotechnika*, vol. 29, no. 6, pp. 50–57, 2023.

[11] P. Socha, V. Miškovský, and M. Novotný, "A comprehensive survey on the non-invasive passive side-channel analysis," *Sensors*, vol. 22, no. 21, p. 8096, 2022.

[12] A. A. Ahmed, M. K. Hasan, S. Islam, A. H. M. Aman, and N. Safie, "Design of Convolutional Neural Networks Architecture for Non-Profiled Side-Channel Attack Detection," *Elektron. Ir Elektrotechnika*, vol. 29, no. 4, pp. 76–81, 2023.

[13] N. Do, V. Hoang, V. S. Doan, and C. Pham, "On the performance of non-profiled side channel attacks based

on deep learning techniques," *IET Inf. Secur.*, vol. 17, no. 3, pp. 377–393, 2023.

[14] M. Randolph and W. Diehl, "Power side-channel attack analysis: A review of 20 years of study for the layman," *Cryptography*, vol. 4, no. 2, p. 15, 2020.

[15] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: a first study," *J. Cryptogr. Eng.*, vol. 1, no. 4, pp. 293–302, 2011.

[16] K. Pu, H. Dang, F. Kong, J. Zhang, and W. Wang, "A Quantitative Analysis of Non-Profiled Side-Channel Attacks Based on Attention Mechanism," *Electronics*, vol. 12, no. 15, p. 3279, 2023.

[17] J. Ming, Y. Zhou, H. Li, and Q. Zhang, "A secure and highly efficient first-order masking scheme for AES linear operations," *Cybersecurity*, vol. 4, pp. 1–15, 2021.

[18] S. Jin, P. Johansson, H. Kim, and S. Hong, "Enhancing time-frequency analysis with zero-mean preprocessing," *Sensors*, vol. 22, no. 7, p. 2477, 2022.

[19] S. R. Shanmugham and S. Paramasivam, "Power analysis attack resilient block cipher implementation based on 1-of-4 data encoding," *ETRI J.*, vol. 43, no. 4, pp. 746–757, 2021.

[20] F. Kenarangi and I. Partin-Vaisband, "Security Network On-Chip for Mitigating Side-Channel Attacks," in *2019 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*, 2019, pp. 1–6.