Explain Some Noise: Ablation Analysis for Deep Learning-based Physical Side-channel Analysis

Lichao Wu
Delft University of Technology
Delft, The Netherlands
l.wu-4@tudelft.nl

Yoo-Seung Won
Nanyang Technological University
Singapore, Singapore
yooseung.won@ntu.edu.sg

Dirmanto Jap
Nanyang Technological University
Singapore, Singapore
djap@ntu.edu.sg

Guilherme Perin
Delft University of Technology
Delft, The Netherlands
guilherme.perin@tudelft.nl

Shivam Bhasin
Nanyang Technological University
Singapore, Singapore
sbhasin@ntu.edu.sg

Stjepan Picek
Delft University of Technology
Delft, The Netherlands
picek.stjepan@gmail.com

***Abstract*—Deep learning-based side-channel analysis represents a powerful option for profiling attacks on power and electromagnetic leakages as it breaks targets protected with countermeasures. While most of the papers report successful results, it is not difficult to find cases where deep learning works better or worse, especially concerning various countermeasures. Current approaches concentrate on various data augmentations or hyperparameter tuning options to make the attacks more powerful. At the same time, understanding what makes an attack difficult has received very little attention.**

This paper proposes a side-channel analysis methodology based on the ablation paradigm to explain how neural networks process countermeasures. Our results show that an ablation is a powerful tool as it allows to understand 1) in which layers various countermeasures are processed, 2) whether it is possible to use smaller neural network architectures without performance penalties, and 3) how to redesign neural networks to improve the attack performance when the results indicate that the target cannot be broken. By using the ablation-based approach, we manage to mount more powerful attacks or use simpler neural networks without any attack performance penalties. We hope this is just the first of the works in the direction of countermeasure explainability for deep learning-based side-channel analysis.

I. Introduction

With an increasing market of embedded devices and its ever-increasing security concerns and vulnerabilities, the demand for certified products has gone up. This leads to thousands of products undergoing strict security evaluations in evaluation laboratories around the world on a daily basis [ALAM19]. Side-channel attack (SCA [KJJ99]) is one such threat against which embedded devices are regularly evaluated. This work focuses on power/electromagnetic (EM) side-channel attacks targeting secret key recovery from cryptographic algorithms. While there is a range of attacks that can be mounted on an embedded device, profiling SCA [CRR02] remains a preferred one as it provides worst-case guarantees under a supervised learning paradigm.

In only a few years, deep learning got established as a preferred option for profiling side-channel analysis (SCA) [MPP16]. The results show that such techniques manage to break diverse targets where many are protected with state-of-the-art protection mechanisms like hiding and masking [ZBHV19], [WP20]. More recently, we see a steady continuation of a trend where researchers manage to find smaller and shallower neural networks that perform well. At the same time, our understanding of how and why the deep learning-based attacks work does not improve. As such, the research directions like explainability and interpretability for deep learning should represent the major goal for future research.

Unfortunately, such directions are difficult to explore and even more difficult to offer conclusive findings. Those difficulties are not unique for SCA. Indeed, explainability and interpretability are questions heavily researched in other domains (e.g., image classification [SWM17], [CLB+18], [AB18]) but without strong success or definitive directions to follow. In 2017, DARPA launched a four-year program on explainable artificial intelligence (XAI) [GA19] to investigate how XAI can improve the understanding, trust, and performance of AI systems. As there are no general findings, it is difficult to expect that the security community can solve these problems for a specific domain like the profiling SCA. Still, as the neural networks used in profiling SCA are not very deep (compared to neural networks used in other domains) and with a trend to become even more shallow (and narrow) [ZBHV19], [WAGP20], there is hope that it could be easier to understand such neural networks. Finally, we note that SCA deals with a commonly neglected phenomenon in other domains - noise. In other domains, the noise comes from the environment or uncertainty of the process, but the researchers aim to remove it. In SCA, the task is to conduct a successful attack despite the noise generated by the environment or countermeasures. Consequently, we must be more interested in how machine learning algorithm processes noise/countermeasures and reduces their effect on the final results.

This leads to the question, why is it even important to explain neural networks? Especially if they work well, which is a (relatively) common case in profiling SCA. There are multiple reasons: 1) if the neural networks work well, we want

to understand why to make them even more powerful, 2) if the neural networks do not work well, we want to understand what is the problem and how to resolve it, and 3) when the neural networks work well, we want to understand what represents the main difficulties for them to design better SCA countermeasures.

There are not many results in SCA going in this direction. For example, most of the effort up to now went into feature visualization [HGG20], [MDP19]. While important and powerful, this is not enough to provide answers to the question above. As such, the SCA community must explore more diverse techniques to improve the state-of-the-art of profiling SCA explainability and interpretability.

Our goal is to provide a methodology enabling a better understanding of neural network inner working. More precisely, we consider the ablation procedure where we "turn off" (ablate) certain parts of a neural network and then observe the impact on such a neural network. By comparing the results before and after ablation, we can pinpoint locations where important SCA information processing happens. Our main contributions are:

1) We propose the methodology to conduct the ablation analysis in profiling SCA.
2) We explore the influence of various types of noise in the presence of ablation. We verify that various countermeasures are not processed equally in all the neural network layers. Additionally, we provide insights into the countermeasures' difficulty for neural networks.
3) We show that even smaller than currently used neural networks can reach top performance in SCA, indicating that the current methodologies can be further improved.
4) Our analysis enables a better understanding of the attack performance and provides insights into how to make the attack more powerful or from where to start to design more resilient countermeasures.
5) We show how ablation can be used to resolve practical issues in SCA like portability, i.e., where the training device and device under attack are different.

The source code for all experiments will be published[1].

II. Background

A. Notation

Calligraphic letters like $\mathcal{X}$ denote sets, and the corresponding upper-case letters X denote random variables and random vectors $\mathbf{X}$ over $\mathcal{X}$. The corresponding lower-case letters x and $\mathbf{x}$ denote realizations of X and $\mathbf{X}$, respectively. k is a key byte candidate that takes its value from the keyspace $\mathcal{K}$, and k^* the correct key byte.

Dataset is as a collection of traces (measurements) $\mathbf{T}$, where each trace $\mathbf{t}_i$ is associated with an input value (plaintext or ciphertext) $\mathbf{d}_i$ and a key $\mathbf{k}_i$. $\boldsymbol{\theta}$ denotes the vector of parameters to be learned in a profiling model (e.g., the weights in neural networks), and $\mathcal{H}$ denotes the set of hyperparameters defining the profiling model.

[1]Blinded for anonymous review

B. Deep Learning and Profiling SCA

Deep learning belongs to a family of machine learning methods based on artificial neural networks with representation learning. Supervised machine (deep) learning deals with the task of learning a function f that maps an input to the output ($f : \mathcal{X} \rightarrow Y$)) based on examples of input-output pairs. The function f is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^n$, where n denotes the number of trainable parameters.

Supervised learning happens in two phases: training and test. This corresponds to profiling SCA, executed in profiling and attack phases. In the rest of this paper, we use the terms profiling/training and attack/testing interchangeably.

1) The goal of the training phase is to learn the parameters $\boldsymbol{\theta}'$ that minimize the empirical risk represented by a loss function on a dataset T of size N ($T = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$). As common in profiling SCA, we consider the c-classification task, where c denotes the number of classes depending on the leakage model we use. More precisely, the classifier is a function that maps input features to label space ($f : \mathcal{X} \rightarrow \mathbb{R}^c$). In the rest of this paper, the function f is a deep neural network with the *Softmax* output layer.
2) In the attack phase, the goal is to make predictions about the classes
$$y(x_1, k^*), \ldots, y(x_Q, k^*),$$
where k^* represents the secret (unknown) key on the device under the attack. The outcome of predicting with a model f on the attack set is a two-dimensional matrix P with dimensions equal to $Q \times c$. The probability $S(k)$ for any key byte candidate k is an SCA distinguisher:
$$S(k) = \sum_{i=1}^{Q} \log(\mathbf{p}_{i,v}). \tag{1}$$
The value $\mathbf{p}_{i,v}$ denotes the probability that for a key k and input d_i, we obtain the class v where class v is derived from the key and input through a cryptographic function and a leakage model.

To assess the attack performance, i.e., the number of measurements required to break a target, it is common to use the guessing entropy (GE) metric [SMY09]. With Q traces in the attack phase, the attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in decreasing order of probability. So, g_1 is the most likely and $g_{|\mathcal{K}|}$ the least likely key candidate. Guessing entropy is the average position of k^* in $\mathbf{g}$. In this work, we calculate partial guessing entropy (i.e., considering a specific key byte), but we denote it as guessing entropy for simplicity.

C. Portability

The strength of profiling SCAs arises from their capability to fully characterize the target device. In literature, most works conduct profiling and test on the same device ("single-device" model). However, in practical scenarios, the presence of external factors, such as process variation or different acquisition methods, may cause the "single-device-model"

attack to fail. These external factors lead to an issue known as portability, subsuming all effects due to different devices and secret information between profiling device and device under attack.

As an example, in [KRKK18], the authors perform a profiling SCA on AES encryption in a wireless keyboard, where they reported 28% success when attacking a keyboard different than the profiling one as opposed to 100% success when profiling and testing on the same keyboard. This issue is popularly known as the portability issue. Though this is a critical issue and evaluation labs face it on a daily basis, not many investigations are conducted in this direction. One of the latest approaches to address portability was presented by Bhasin et al. [BCH+20] at NDSS 2020, by considering the multi-device model (MDM). The idea is to use multiple copies of devices similar to the targeted one for training and validation. Then, the trained model could better generalize the leakage and minimize the risk of overfitting to the training device. MDM was also validated by [DGD+19], [GJS19]. [RBA20] adopts an alternate approach to measure the difference between devices and exploit that to improve profiled SCA. MDM trains the model to learn the similarity and discrepancy by default. Alternately, Zhang et al. [ZSX+20] analyzed the portability issue for heterogeneous devices (clone device is different from victim device), which we consider out of scope. However, one possible drawback is that the evaluator needs to acquire measurements from multiple copies of the same device, which might be either expensive (in time or equipment) or simply not available or both.

D. SCA Countermeasures

To mitigate side-channel attacks, designers deploy a series of countermeasures. The main objective of these countermeasures is to either reduce or ideally remove sensitive leakage from the side-channel traces [MOP07]. This section describes some of the commonly used SCA countermeasures, which we also use in our analysis. All countermeasures are implemented following the open-source code[2].

The first countermeasure is the Gaussian noise addition. This corresponds to unintentional and intentional activity captured in parallel to the sensitive operation, which contributes to the measurement's noise component, directly reducing the signal-to-noise ratio (SNR) of the captured traces. Unintentional contribution caters to non-sensitive parts of the computation, noise from connectors, digital to analog conversion, etc. Dummy operations or dedicated noise generators introduce intentional noise. With a low signal-to-noise ratio (SNR), the attack generally requires more traces to be successful. For our experiments, we add a random value drawn from a normal distribution with zero mean and standard deviations $[1.0, 5.0]$, respectively, to investigate from low to high noise settings.

The second investigated countermeasure is desynchronization. Similar to Gaussian noise, any misalignment in traces reduces the SNR, forcing the attacker to either acquire more traces or execute an additional re-alignment step. While the Gaussian noise is added in the amplitude domain, desynchronization introduces random noise in the time domain. The re-alignment is generally based on signal processing techniques and is a complex operation in itself. It becomes even harder when the traces are noisy, either due to the target device itself or any underlying countermeasure. We introduce up to 5 points of desynchronization of either polarity as it is the maximum value leading to successful attacks in our experiments.

Finally, we study the effect of clock jitter as a countermeasure. The clock signal, which drives all digital circuits, can have some random jitter. This jitter is normally undesirable as it affects the performance and reliability of the underlying circuit. Clock jitters increase the randomness for each point in the time domain. The accumulation of the deforming effect increases the misalignment of the traces and decreases the overall SNR. We simulate the clock jitters by randomly adding or removing points with a pre-defined range. If the generated number is positive, points are added to the trace and otherwise removed. We add jitter of up to a single point. Although this is the smallest value that can be added, it still prevents GE from converging for most of the cases. We discuss this in detail in Section V.

E. Neural Network Architectures

1) Multilayer Perceptron: The multilayer perceptron (MLP) is a feed-forward neural network mapping sets of inputs onto sets of appropriate outputs. MLP consists of multiple layers (at least three) of nodes in a directed graph, with each layer fully connected to the next one.

2) Convolutional Neural Networks: Convolutional neural networks (CNNs) commonly consist of three types of layers: convolutional layers, pooling layers, and fully-connected layers. The convolution layer computes the output of neurons connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. Pooling decrease the number of extracted features by performing a down-sampling operation along the spatial dimensions. The fully-connected layer (the same as in MLP) computes either the hidden activations or the class scores.

F. Related Work

The domain of profiling SCA started in 2002, with the template attack [CRR02]. While this attack is the most powerful one from the information-theoretic perspective, in practice, it suffers from unrealistic assumptions (unlimited number of profiling traces, noise following Gaussian distribution). Then, in the first years of machine learning-based SCA, researchers considered simpler machine learning techniques that commonly performed similar or slightly better than template attack. Common examples of the used approaches are random forest [LMBM13], support vector machines [HGM+11], [HZ12], Naive Bayes [PHG17], and multilayer perceptron [GHO15]. From 2016, the SCA community

[2]Available at https://github.com/AISyLab/Denoising-autoencoder

shifted a large part of its attention to the deep learning techniques [MPP16]. The two most explored approaches are multilayer perceptron (more complex architectures than before) and convolutional neural networks. Both of those approaches reached top performance where it is even possible to break implementations protected with countermeasures [CDP17], [KPH+19]. Only recently, the community started to expand the deep learning perspective for profiling SCA, see, e.g., autoencoders that are used to pre-process the traces to remove the influence of countermeasures [WP20].

As the trend of deep learning in SCA is relatively new (compared to other domains), most works still concentrate on improving the attack performance. For instance, in the beginning, deep learning showed potential but not much more than that [MPP16], [PSK+18]. Next, we can recognize a more mature phase where authors reported strong attack performance, even in the presence of countermeasures [CDP17], [KPH+19]. Finally, more recent results manage to improve the performance even further where interestingly, we see the trend of using smaller (more shallow) deep learning architectures [ZBHV19], [WAGP20], [PCP20].

While far from completed (as the results can be improved even further), one can recognize that deep learning does represent a significant step forward for the profiling SCA. Simultaneously, the SCA community's understanding of what happens during the deep learning-based SCA process (i.e., interpretability and explainability) is much more limited. Interpretability is the degree to which a human can consistently predict the model's result [KKK16]. On the other hand, explainability is the extent to which a machine learning system's internal mechanics can be explained in human terms. Several works consider visualization techniques to find relevant features and improve the interpretability [HGG20], [MDP19]. Van der Valk et al. considered the activation functions in neural networks to explain what neural networks learn while training on different side-channel datasets or even datasets that are not side-channel measurements [vdVPB19].

In this work, we aim to bridge this gap and use a well-established technique from deep learning - ablation to understand how noise and countermeasures are processed in deep learning architectures in SCA.

III. Ablation & SCA

A. Ablation

Ablation is a process long used in neuroscience, where controlled damages are introduced in neural tissue to investigate the impact of damages on the brain's capabilities to perform assigned tasks. This approach provides deep insights and explanations about each part of the tissue's structure and role when reacting to external stimuli [Sch77]. As the complexity of neural networks increased, the explainability of models has become an open question. As a natural extension, ablation is adopted to understand neural networks better. An ablation study investigates the performance of the system by removing certain components to understand the contribution of the component to the system [MLdPM19]. Note that ablation requires that the system show slow degradation, i.e., that the system continues to work even when specific components are missing or reduced. A straightforward way to conduct ablation for neural networks is to set the weights and biases of a particular neuron (therefore, ablating that neuron) to zero. While ablation often considers smaller changes to the system, we will not differentiate it based on the number of components we remove.

There is a connection between ablation and an approach called pruning, which corresponds to the systematic removal of parameters from an existing system [HPTD15]. Pruning in neural networks is commonly done based on the magnitude of the weights or by setting a threshold value. In both cases, the weights contributing the least to the task at hand (e.g., classification) are removed. As we see, both pruning and ablation can conduct the same operation type - removing a part of a system. Still, the difference that we emphasize is in the underlying motive to conduct such an operation. Pruning is a widely used approach to speed up the training and inference while minimizing the impact on the trained network's original performance. On the other hand, ablation reduces trainable parameters to gain insights and interpret the trained network inner working. As we are interested in understanding how neural networks work in profiling SCA and how they deal with various types of countermeasures, we will focus on ablation. Next, we propose a methodology to ablate neural networks used for profiling SCA to understand the neural network's inner functioning better.

B. Proposed Ablation Methodology

Explainability is one of the main issues for deep learning. Within the SCA community, recent results [WP20] indicate that deep learning performance can vary dramatically concerning different types of noise/countermeasures. Unfortunately, the precise reason for triggering the differences in the learning process is unknown. Intuitively, it may relate to how the model adapts to the noise: a naive noise/countermeasure could be easily handled by, for example, a single layer. When the noise increases, more layers may need to be used. Unfortunately, to the best of our knowledge, there is no approach to offer a concrete answer.

To systematically understand how a neural network model behaves before and after adding noise/countermeasures, we follow the ablation methodology presented in Algorithm 1. First, the model is trained with the original dataset. Then, when adding noise to the dataset and re-training the model, the network changes should relate to the noise's addition. The main ablation procedure starts by randomly selecting and ablating ρ% of the neurons/convolution filters. For neurons, we do this by setting their weights and biases to zero. For convolutional filters, we change the convolution filter size, and we recreate the neural network. Note that we randomly select neurons/convolution filters to be ablated as, in general, one does not know what part of the neural network contributes to the final neural network output.

Next, we calculate GE when ablating each layer independently. We also record the weights for each layer. The ablated model is re-trained for τ epochs to adjust the weights. We use re-training as there was a change in the neural network model, and we must allow it some additional training time to adjust for the changes in the architecture. Finally, we calculate GE to evaluate the recovery capability of the model.

Note that the ablation is performed in a layer-wise manner instead of a neuron/convolution filter manner. The reason is that each neuron/convolution filter's contribution can fluctuate due to the random weight initialization. As a result, it is difficult to reach a consistent conclusion when one repeats the proposed ablation methodology with a different pre-trained model. Differing from that, by ablating the network layer-wise, we can systematically study the ablation and countermeasure effect. Since the neurons/convolution filters are randomly selected, the ablation procedure is repeated σ times to obtain more reliable results. Following the methodology, the number of tests to be performed is positively correlated to the model depth. Fortunately, since the model is pre-trained, the recovery training can be efficient. Assuming the recovery training for one model requires two minutes, we require on average two and a half hours to finish the ablation procedure. [3]

Algorithm 1 SCA Ablation Methodology.

procedure ABLATE(original dataset T, countermeasure level (intensity) γ, repeat time σ, ablate rate ρ, recovery epoch τ, ablated layer l)
  $M_{pre} \leftarrow$ Pre-train Model with T
  $T_\gamma \leftarrow$ Add countermeasure with level γ
  for i = 0 to σ **do**
    if i == 0 **then** ▷ No ablation: for ref. GE and weight calculation (with T_γ)
      ρ = 0
    end if
    $M^{\rho}_{pre}, W^{\rho,i}_{pre} \leftarrow$ Ablate(M_{pre}) ▷ Remove ρ% of neurons/filters from layer l
    $GE^{\rho,i}_{pre} \leftarrow$ Attack(M^{ρ}_{pre})
    $M^{\rho}_{abl}, W^{\rho,i}_{abl} \leftarrow$ Train(M^{ρ}_{pre}) ▷ Train with T_γ for τ epochs
    $GE^{\rho,i}_{abl} \leftarrow$ Attack(M^{ρ}_{abl})
  end for
  $W_\rho \leftarrow \frac{1}{\sigma}\sum_{i=0}^{\sigma}(W^{\rho,i}_{pre}$ - $W^{\rho,i}_{abl})$
  if i > 0 **then**
    $GE^{\rho,i}_{pre} \leftarrow GE^{\rho,0}_{pre}$ - $GE^{\rho,i}_{pre}$
    $GE^{\rho,i}_{abl} \leftarrow GE^{\rho,0}_{abl}$ - $GE^{\rho,i}_{abl}$
  end if
end procedure

[3]This is naturally more computationally expensive than calculating GE for the original model only, we consider the information obtained through ablation more than justifying the extra computational time.

IV. Experimental Setup

A. Threat Model

We consider a common profiling side-channel setting focusing on power/EM side-channel attacks targeting secret key recovery from cryptographic algorithms. This model is a standard model as numerous certification laboratories evaluate hundreds of security-critical products under this model daily.

We assume an adversary with access to a clone device running the target cryptographic algorithm, normally on an embedded device. This clone device can be queried with known/chosen parameters (keys, plaintext, etc.) while the corresponding leakage measurements, like power or electromagnetic emanation, are recorded. Ideally, the adversary can make infinite queries and build a corresponding database of side-channel leakage measurements to build a precise profiling model. This forms the profiling phase.

Next, the adversary queries the device under attack with known plaintext to recover the secret key by querying the characterized model with corresponding side-channel leakage traces. This represents the attack phase. We investigate both single device setup where the measurements in both phases are done on the same device and portability setup where the clone device and the device under attack differ.

B. General Setup

Note that while one could ablate the neurons/convolution filters for any percentage, we give results for three levels: $\rho = \{10, 50, 90\}$, to investigate the behavior of neural networks for various settings (i.e., when we do a small change, medium change, or a large change to the neural network architecture). We run the recovery training for $\tau = \{10\}$ epochs as the models are pre-trained, and they do not require a long time to adapt to the changes in the network architectures due to the ablation procedure. GE is calculated over 100 independent experiments to obtain statistically significant results. Finally, GE and weight variation presented in the experiments are averaged over $\sigma = \{5\}$ ablation experiments. All of the experiments are implemented with the TensorFlow [AAB+15] computing framework and Keras deep learning framework [C+15].

C. Datasets

We consider two popular datasets widely adopted in SCA research: ASCAD with the fixed key and ASCAD with random keys[4]. The measurements are obtained from an 8-bit AVR microcontroller running an AES-128 implementation, where the side-channel is electromagnetic emanation [BPS+20]. Both datasets have masking countermeasure, making them more difficult to attack and closer to reality. Note that the ASCAD with random keys dataset uses different keys for training and attack, but the corresponding measurements are obtained from the same device. As most of the profiling attacks would record traces with random keys during the profiling phase while attacking the traces with an unknown fixed key, ASCAD with random keys would simulate the real-world setting better [5].

[4]https://github.com/ANSSI-FR/ASCAD

[5]While still not being realistic as not considering portability.

1) ASCAD with Fixed Key Dataset: This version of the ASCAD dataset has 50 000 traces for profiling and 10 000 traces for the attack. 5 000 traces from the profiling set are used for validation. We use a pre-selected window of 700 features for the side-channel trace, and we attack key byte 3, which is the first masked key byte (as recommended by the dataset).

2) ASCAD with Random Keys Dataset: The second ASCAD version has random keys, and the dataset consists of 200 000 traces for profiling and 100 000 traces for the attack. We use 5 000 traces from the attack set for validation. We use a pre-selected window of 1 400 features for this dataset and attack key byte 3 (the first masked key byte).

D. Attack Architectures

In Table I, we depict the neural network architecture hyperparameters used. Here, four models with different complexities, 'simple' MLP (MLP_s) [BPS$^+$20], 'complex' MLP (MLP_c) [WPP20], 'simple' CNN (CNN_s) [ZBHV19], and 'complex' CNN CNN_c [BPS$^+$20] are designed based on related works and used for evaluation. By evaluating those models, we can better understand the noise processing in different conditions. The hyperparameter selection is adapted based on the dataset being tested. The experiments are conducted under the Hamming weight (HW) and the identity (ID) leakage settings.

Note that the difference in the number of trainable parameters for MLP is not so significant as for CNN. Indeed, MLP_c is around 30% larger than MLP_s. At the same time, CNN_c is three orders of magnitude larger than CNN_s. Such a large difference comes from significantly larger fully-connected layers and primarily the increase in the size and number of convolution layers.

Network	Architecture	lr / Epochs	Batch Size
MLP_s (HW)	Dense(200)*5	1e-4 / 100	100
MLP_s (ID)	-	3e-5 / 200	100
MLP_c (HW)	Dense(200)*8	1e-4 / 100	100
MLP_c (ID)	-	3e-5 / 200	100
CNN_s (HW)	Conv(4)+Dense(10)*2	5e-4 / 100	100
CNN_s (ID)	-	1e-6 / 200	100
CNN_c (HW)	Conv(64,128,256,512,512) +Dense(1 024)*2	1e-4 / 75	200
CNN_c (ID)	-	1e-4 / 75	200

TABLE I: Baseline deep learning architectures.

V. Experimental Results

The results presented here represent only a small part of the conducted experiments. Additional results are located in Appendix A. Since our experiments show that CNN_s is giving similar results to other considered architectures, we show the corresponding figures in the appendix. Furthermore, we omit experiments where the results indicate that the model did not fit the data as in that case, ablation can give only limited information (as showcased for the ASCAD with the fixed key scenario and discussed in Section V-A).

Recall, all the experiments are done layer-wise, meaning that a single experiment ablates one layer only. To conserve space, we depict results for all layers in a single figure (still, experiments are independent, so each column should be considered independently). We depict our results with two types of graphs: one that shows the GE difference concerning various layers and one that depicts weight difference for the various layers. Note that we give the average performance for all possible values of the attack traces.

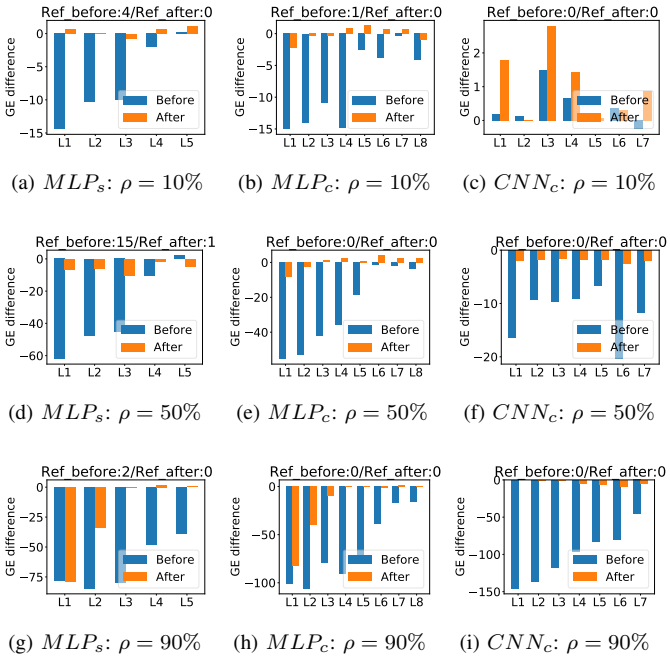
The figure title provides basic information on the neural network performance without ablation for the GE difference figures. The value "Ref_before" denotes the GE value for the reference model that is trained with the original version of the dataset and then used to attack the dataset where the noise is added, while the value "Ref_after" denotes the GE value for the reference neural network after re-training (which, for the reference model means additional training for τ epochs). The differences between the reference model's attack performance (observing results per columns) as we randomly select attack traces for each experiment which causes some variation in the attack performance. In each sub-figure, we depict the results where we compare "Ref_before" with the ablated network that is not re-trained ("Before") and "Ref_after" with the ablated network after the re-training phase ("After"). Values less than 0 denote that the original network ("Ref_before" or 'Ref_after") performed better, while the values larger than 0 denote that the ablated network (with or without re-training) performed better.

Additionally, we show the neural network weight differences for each layer, comparing the weight values before and after the re-training procedure. Note that the neural networks are ablated in both cases. As an example, consider Figure 2 (a) where the value i on the x-axis represents the differences in weights connecting $layer_i$ and $layer_{i+1}$. When i equals zero, the weights between the input layer and the first hidden layer are averaged and compared. When i equals five, we process the weight shared by the last hidden layer and the output layer.

A. Results for the ASCAD with the Fixed Key Dataset

Figure 1 presents results for the GE differences when considering Gaussian noise with a standard deviation of 1. Additional results for CNN_s are given in Appendix, Figure 12. Observe from the results that both "Ref_before" and "Ref_after" show very good behavior (GE of 0 or close to 0). This means that the training process is sufficiently long, and it is easy for a neural network to adapt to changes in the test set if those changes come in the form of moderate Gaussian noise. Somewhat larger performance variation happens for MLP_s, as short re-training is required to adapt to changes due to the test set's Gaussian noise and a smaller network capacity. Stated differently, small network capacity allowed the original network to model the data. However, that modeling did not include any robustness to small differences.

Gaussian noise causes more significant changes for MLP architectures in the first layers, indicating those layers deal



(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_c: $\rho = 10\%$

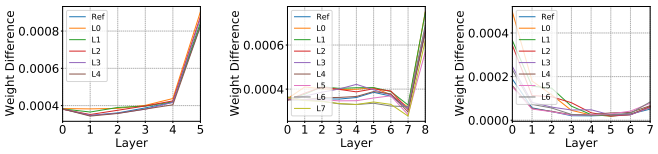
(d) MLP_s: $\rho = 50\%$ (e) MLP_c: $\rho = 50\%$ (f) CNN_c: $\rho = 50\%$

(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_c: $\rho = 90\%$

Fig. 1: **Guessing entropy difference before and after the recovery training: Gaussian noise** $(\gamma, \sigma, \rho, \tau) = (1.0, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key

with the noise. Even significant ablation rates do not cause large final performance changes, implying we can use smaller MLPs. Thus, a designer who wants to optimize the MLP architectures should remove the neurons from the final layers of MLP. What is more, GE can be even better after ablation (see, e.g., Figures 1a and 1e), suggesting that too large model capacity caused neural network model to overfit even in the presence of Gaussian noise. For CNN_s, the noise similarly influences fully-connected layers as for MLP. After the re-training, the convolution layer does not show much change due to ablation, meaning that Gaussian noise is not processed in any special way in that layer. These results indicate that strong CNN_s performance in the presence of Gaussian noise comes from implicit feature selection and not spatial invariance (which is to be expected as there is no misalignment). For CNN_c, we see improvement after ablation if the ablation rate is low, while the difference is small to negligible for larger ablation rates. Note while it seems there are significant GE changes in the beginning layers for $\rho = 10\%$, the scale is different, so the changes are actually small.

In Figure 2, we depict the weight differences for three considered neural networks (other cases are in Appendix, Figure 13). For MLP architectures, especially when considering smaller ρ, there are changes in all but the last layer. This means that the neural network adapts to smaller architectures due to ablation and has a similar attack performance. Somewhat larger changes happen at the end, but this is expected as this is the neural network part building the final probabilities, and we expect those to change whenever we change the neural network architecture. Observe larger weight differences in the layer after the ablation (see, e.g., Figure 13 (d)), indicating the next layer adjusting to the new architecture. For MLP_c, the penultimate layer commonly has very small weight differences, suggesting once more that the architectures are larger than required (i.e., that specific layer does not do much useful information processing). For CNN_s, the largest differences happen in the convolution layer, which means that feature selection adapts for noise, but the classification layers can process the information in a very similar manner. For large ablation rates, the biggest change is in the layer where the ablation happened. Considering the drop in the weight differences for subsequent layers (see, e.g., Figure 13 (f)), we can conclude that the neural network model still has enough capacity to adapt. Finally, for CNN_c, the largest weight differences happen in the beginning (in line for results for CNN_s, but we also observe some more activity in the fully-connected layers as they adjust to the ablated layers.

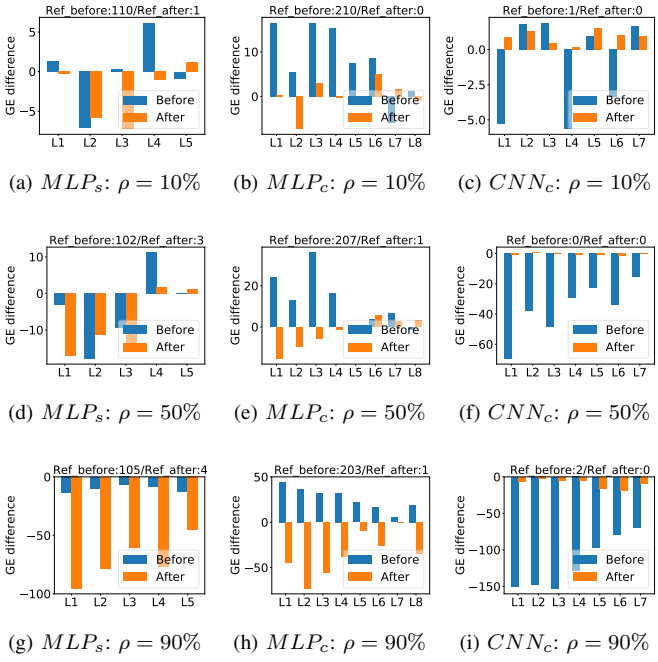


(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 50\%$ (c) CNN_c: $\rho = 90\%$

Fig. 2: **Weight difference before and after the recovery training: Gaussian noise** $(\gamma, \sigma, \rho, \tau) = (1.0, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key.

Next, we consider the desynchronization countermeasure (Figure 3), the results for CNN_s are listed in Figure 14 in Appendix. The "Ref_before" result implies that desynchronization represents a more difficult countermeasure to adapt to, and the original neural network model cannot adapt to this countermeasure in the attack set. Still, even short re-training shows a significant performance improvement ("Ref_after"). Thus, the network has more difficulty dealing with desynchronization than Gaussian noise if not trained on it. At the same time, short re-training allows neural networks to adapt to this countermeasure.

The GE improvements that can be seen for the "Before" case are not relevant. Indeed, the performance of "Ref_before" is similar to random guessing, so somewhat better GE does not mean we can break the target. However, the results for the "After" case are much more important. First, we notice that MLP architectures could be reduced to a certain degree. If the reduction happens in the deeper layers, the performance can even improve, while ablation in the first layers causes performance degradation. It is interesting to note that changes are more significant for a larger part of the network (cf. Figure 3), which means that MLP requires more layers to deal with the desynchronization countermeasure. For CNN_s



(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_c: $\rho = 10\%$

(d) MLP_s: $\rho = 50\%$ (e) MLP_c: $\rho = 50\%$ (f) CNN_c: $\rho = 50\%$

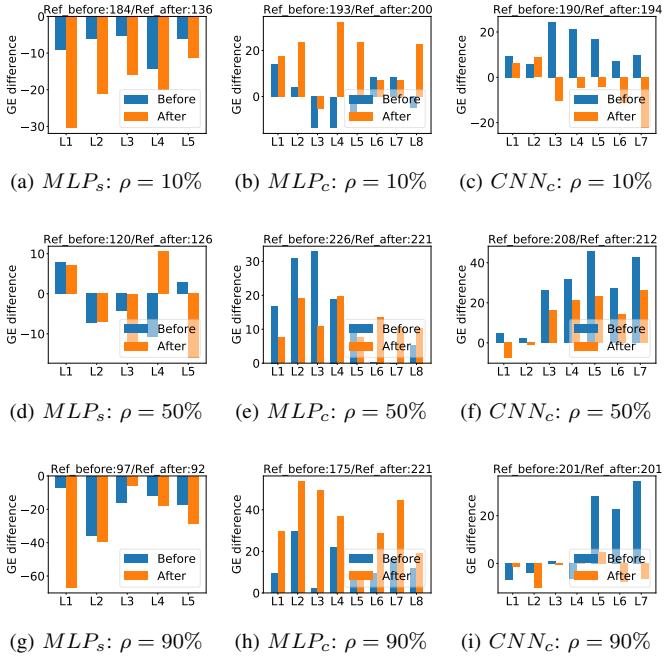
(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_c: $\rho = 90\%$

Fig. 3: **Guessing entropy difference before and after the recovery training: Desynchronization** $(\gamma, \sigma, \rho, \tau) = (5, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key.

(Figure 14 in Appendix), ablating the convolution layer causes large differences, which means that CNN_s's spatial invariance is the determining factor for its success when dealing with desynchronization. The changes for the fully-connected layers are consistent with a small MLP (as there are only two dense layers in CNN_s). For CNN_c, ablation causes very small changes in several layers, indicating that the network capacity is more than enough, so various changes are easily adjusted for with the rest of the architecture.

Figure 15 in Appendix presents the weight differences for the desynchronization scenario. The results for MLP show small changes in weights if the ablation rate is lower and the MLP architecture is smaller. This suggests that a small neural network does not have many options on building a model, so small changes do not affect the performance as we basically build the same neural network model (minus the neurons that are not needed as we already noticed there is some redundancy). Somewhat bigger changes happen for MLP_c as this model has more capacity, allowing a better fit to data. Removing parts of the network require more weight changes as now, there are fewer options to fit the data in an equally expressive manner. For CNN_s, the largest differences happen in the convolution layer. This again confirms the importance of a convolution when dealing with a countermeasure working in the time domain, like desynchronization. Additionally, we see significant changes in the layer where ablation happened, indicating there was not enough capacity to use the rest of that specific layer to model the data in the same way. For CNN_c, we observe small weight differences in layers, showcasing that the neural network has more than enough capacity to model the desynchronization countermeasure, resulting in easy adaptation to ablation. At the same time, this also means we can reduce the size of the network significantly and maintain the performance level.

Finally, in Figure 4, we depict the results for the clock jitter (results for CNN_s are in Appendix, Figure 16). Now, both "Ref_before" and "Ref_after" give poor GE values. This means that the network did not manage to learn a model that fits data or adapt to the test set changes. As such, the results are more difficult to explain since there are no successful attacks. We can notice that all the MLP layers deal with the countermeasure as significant changes are happening throughout all the layers of both MLP and CNN. Interestingly, while the results for CNN_s indicate we require more layers, even CNN_c does not show good attack performance. Combined with the fact that there is a significant activity for deeper layers, this indicates we need even larger architectures for this dataset.



(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_c: $\rho = 10\%$

(d) MLP_s: $\rho = 50\%$ (e) MLP_c: $\rho = 50\%$ (f) CNN_c: $\rho = 50\%$

(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_c: $\rho = 90\%$

Fig. 4: **Guessing entropy difference before and after the recovery training: Clock Jitter** $(\gamma, \sigma, \rho, \tau) = (1, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key.

The weight differences (Figure 17) become more significant for deeper MLP layers compared to previous countermeasures. This shows that clock jitter significantly influences the layers just before the classification part. For CNN_s, the convolution layer changes are the most dramatic ones, which means that this is the layer dealing the most with the countermeasure.

The ASCAD dataset results with a fixed key and the ID leakage model are located in Appendix A. For Gaussian

noise (Figures 18 and 19), the results are in line with the results for the HW leakage model. Again, the neural network model capacity is more than sufficient as even much smaller MLP architectures behave similarly. Additionally, we observe a larger influence of the countermeasure, implying that more classes make the classification problem more difficult (as expected). For CNN_s, we observe a minimal influence of the convolution layer. Interestingly, the weight differences are now much more pronounced, which means that they are more spread over a larger number of classes. Then, ablating any part of the network causes larger changes in the weights. The ID leakage model's desynchronization results (Figures 20 and 21) confirm a combined effect of more classes and a more difficult countermeasure, resulting in more MLP layers dealing with the countermeasure. For CNN_s, we observe a significantly smaller influence of the convolution layer. We postulate this happens as features discriminate more for 256 classes than for nine classes, and we require a less powerful convolution layer to select the most discriminative features. Finally, for the clock jitter countermeasure, there is an activity in all layers, again showing that we require, in general, larger neural networks to deal with such a complex countermeasure. As the GE results indicate we cannot break the target, we omit the results.

B. Results for the ASCAD with Random Keys Dataset

Aligned with the ablation experiment performed for ASCAD with a fixed key, three types of noise, Gaussian noise (standard deviation of 1 or 5), desynchronization (max=5), and clock jitter (max=1), are added to ASCAD with random keys. The most representative results are given in this section, while the rest is located in Appendix B. Note that we do not present results for the ID leakage model as the results were not good, i.e., the attacks did not work. The conclusion that can be obtained in that case is similar to the clock jitter scenario for ASCAD with a fixed key.

For Gaussian noise with standard deviation of 1, the representative results are shown in Figure 5, while other GE difference results are listed in Figure 22 in Appendix B. According to "Ref_before" and "Ref_after", the trained neural network easily adapts to changes in the test set as the Gaussian noise has limited influence on the neural network models' performance.

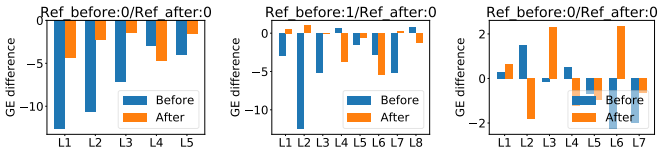
In terms of GE variation before re-training, the effect of ρ on the model's attack performance is significant: a higher ablation ratio ρ leads to a larger GE variation before training. Similar to the results for ASCAD with a fixed key, ablation on the shallower MLP layers causes more damage to the model than the deeper layer. Even 90% of the ablation results in a limited performance degradation (Figures 5g and 5h). Deeper layers have more capability to handle the ablation effect but there is also more redundancy. After the recovery training, GE increased significantly in most cases, while the model's recovery capability varies when ablating different layers. For MLP, ablating the shallower layers, as shown in Figure 5g and 5h, results in the performance degradation, although the recovery training reduces the GE variation compared to the non-ablated models. When the deeper layers are ablated, the GE performance is similar or even better than the reference model. The likely reason for the performance improvement is that the original model is overfitted.

For CNN_s, the GE variation between the reference and the ablated model increased after the recovery training (i.e., Figure 22c), indicating that the ablation reduces the model's capability in feature extraction and data classification. While previous results indicate that the convolution layer does not play an important role in dealing with Gaussian noise, we observe a different behavior in this case. This happens due to a very small neural network model and more variation in the dataset due to random keys. Increasing the model size again confirms our previous results that the convolution layer does not have the central role for dealing with Gaussian noise. Indeed, for CNN_c, the same ablation ratio ρ causes almost no performance drop. For dense layers in CNN_c, we see that ablating can improve the performance, which means the neural network is too complex for the task, and it can overfit.

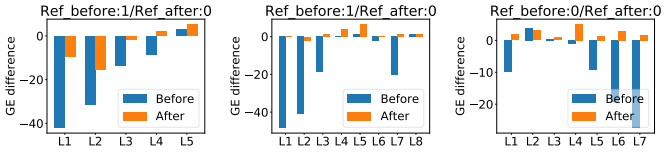
The weight differences results are given in Figure 23 in Appendix. Similar to the observations for ASCAD with a fixed key: with an increasing ablation ratio ρ, the overall weight variation increases no matter what layer is ablated. For MLP, the biggest changes occur in the deeper layer (Figures 23k and 23l). Following this, a naive MLP design choice would be to remove neurons in the deeper layers [Wei20]. For MLP_c, the small weight differences in the penultimate layer again indicate that the architecture is larger than required. For CNN, the weight variation is more concentrated in the shallower layers. Interestingly, for CNN_c when we ablate deeper layers (i.e., L4/L5 in Figure 23l), the weight variation has almost no changes compared to the reference, which indicates the redundancy in those layers.

Next, we introduce desynchronization to the dataset (Figure 6, the rest of the results are in Figure 24 in appendix. Differing from the Gaussian noise scenario, desynchronization adds noise to the horizontal level, making the dataset more difficult for classification. Similar to the ASCAD with the fixed key scenario, the profiling model cannot adapt to desynchronization without the recovery training, but having ten epochs is more than sufficient for neural networks to adapt.

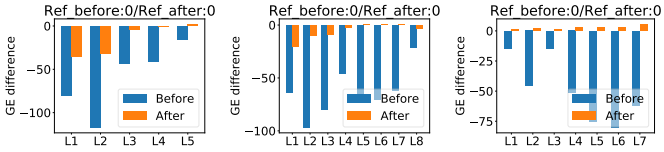
For MLP architectures, more layers are involved in this countermeasure than for Gaussian noise, and ablating the neurons reduces the performance, which means that the network needs the original capacity for good results. For CNN architectures, there is an interesting behavior. For CNN_s, GE is poor, indicating that the attack does not work, but the results after ablation and re-training significantly improve compared to original results ("Ref_before" vs. "Ref_after"). At the same time, ablating specific layers does not deteriorate the performance significantly. This indicates that those layers can be reduced in size, but we require additional layers. Consequently, CNN_c shows much better attack performance. With this new profiling model with a significantly higher capacity, the convolution filters can be smaller, but we need to keep sufficiently large dense layers (still, lower ablation rates



(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_c: $\rho = 10\%$

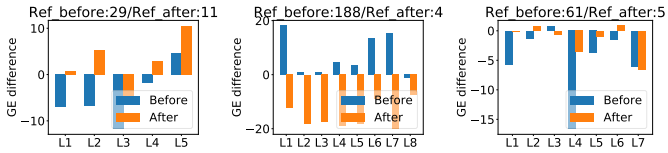


(d) MLP_s: $\rho = 50\%$ (e) MLP_c: $\rho = 50\%$ (f) CNN_c: $\rho = 50\%$

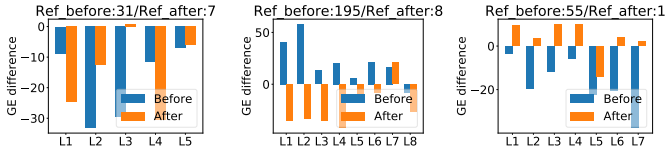


(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_c: $\rho = 90\%$

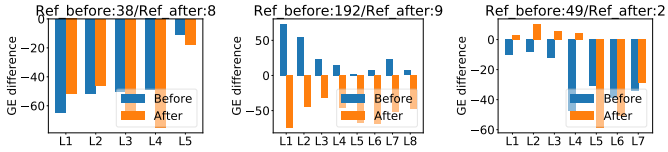
Fig. 5: **Guessing entropy difference before and after the recovery training: Gaussian noise** $(\gamma, \sigma, \rho, \tau) = (1, 10, (10\%, 50\%), 10)$ for the HW leakage model on ASCAD with random keys.



(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_c: $\rho = 10\%$



(d) MLP_s: $\rho = 50\%$ (e) MLP_c: $\rho = 50\%$ (f) CNN_c: $\rho = 50\%$



(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_c: $\rho = 90\%$

Fig. 6: **Guessing entropy before and after the recovery training: Desynchronization** $(\gamma, \sigma, \rho, \tau) = (5, 10, 90\%, 10)$ for the HW leakage model on ASCAD with random keys.

are fine and can even improve the attack performance.

Figure 25 shows the weight differences for the desynchronization countermeasure. For MLP, aligned with the observation discussed before, weight changes are correlated with the ablation rate and architecture size. Moreover, compared with Gaussian noise, desynchronization is more difficult to deal with, indicating that more layers would be involved in adapting to this type of noise. For CNN_s, the convolution layer's weight differences are small, suggesting that the layer is not powerful enough for processing the countermeasure. For CNN_c, the biggest weight changes are in the convolution layers, confirming that desynchronization is best resolved with the convolution layer and that several convolution layers may be needed. The observation again confirms the importance of a convolution when dealing with a countermeasure working in the time domain, as is the case for desynchronization.

Finally, we consider the clock jitter scenario. Similar to the previous results for clock jitter, both "Ref_before" and "Ref_after" give poor attack performance, which means that the network failed to extract meaningful features from the data. This is also confirmed by observing weight differences as there are only minimal changes, which means that the neural networks did not learn to fit the data. Since the results are less informative, ablation does not provide many insights.

C. What Could We Explain?

Based on the extensive investigations testing different datasets, neural networks, leakage models, and countermeasures on previous sections, spanning several hundred experiments conducted in the last subsections, we learn the following key takeaways about deep learning-based profiling SCA. Firstly, Gaussian noise is easier to handle for a deep neural network as a countermeasure when compared to desynchronization and clock jitter, clock jitter being the hardest. As a result, the latter two countermeasures need a deeper network, while Gaussian noise is learned in the initial layers allowing the use of rather shallower networks. In CNNs, the convolution mostly handles desynchronization while having minimal impact on Gaussian noise. Moreover, neural networks struggle with countermeasure when the dataset has more label classes, as for both the datasets, the imbalanced HW dataset (following binomial distribution) was easier to handle as compared to the balanced ID dataset. Lastly, with the weight differences, we see that the impact of ablation is mostly handled by the layer immediately following the ablated layer.

VI. Application to the Multiple Device Model

Next, we propose another application of ablation in tackling portability issues for the profiling SCA, based on learned lessons on the explainability of neural networks from the previous section. While the adoption of the Multiple Device Model (MDM) was proposed as a practical solution to portability in [BCH+20] (i.e., train and validate on multiple copies of training device rather than just one), the availability of multiple copies of a device remains a practical constraint. In fact, the availability of multiple devices is a scoring criterion in

common criteria evaluations [Lom16]. A worst-case adversary assumes the availability of multiple copies of the device.

This study's main goal is to eliminate/mitigate the multiple device assumptions but still generalize and address the portability issue while achieving the same or similar performance as MDM, thus performing a worst-case analysis. We propose *Multiple Device Model from Single Device (MDMSD)*.

It was hypothesized in [BCH$^+$20] that portability could be seen as additive Gaussian noise, but this hypothesis was never validated empirically. We validate this hypothesis through ablation. In particular, we show that ablating layers in the case of portability have similar behavior to the Gaussian noise countermeasure in the previous section. We exploit this understanding of the profiling model to bridge the gap between the single device model and MDM with our proposed MDMSD.

A. Datasets

The previously used datasets provide training and attack data from the same device, thus not considering portability issues. Next, we consider two portability-specific datasets. The detailed setup for the datasets is summarised in Table II.

1) [BCH$^+$20] dataset: The dataset contains measurements from four copies of the target, AVR Atmega328p 8-bit microcontroller, set up in parallel. It measured 50 000 power side-channel traces corresponding to 50 000 random plaintexts. The trace comprised of 600 sample points (features), containing only the execution of the first SubBytes operation of an unprotected AES-128, i.e., 16 S-box look-up, where each S-box is an 8-bit input to an 8-bit output nonlinear mapping. The dataset was then collected based on the measurements from four different boards (B1, B2, B3, B4) with three randomly chosen secret fixed keys (K1, K2, K3).

2) [oCHS18] dataset: This dataset refers to the CHES Capture-the-flag (CTF) AES-128 trace set running on an STM32 microcontroller, released in 2018 [oCHS18]. It consists of different sets of power traces of masked AES-128, with 650 000 sample points per trace. To accelerate our experiments, we focus on a relevant window of 600 points only. The first four sets contained 10 000 power traces. The first three sets (Set 1 to 3) were collected from three different devices (denoted by A, B, C), and each trace corresponds to encryption with a randomly chosen key. Set 4 contains power traces from Device C, with a single fixed key (K4). Set 5 contained 1 000 power traces collected from device C with a fixed key K5, and Set 6 contained 1 000 power traces collected from a new device D with fixed key K6.

B. Adapting Ablation Methodology for Portability Setting

We adapt our previously proposed methodology in Section III-B to handle portability issues. In particular, we tune the methodology for MDMSD setting as shown in Algorithm 2. The proposed algorithm's objective is to ablate and re-train the model from a given training on the original device such that it does not overfit to the training device itself but can generalize to a range of devices.

Dataset Reference	Device	Fix/Random Key Type	Key	Notation
[BCH$^+$20]	B1	Fix	K1	B1_K1
	B2	Fix	K2	B2_K2
	B3	Fix	K1	B3_K1
	B4	Fix	K3	B4_K3
[oCHS18]	Device A	Random	-	A_RN
	Device B	Random	-	B_RN
	Device C	Random	-	C_RN
	Device C	Fix	K4	C_K4
	Device C	Fix	K5	C_K5
	Device D	Fix	K6	D_K6

TABLE II: The target datasets for MDMSD.

The adversary collects the traces for training and testing based on the original device o, denoted as $Train_o$ and $Test_o$ respectively. The original model, ML_o, is first trained on this device with epoch τ_o on $Train_o$. The GE for pre-trained model (GE_o) is then computed as based on model ML_o and $Test_o$ dataset with additional noise α. The adversary then ablates ML_o with a rate ρ and re-trains with the same number of epochs τ_o to obtain the new ablated model ML_r^ρ. The GE for the ablated model (GE_r^ρ) is then computed from model ML_r^ρ and dataset $(Test_o + Noise(\beta))$. Next, the adversary defines the threshold margin m. While the condition $GE_r^\rho > (m.GE_o)$ holds, the adversary repeats the while loop in Algorithm 2 starting from ML_o. Once the condition is no longer fulfilled, the ablated model is ready to handle portability. The adversary can then test the data from the victim device, $Test_v$, and calculate GE_v^ρ and thus the secret key.

Algorithm 2 Methodology for MDMSD.

procedure MDMSD(The original device o with train, test dataset $Train_o, Test_o$ and training epoch τ_o, Victim device v with test dataset $Test_v$ and training epoch τ_r, threshold margin m, Noise value for train and test α, β, Ablate rate ρ)
  $ML_o \leftarrow$ Pre-train Model with $Train_o$, epoch τ_o
  $(GE_r^\rho, GE_o) \leftarrow (\infty, 0)$
  while $GE_r^\rho > (m.GE_o)$ **do**
    $ML_r \leftarrow ML_o$
    $GE_o \leftarrow$ Attack(ML_o)
      ▷ Compute GE on $(Test_o + Noise(\beta))$)
    $ML_r^\rho \leftarrow Ablate(ML_r)$
    $ML_r^\rho \leftarrow$ Train(ML_r^ρ)
      ▷ Train with $(Train_o + Noise(\alpha))$, epoch τ_r
    $GE_r^\rho \leftarrow$ Attack(ML_r^ρ)
      ▷ Compute GE on $(Test_o + Noise(\beta))$)
  end while
  $GE_v^\rho \leftarrow$ Attack(ML_r^ρ) ▷ Compute GE on $Test_v$
  Return GE_v^ρ
end procedure

This MDMSD approach is aligned with the one proposed in Algorithm 1. We tested three different ablation rate (10%, 50% and 99%) for [BCH$^+$20] dataset. 99% ablation gave the best result, about 6× better than other ablation rates (see Figure 26 in Appendix). By 99% ablation, we mean ablating the whole layer except for a single neuron (to maintain the connectivity between layers). We hypothesize that portability causes overfitting, which affects the whole layer, and thus

ablating the full layer (99%) is better. Consequently, we use this configuration in the following experiments.

Parameters α and β must be chosen carefully to better represent noise from portability. If α and β take a similar value and are too small, the resulting GE_o and GE_r^ρ will be relatively the same and would not address the portability issue. By setting a larger α, both ML_o and ML_r^ρ will not learn much information, so the attack on the test dataset will also fail, even with ablation and re-training. Therefore, we use relatively small α to make sure the model will work, then use ablation to fight with large β that represents the portability-induced noise. Finally, if the condition $GE_r^\rho \leq (m.GE_o)$ is satisfied, we stop the Algorithm 2 and obtain the final GE from $Test_v$ dataset of victim device. If m is 1, the re-trained model ML_r^ρ is better than ML_o because GE_r^ρ is smaller than GE_o. However, ablation can lead to cases where GE_r^ρ could be slightly higher than GE_o. To counter such scenarios, we provide a 5% leverage to GE_r^ρ, thus $m = 1.05$.

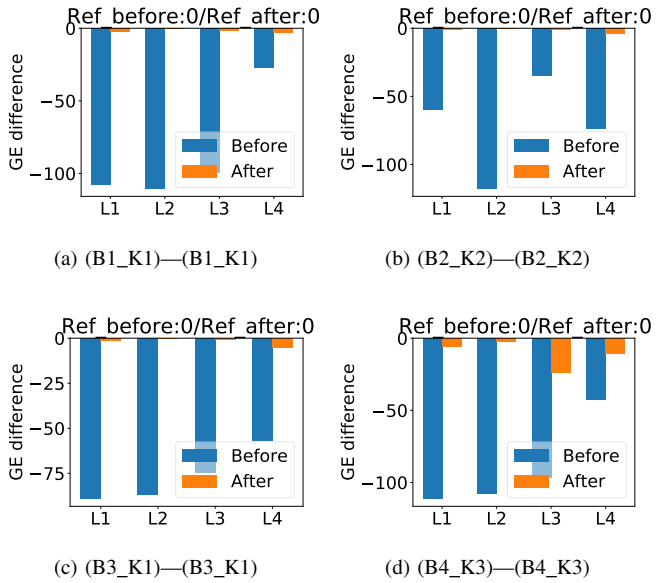
C. Evaluation Results

We use the MLP2 architecture proposed in [BCH$^+$20] for the following experiments. This architecture is selected as the best performing one since it has sufficient capacity to model the data and yet does not overfit as easily as the investigated CNNs. More precisely, MLP2 architecture has four hidden layers where each layer has 500 neurons, the batch size is 256, the number of epochs is 50, the loss function is categorical cross-entropy, and the optimizer is $RMSprop$ with a learning rate of 0.001.

1) Results for [BCH$^+$20] Dataset: We train MLP2 (ML_o) for the dataset (Line 2 of Algorithm 2), with the (train)—(test) datasets as follows: (B1_K1)—(B1_K1), (B2_K2)—(B2_K2), (B3_K1)—(B3_K1), (B4_K3)—(B4_K3).

To apply the MDMSD, we use the following parameter settings: m: 1.05, α: 5×10^{-4}, β: $20 \times \alpha$, τ_o: 50, τ_r: 50. The re-trained ML_r^ρ performs slightly worse than the original model ML_o if the ablated layer handles overfitting from the original dataset $Train_o$. We use the 50 epoch for training (and re-training) as in [BCH$^+$20]. The results of Line 6 and 9 in Algorithm 2 for each dataset are shown in Figure 7.

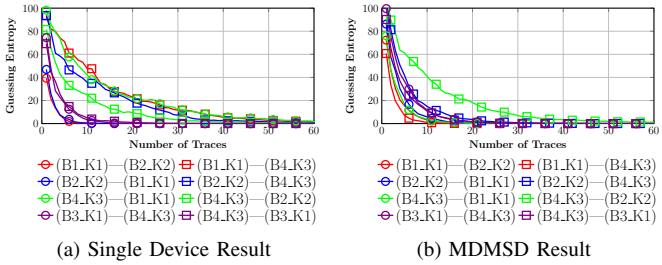
In Figure 7, the ablated second layer (L2) seems to achieve better performance since GE_r is less than $1.05 \times GE_o$ for all experiments. Therefore, we utilize the re-trained architecture (ML_r^ρ) by ablating the second layer (L2). To directly compare with previous results [BCH$^+$20], we plot the progression of GE for re-trained architecture in Figure 8b. Figure 8a benchmarks the original result from [BCH$^+$20]. We see that MDMSD result outperforms the original result. Except for (B4_K3)—(B1_K1), it mostly only requires 10-20 traces to recover the correct key. To better represent the results, we also compute the averaged GE for the eight results reported in Figures 8a and 8b [BCH$^+$20]. The averaged results are shown in Figure 11a. MDMSD requires half the traces (about 30) as compared to original results (about 60 traces) to break the target.



(a) (B1_K1)—(B1_K1) (b) (B2_K2)—(B2_K2)

(c) (B3_K1)—(B3_K1) (d) (B4_K3)—(B4_K3)

Fig. 7: **Guessing entropy difference before and after the recovery training for [BCH$^+$20] dataset**



(a) Single Device Result (b) MDMSD Result

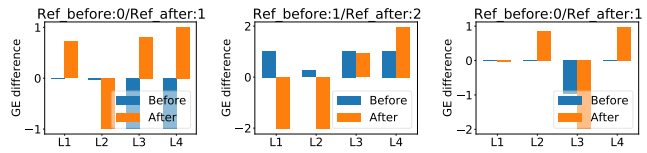
Fig. 8: Result for [BCH$^+$20] Dataset.

Note that MDMSD is proposed to bridge the gap between MDM and a single device threat model. If multiple devices are available, MDM should always be preferred.

2) Results for [oCHS18] Dataset: Next, for [oCHS18] dataset we focus on the KeySchedule leakage rather than S-box operation as reported in [DFK19]. Specifically, we aim to recover the Hamming weight of 0^{th} byte round key in the KeySchedule operation. As the leakage is the Hamming weight of a byte, the range for GE is between 0 and 8.

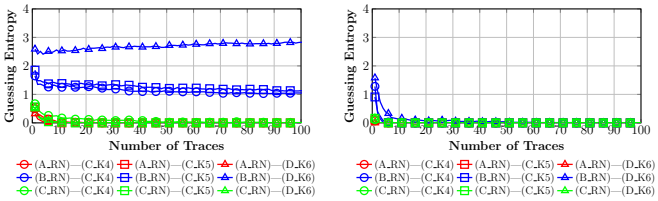
We perform the cross-device attack on [oCHS18] dataset to compare our approach. As seen in Figure 10a, we cannot recover the Hamming weight information when we train B_RN dataset. To apply the MDMSD, we use the parameter setting: m: 1.05, α: 5×10^{-4}, β: $20 \times \alpha$, τ_o: 50, τ_r: 50.

In Figure 9, ablating L4 satisfies $GE_r \leq 1.05 \times GE_o$. Unlike to original result, all Hamming weight information is recovered using less than 50 traces (see Figure 10b). Especially, except for (B_RN)—(D_K6), only ten traces are needed to recover the Hamming weight of the round key (about 5 on average considering all experiments), while the



(a) (A_RN)—(C_K4) (b) B_RN—(C_K4) (c) (C_RN)—(C_K4)

Fig. 9: **Guessing entropy difference before and after the recovery training for [oCHS18] dataset**



(a) Single Device Result (b) MDMSD Result

Fig. 10: Result for [oCHS18] dataset.

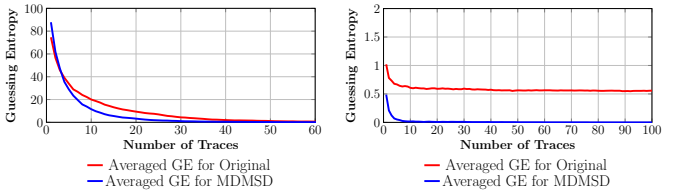
single device results never converge. As the dataset is fairly simple to attack, changes observed for all layers in Figure 9 are very small, and ablating other layers had a similar effect (we considered L4 as it gave the best results).

VII. Discussion & Summary

In this paper, we present the ablation methodology for deep learning-based SCA. We concentrate on the behavior of various types of noise (Gaussian noise, desynchronization, clock jitter) and investigate a plethora of experimental settings (neural networks, datasets, leakage models). Our results indicate how various types of noise affect different neural networks, allowing us to better understand the inner working of neural networks. When the deep learning-based SCA breaks the target, our methodology allows 1) to understand in what layers the noise is handled, 2) gives intuition how difficult the countermeasure is, and 3) allows us to understand whether smaller neural networks can be used while reaching the same performance. When deep learning-based SCA cannot break the target, our methodology allows us to understand in what layers is the largest influence of noise, thus indicating the architectural parts that should be redesigned.

We enumerate the most important findings that we consistently observed in the experiments:

1) Gaussian noise is an easier countermeasure for neural networks than desynchronization. This can be seen because portability for Gaussian noise makes no issues, while for desynchronization, we require re-training to adapt to the changes in the dataset.
2) Clock jitter is an even more difficult countermeasure, and we require comparably larger neural networks for successful attacks. Note that while we mostly do not manage to break datasets with the clock jitter countermeasure, it does not mean it is impossible to break it, but only that it requires more measurements as demonstrated by Cagli et al. [CDP17].
3) Gaussian noise is handled in the first layers of MLP architectures (indicating rather shallow architectures are sufficient), while desynchronization and clock jitter countermeasures require more layers and deeper architectures. Gaussian noise is only minimally processed in the CNNs' convolution layer, while for desynchronization, the convolution layer plays the biggest role. This confirms the results of Zaid et al. [ZBHV19] where the AES_HD architecture is rather shallow with two dense layers, and ASCAD (Desync=100) has a deep architecture with ten dense layers.
4) A neural network aims to adapt its weights in the layer where ablation happens and in the next layer (less influenced layers means that the adapting is easier).
5) Ablation can indicate what part of the neural network needs to be adjusted for an attack to work. We show this with the desynchronization countermeasure and ASCAD with random keys.
6) Ablation indicates where countermeasures are processed and whether a neural network can be made smaller. However, in its current form, it does not show what parts exactly can be removed without a performance penalty.
7) We require at least some model learnability for ablation to provide meaningful results. If the trained model performs on the level of random guessing, it is hard to explain such a neural network's inner working. Still, the ablation study can serve as a strong indication of such behavior, especially considering that recently, Wu et al. showed how guessing entropy could be a misleading metric [WWK+20]. If GE shows poor performance (e.g., on the level of random guessing) and ablating a neural network does not show any differences in weights, it is clear that the model did not learn anything.
8) Ablation is a useful tool in understanding how neural networks work and how SCA countermeasures are processed. Still, that does not mean every ablation experiment will be equally easy to explain.
9) Ablation can help bridge the gap between the single device model and MDM when multiple devices are not available. Ablating layers responsible for the network's overfitting to a single device can help the model to



(a) Averaged GE Result for [BCH+20] dataset (b) Averaged GE Result for [oCHS18] dataset

Fig. 11: Results for the two datasets considering portability.

generalize better.

We mainly considered ablations happening in a single layer only. While we are confident that such an approach gives the most explainable results, we would like to investigate in the future what happens when we ablate more layers simultaneously. This is especially interesting for CNNs, where we can ablate convolution and fully-connected layers. Finally, as the current deep learning-based SCA trend uses relatively small neural networks, we consider our work perfectly aligned with the current state-of-the-art. Still, it would be interesting to investigate ablation on larger neural network architectures, as we postulate such architectures will become increasingly important with the improvements in the countermeasures.

REFERENCES

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[AB18] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[ALAM19] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE symposium on security and privacy (sp)*, pages 1362–1380. IEEE, 2019.

[BCH+20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[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. Cryptographic Engineering*, 10(2):163–188, 2020.

[C+15] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 45–68, Cham, 2017. Springer International Publishing.

[CLB+18] Chaofan Chen, Oscar Li, Alina Barnett, Jonathan Su, and Cynthia Rudin. This looks like that: deep learning for interpretable image recognition. *CoRR*, abs/1806.10574, 2018.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[DFK19] Tobias Damm, Sven Freud, and Dominik Klein. Dissecting the ches 2018 aes challenge. *IACR Cryptol. ePrint Arch.*, 2019:783, 2019.

[DGD+19] Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. X-deepsca: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*, page 134. ACM, 2019.

[GA19] David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI Magazine*, 40(2):44–58, 2019.

[GHO15] Richard Gilmore, Neil Hanley, and Maire O'Neill. Neural network based attack on a masked implementation of AES. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 106–111, May 2015.

[GJS19] Aron Gohr, Sven Jacob, and Werner Schindler. CHES 2018 side channel contest CTF - solution of the AES challenges. *IACR Cryptol. ePrint Arch.*, 2019:94, 2019.

[HGG20] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 645–666, Cham, 2020. Springer International Publishing.

[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.

[HPTD15] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *LNCS*, pages 249–264. Springer, 2012.

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

[KKK16] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2280–2288. Curran Associates, Inc., 2016.

[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 Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[KRKK18] Kwonyoup Kim, Sangryeol Ryu, Taehyun Kim, and Taewon Kim. AES Wireless Keyboard – Template Attack for Eavesdropping. BlackHat Asia, Singapore, 2018.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A Machine Learning Approach Against a Masked AES. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2013. Berlin, Germany.

[Lom16] Victor Lomne. Common criteria certification of a smartcard: a technical overview. *CHES 2016*, 2016.

[MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 145–167. Springer, 2019.

[MLdPM19] Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. Ablation studies in artificial neural networks. *CoRR*, abs/1901.08644, 2019.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag, Berlin, Heidelberg, 2007.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[oCHS18] Conference on Cryptographic Hardware and Embedded Systems. Ches 2018 ctf, 2018. https://chesctf.riscure.com/2018/news.

[PCP20] Guilherme Perin, Lukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):337–364, Aug. 2020.

[PHG17] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. Template attack versus bayes classifier. *J. Cryptogr. Eng.*, 7(4):343–351, 2017.

[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 Anupam Chattopadhyay, Chester Rebeiro, and Yuval Yarom, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 157–176, Cham, 2018. Springer International Publishing.

[RBA20] Unai Rioja, Lejla Batina, and Igor Armendariz. When similarities among devices are taken for granted: Another look at portability. In Abderrahmane Nitaj and Amr M. Youssef, editors, *Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings*, volume 12174 of *Lecture Notes in Computer Science*, pages 337–357. Springer, 2020.

[Sch77] Peter H Schiller. The effect of superior colliculus ablation on saccades elicited by cortical stimulation. *Brain research*, 1977.

[SMY09] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 443–461, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[SWM17] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[vdVPB19] Daan van der Valk, Stjepan Picek, and Shivam Bhasin. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. Cryptology ePrint Archive, Report 2019/1477, 2019. https://eprint.iacr.org/2019/1477.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):147–168, Jun. 2020.

[Wei20] Léo Weissbart. Performance analysis of multilayer perceptron in profiling side-channel analysis. In *International Conference on Applied Cryptography and Network Security*, pages 198–216. Springer, 2020.

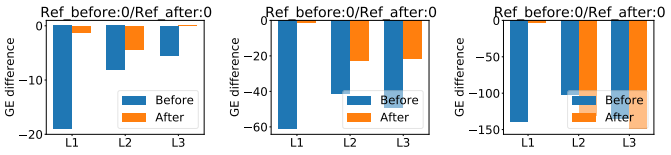
[WP20] Lichao Wu and Stjepan Picek. Remove some noise: On preprocessing of side-channel measurements with autoencoders. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):389–415, Aug. 2020.

[WPP20] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2020/1293, 2020. https://eprint.iacr.org/2020/1293.

[WWK+20] Lichao Wu, Léo Weissbart, Marina Krček, Huimin Li, Guilherme Perin, Lejla Batina, and Stjepan Picek. On the attack evaluation and the generalization ability in profiling side-channel analysis. Cryptology ePrint Archive, Report 2020/899, 2020. https://eprint.iacr.org/2020/899.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.

[ZSX+20] Fan Zhang, Bin Shao, Guorui Xu, Bolin Yang, Ziqi Yang, Zhan Qin, and Kui Ren. From homogeneous to heterogeneous: Leveraging deep learning based power analysis across devices. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pages 1–6. IEEE, 2020.



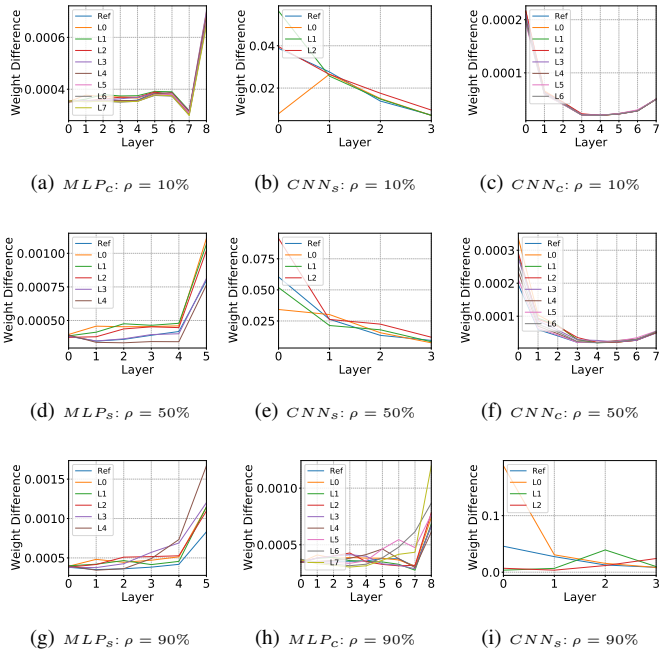
(a) CNN_s: $\rho = 10\%$ (b) CNN_s: $\rho = 50\%$ (c) CNN_s: $\rho = 90\%$

Fig. 12: **Guessing entropy difference before and after the recovery training for CNN_s: Gaussian noise** $(\gamma, \sigma, \rho, \tau) = (1.0, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key.

APPENDIX

A. Results for the ASCAD with the Fixed Key Dataset

The additional results for ASCAD with the fixed key are presented in this section. First, we depict the results for the HW leakage model and CNN_s in Figure 12. Figure 13 gives the weight differences for the HW leakage model. In Figure 15, we give the weight difference results for the desynchronization countermeasure and the HW leakage model. Finally, Figures 16 and 17 present the results for the HW leakage model and clock jitter.



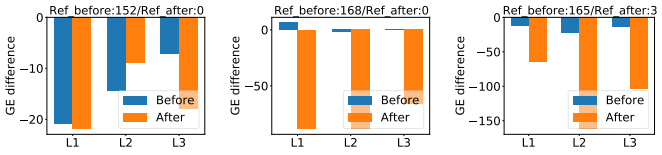
(a) MLP_c: $\rho = 10\%$ (b) CNN_s: $\rho = 10\%$ (c) CNN_c: $\rho = 10\%$

(d) MLP_s: $\rho = 50\%$ (e) CNN_s: $\rho = 50\%$ (f) CNN_c: $\rho = 50\%$

(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_s: $\rho = 90\%$

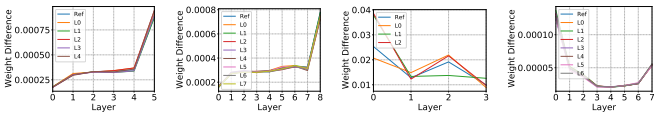
Fig. 13: **Weight difference before and after the recovery training: Gaussian noise** $(\gamma, \sigma, \rho, \tau) = (1.0, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key.

Next, Figures 18 and 19 give the results for the Gaussian noise with $\gamma = 1$ for the ID leakage model. Finally, Figures 20 and 21 give results for desynchronization.

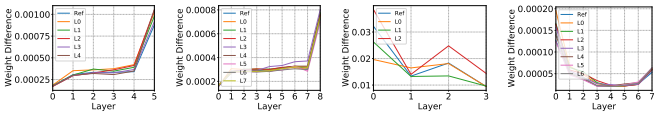


(a) CNN_s: $\rho = 10\%$ (b) CNN_s: $\rho = 50\%$ (c) CNN_s: $\rho = 90\%$

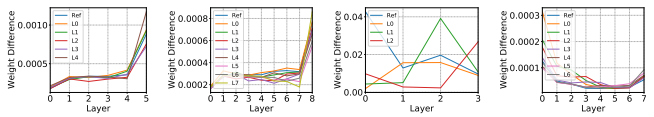
Fig. 14: **Guessing entropy difference before and after the recovery training for CNN_s: Desynchronization** $(\gamma, \sigma, \rho, \tau) = (5, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key.



(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_s: $\rho = 10\%$ (d) CNN_c: $\rho = 10\%$

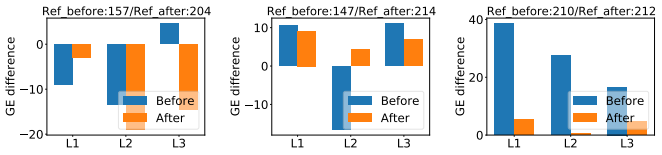


(e) MLP_s: $\rho = 50\%$ (f) MLP_c: $\rho = 50\%$ (g) CNN_s: $\rho = 50\%$ (h) CNN_c: $\rho = 50\%$



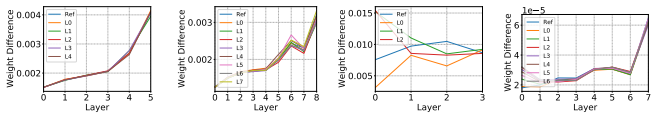
(i) MLP_s: $\rho = 90\%$ (j) MLP_c: $\rho = 90\%$ (k) CNN_s: $\rho = 90\%$ (l) CNN_c: $\rho = 90\%$

Fig. 15: **Weight difference before and after the recovery training: Desynchronization** $(\gamma, \sigma, \rho, \tau) = (5, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key.

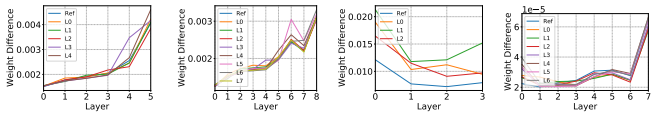


(a) CNN_s: $\rho = 10\%$ (b) CNN_s: $\rho = 50\%$ (c) CNN_s: $\rho = 90\%$

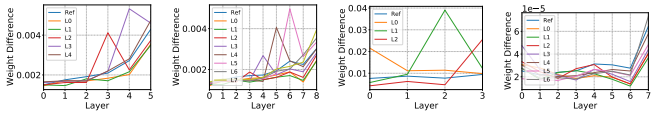
Fig. 16: **Guessing entropy difference before and after the recovery training for CNN_s: Clock Jitter** $(\gamma, \sigma, \rho, \tau) = (1, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key.



(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_s: $\rho = 10\%$ (d) CNN_c: $\rho = 10\%$

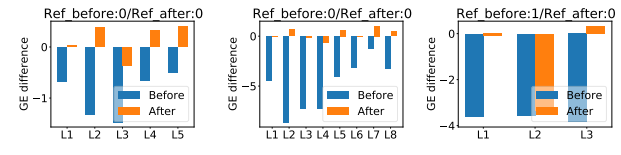


(e) MLP_s: $\rho = 50\%$ (f) MLP_c: $\rho = 50\%$ (g) CNN_s: $\rho = 50\%$ (h) CNN_c: $\rho = 50\%$



(i) MLP_s: $\rho = 90\%$ (j) MLP_c: $\rho = 90\%$ (k) CNN_s: $\rho = 90\%$ (l) CNN_c: $\rho = 90\%$

Fig. 17: **Weight difference before and after the recovery training: Clock Jitter** $(\gamma, \sigma, \rho, \tau) = (1, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with the fixed key.



(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_s: $\rho = 10\%$

(d) MLP_s: $\rho = 50\%$ (e) MLP_c: $\rho = 50\%$ (f) CNN_s: $\rho = 50\%$

(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_s: $\rho = 90\%$

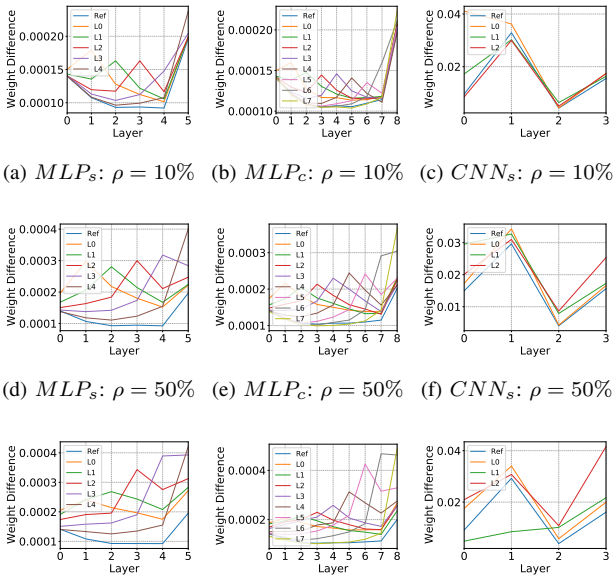
Fig. 18: **Guessing entropy difference before and after the recovery training: Gaussian noise** $(\gamma, \sigma, \rho, \tau) = (1.0, 10, (10\%, 50\%, 90\%), 10)$ for the ID leakage model on ASCAD with the fixed key.

B. Results for the ASCAD with Random Keys Dataset

For the ASCAD with random keys, we provide results for CNN_s and HW leakage model in Figure 22, while the weight differences are in Figure 23. Finally, Figure 24 gives the results for CNN_s for the desynchronization countermeasure in the HW leakage model and Figure 25 gives the weight difference results.

C. Methodology for the Multiple Device Model from a Single Device

The results for MDMSD with different ablation rates are given in Figure 26. Observe how ablation rates of 10% and

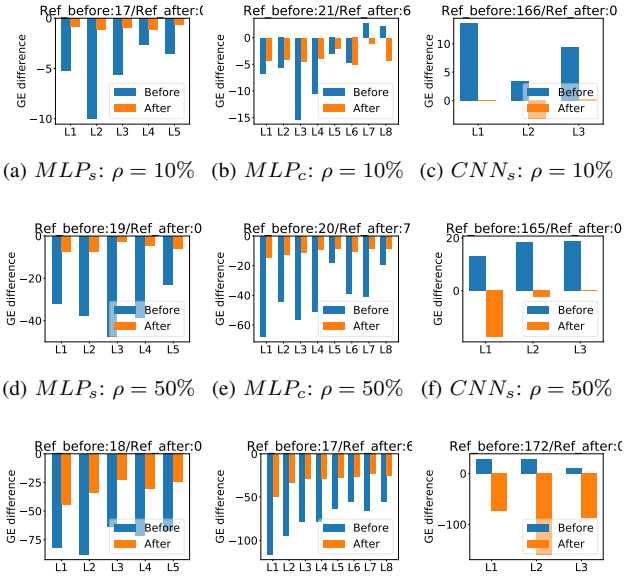


(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_s: $\rho = 10\%$

(d) MLP_s: $\rho = 50\%$ (e) MLP_c: $\rho = 50\%$ (f) CNN_s: $\rho = 50\%$

(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_s: $\rho = 90\%$

Fig. 19: **Weight difference before and after the recovery training: Gaussian noise** $(\gamma, \sigma, \rho, \tau) = (1.0, 10, (10\%, 50\%, 90\%), 10)$ for the ID leakage model on ASCAD with the fixed key.

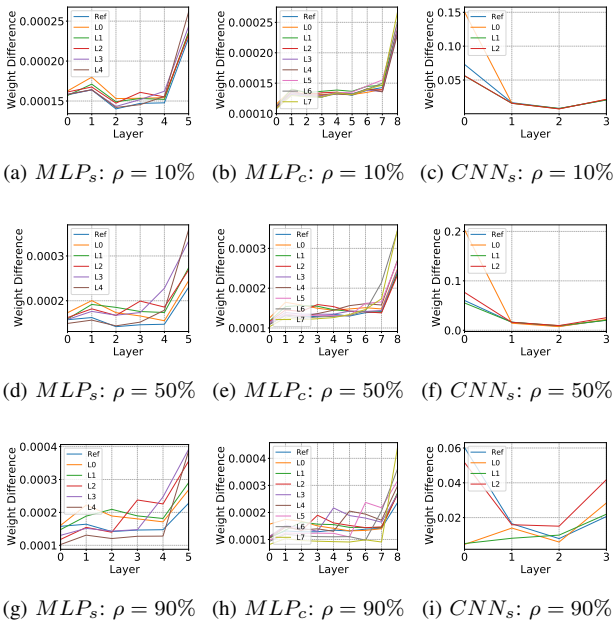


(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_s: $\rho = 10\%$

(d) MLP_s: $\rho = 50\%$ (e) MLP_c: $\rho = 50\%$ (f) CNN_s: $\rho = 50\%$

(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_s: $\rho = 90\%$

Fig. 20: **Guessing entropy difference before and after the recovery training: Desynchronization** $(\gamma, \sigma, \rho, \tau) = (5, 10, (10\%, 50\%, 90\%), 10)$ for the ID leakage model on ASCAD with the fixed key.

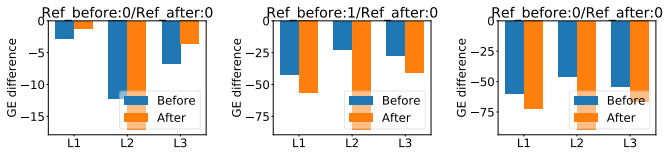


(a) MLP_s: $\rho = 10\%$ (b) MLP_c: $\rho = 10\%$ (c) CNN_s: $\rho = 10\%$

(d) MLP_s: $\rho = 50\%$ (e) MLP_c: $\rho = 50\%$ (f) CNN_s: $\rho = 50\%$

(g) MLP_s: $\rho = 90\%$ (h) MLP_c: $\rho = 90\%$ (i) CNN_s: $\rho = 90\%$

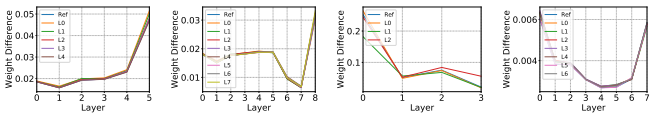
Fig. 21: **Weight difference before and after the recovery training: Desynchronization** $(\gamma, \sigma, \rho, \tau) = (5, 10, (10\%, 50\%, 90\%), 10)$ for the ID leakage model on ASCAD with the fixed key.



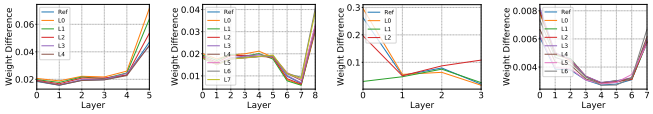
(a) CNN_s: $\rho = 10\%$ (b) CNN_s: $\rho = 50\%$ (c) CNN_s: $\rho = 50\%$

Fig. 22: **Guessing entropy difference before and after the recovery training for CNN_s: Gaussian noise** $(\gamma, \sigma, \rho, \tau) = (1, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with random keys.

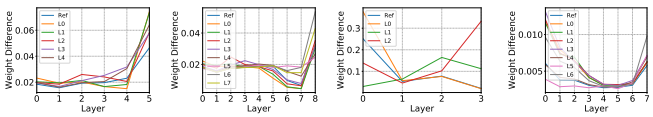
50% give similar results, while 99% ablation rate performs significantly better.



(a) MLP_s: ρ = 10% (b) MLP_c: ρ = 10% (c) CNN_s: ρ = 10% (d) CNN_c: ρ = 10%

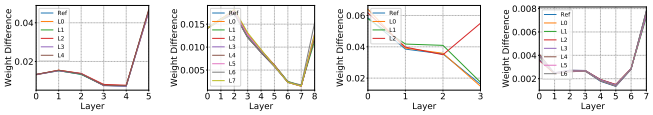


(e) MLP_s: ρ = 50% (f) MLP_c: ρ = 50% (g) CNN_s: ρ = 50% (h) CNN_c: ρ = 50%

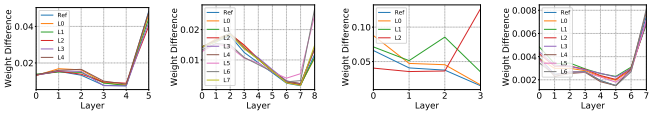


(i) MLP_s: ρ = 50% (j) MLP_c: ρ = 50% (k) CNN_s: ρ = 50% (l) CNN_c: ρ = 50%

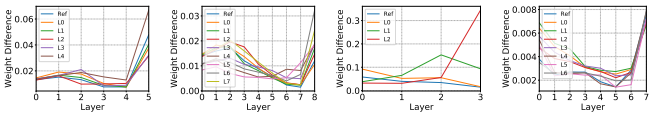
Fig. 23: **Weight difference before and after the recovery training: Gaussian noise** $(\gamma, \sigma, \rho, \tau) = (1, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with random keys.



(a) MLP_s: ρ = 10% (b) MLP_c: ρ = 10% (c) CNN_s: ρ = 10% (d) CNN_c: ρ = 10%

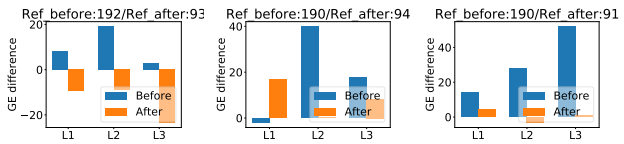


(e) MLP_s: ρ = 50% (f) MLP_c: ρ = 50% (g) CNN_s: ρ = 50% (h) CNN_c: ρ = 50%



(i) MLP_s: ρ = 90% (j) MLP_c: ρ = 90% (k) CNN_s: ρ = 90% (l) CNN_c: ρ = 90%

Fig. 25: **Weight difference before and after the recovery training: Desynchronization** $(\gamma, \sigma, \rho, \tau) = (5, 10, (10\%, 50\%), 10)$ for the HW leakage model on ASCAD with random keys.



(a) CNN_s: $\rho = 10\%$ (b) CNN_s: $\rho = 50\%$ (c) CNN_s: $\rho = 90\%$

Fig. 24: **Guessing entropy difference before and after the recovery training for CNN_s: Desynchronization** $(\gamma, \sigma, \rho, \tau) = (5, 10, (10\%, 50\%, 90\%), 10)$ for the HW leakage model on ASCAD with random keys.

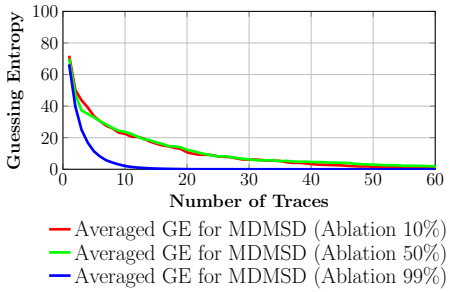


Fig. 26: Results of averaged GE for (B1_K1) —(B2_K2) and (B1_K1)—(B4_K3).