A Systematic Study of Data Augmentation for Protected AES Implementations

Huimin Li[1] and Guilherme Perin[2]

[1] Delft University of Technology, The Netherlands
[2] Leiden University, The Netherlands
h.li-7@tudelft.nl, g.perin@liacs.leidenuniv.nl

Abstract. Side-channel attacks against cryptographic implementations are mitigated by the application of masking and hiding countermeasures. Hiding countermeasures attempt to reduce the Signal-to-Noise Ratio of measurements by adding noise or desynchronization effects during the execution of the cryptographic operations. To bypass these protections, attackers adopt signal processing techniques such as pattern alignment, filtering, averaging, or resampling. Convolutional neural networks have shown the ability to reduce the effect of countermeasures without the need for trace preprocessing, especially alignment, due to their shift invariant property. Data augmentation techniques are also considered to improve the regularization capacity of the network, which improves generalization and, consequently, reduces the attack complexity.

In this work, we deploy systematic experiments to investigate the benefits of data augmentation techniques against masked AES implementations when they are also protected with hiding countermeasures. Our results show that, for each countermeasure and dataset, a specific neural network architecture requires a particular data augmentation configuration to achieve significantly improved attack performance. Our results clearly show that data augmentation should be a standard process when targeting datasets with hiding countermeasures in deep learning-based side-channel attacks.

Keywords: Side-channel Attacks, Deep Learning, Data Augmentation, Hiding Countermeasures

1 Introduction

Side-channel attacks (SCAs) represent a realistic threat to electronic systems processing confidential information. SCA is a non-invasive attack that targets assets such as keys from cryptographic modules in software or hardware implementations. These cryptographic implementations are present in chips applied to the Internet-of-Things, payment, automotive, and content protection industries, just to name a few. SCA is conducted by monitoring physical side-channel information that is unintentionally leaked by electronic circuits, such as power consumption, electromagnetic emissions, and execution time. The leaked information might be statistically related to the confidential data being processed by the circuit, such as cryptographic keys.

SCA is divided into two main categories: direct attacks such as differential power analysis [10] or correlation power analysis [2] that exploit the statistical relation between side-channel measurements and secret information, and two-step or profiled attacks [4]. In those attacks, a profiling model is learned from side-channel information collected from an open target, and this model is later used to retrieve secret information from a victim's device. This way, profiled attacks follow a supervised learning strategy, and for this reason, recently, deep neural networks have been widely considered for profiled attacks [17] due to their practical advantages in comparison to previous techniques such as Gaussian template attacks [4].

To mitigate SCA, manufacturers implement countermeasures that aim at breaking the statistical relation between side-channel information and secret keys. Two main types of countermeasures are typically applied: masking and hiding. Masking countermeasures add random values (i.e., masks) to sensitive bytes during cryptographic executions. The main goal of hiding countermeasures is to reduce the Signal-to-Noise Ratio (SNR) of side-channel measurements by intentionally adding noise to the circuit. The most common hiding countermeasures methods are noise generators, e.g., parallel circuits that produce significant power consumption to hide the power consumption of sensitive operations, and desynchronization, e.g., random delays that shift the target operation in time. Desynchronization efficiently protects cryptographic implementations because side-channel attack methods such as DPA or template attacks require side-channel measurements aligned in the time domain.

When dealing with hiding countermeasures, a standard side-channel analysis procedure is to apply signal processing to remove noise with filtering, averaging, or resampling. To bypass desynchronization, techniques such as static or dynamic alignment [24] are common solutions. Although post-signal processing tends to improve side-channel analysis results, the process faces several limitations, especially the large time overheads in side-channel evaluations, the requirement for costly and specialized equipment, and, in some cases, the inability to successfully conduct signal processing over raw measurements due to stronger hiding countermeasures. Convolutional neural networks (CNN) have shown promising results in bypassing desynchronization protections [3,28]. Convolution blocks, typically composed of a combination of convolution and pooling layers, provide a shift-invariant property that makes CNN less sensitive to side-channel trace misalignment, especially when used as a profiling model. One way to further improve the robustness of a CNN against trace misalignment is by training the model with data augmentation. Data augmentation is an explicit regularization technique that increases training data size by generating additional synthetic data during training. Essentially, in side-channel analysis, what a data augmentation process does is reproduce the effect of existing hiding countermeasures from measured side-channel traces. This way, the augmented training set tends to represent a better sample of the true (and unknown) leakage distribution of side-channel traces. This process improves CNN generalization as the model has fewer chances to overfit the training data.

Although data augmentation is a well-known method to cope with hiding countermeasures in side-channel measurements [3,19], it is not clearly answered how to implement data augmentation for specific targets or datasets properly and what is the best augmentation configuration. For instance, to reduce the protective effect of desynchronization, one tries to create a data augmentation process that randomly shifts the training set at each training epoch. Still, knowing the ideal amount of trace samples to shift for a certain trace set has been unanswered so far. Moreover, the required number of augmented data that provides the best results was never investigated. In this work, we provide results showing that each specific neural network architecture requires a particular data augmentation configuration, which makes the problem even more complicated. The same also applies to hiding countermeasures based on additive (Gaussian) noise.

In this paper, we focus on profiling SCA and verify to what extent data augmentation suppresses the protective effects of hiding countermeasures. We skip signal processing and rely solely on the regularization and generalization ability of convolutional neural networks to deal with noisy datasets. We perform a systematic data augmentation analysis by deploying an analysis methodology that identifies the best data augmentation strategy for a given dataset containing specific hiding countermeasures. Our results demonstrate that each neural network architecture and dataset require a specific data augmentation strategy. Interestingly, with the correct data augmentation configuration, we can turn an inefficient CNN that does not recover the key (with a given number of attack traces) into a successful CNN model that recovers the correct key with state-of-the-art results. Moreover, the performance of CNN models with the best data augmentation configuration found with our analysis methodology is the best reported in the literature so far with higher levels of trace desynchronization. For the `ASCAD` dataset, we can successfully recover the key with less than 50 attack traces when the desynchronization level is up to 200 sample points. For the `DPAv4.2` dataset, our best CNN model with the best data augmentation configuration recovers the key with a single attack trace when the desynchronization level is up to 150 sample points. Our analysis indicates that data augmentation should be a standard process when evaluating cryptographic implementations with hiding countermeasures in the context of deep learning-based profiling SCA.

2 Background

2.1 Deep Learning-based Profiling Side-channel Analysis

We define a side-channel trace set as $\mathcal{X}$ with size N, where $\mathtt{x}_i$ is the i-th observation of $\mathcal{X}$. With an additional under-script term, $\mathtt{x}_{i,s}$, we refer to a feature (or sample) s in a side-channel observation $\mathtt{x}_i$. For each side-channel observation

$\mathtt{x}_i$, we assign a label $\mathtt{y}_i \in \mathcal{Y}$. Each side-channel observation $\mathtt{x}_i$ represents side-channel leakages obtained from a target while running a cryptographic operation such as encryption. The label $\mathtt{y}_i$ is derived from a selection function that returns the intermediate variable (in our case, a byte value) associated with the executed cryptographic operation. For instance, when the cryptographic operation is an AES encryption function $\mathcal{C} = \mathbf{E}(\mathcal{D}, \mathcal{K})$ with a secret key $\mathcal{K}$ and plaintext $\mathcal{D}$, which returns a ciphertext $\mathcal{C}$, the selection function can be represented as the output byte of the S-Box in the first encryption round, i.e., $\mathtt{y}_i = \texttt{S-Box}(\mathtt{d}_j \oplus \mathtt{k}_j)$, where $\mathtt{k}_j \in \mathcal{K}$ is the j-th key byte and $\mathtt{d}_j$ represents the j-th byte from plaintext $\mathcal{D}$.

In a deep learning-based profiling SCA application, the main goal is to train a deep neural network $f(\mathcal{L}, \theta, \mathcal{T})$, defined by a set of parameters θ, with a training set $\mathcal{T} = (\mathcal{X}_{train}, \mathcal{Y}_{train})$, to minimize the loss function $\mathcal{L}$. The trained neural network, or simply model, is validated with a separate validation set of size V, $\mathcal{V} = (\mathcal{X}_{val}, \mathcal{Y}_{val})$ by measuring the validation loss value.

Although minimizing validation loss is an efficient validation metric, the best way to verify the performance of a model in the SCA context is by computing SCA metrics such as key rank with the given validation set. By predicting a trained model $f(\mathcal{L}, \theta, \mathcal{T})$ with $\mathcal{V}$, we obtain a set of class probabilities P, where $\mathtt{p}_{i,\mathtt{y}} \in P$ indicates the probability of observing label $\mathtt{y}$ for a given side-channel observation $\mathtt{x}_i \in \mathcal{V}$. Because labels $\mathtt{y}$ depend on the key byte $\mathtt{k}_i$ from the validation set $\mathcal{V}$, the key rank is a process that returns the most likely key byte candidate or hypothesis $\mathtt{k}_h$ among all possible key values, which includes the 256 possible byte values. This way, we compute the likelihood g_h for each key candidate $\mathtt{k}_h$ as follows:

$$g_h = \sum_{i=0}^{V-1} \mathtt{log}\ \mathtt{p}_{i,\mathtt{y}}. \tag{1}$$

By repeating this process for $h = 1 \ldots 256$, we obtain a vector of sorted key likelihoods $\mathtt{g} = \{g_h\}$, $h \in [1, 256]$, by order of magnitude of g_h values. The position of the correct key candidate inside sorted $\mathtt{g}$ gives the key rank for the validation set. The guessing entropy [22,18] of the correct key is given by an empirical process in which we repeat the key rank process multiple times (each time with a different and randomly selected subset from $\mathcal{V}$), and we obtain an average key likelihood or key guessing vector $\mathtt{g}$ and get the average position of the correct key $\mathtt{k}^*$ inside $\mathtt{g}$.

In this paper, we refer to the guessing entropy of the correct key byte candidate as $\mathtt{ge}^*$. Another metric to verify the performance of a trained model $f(\mathcal{L}, \theta, \mathcal{T})$ against a validation set $\mathcal{V}$ is by obtaining the minimum size of V (i.e., the minimum number of validation traces) that are necessary to achieve $\mathtt{ge}^* = 1$ (which means that the correct key byte candidate has the lowest guessing entropy among all key byte candidates), which we refer as $N_{\mathtt{ge}^*=1}$.

2.2 Data Augmentation

In the deep learning community, data augmentation is considered in state-of-the-art applications, such as image classification [7,23,12]. It refers to the process of increasing the size of the training set by artificially generating additional training data with dynamic changes during the training of a model. These changes must preserve the class properties of the training set. The training set represents an approximate distribution, given by a finite set $\mathcal{T}$, from a true and unknown distribution $\mathcal{R}$. By augmenting the training set $\mathcal{T}$, one expects that the $\mathcal{T}$ becomes a better representation of $\mathcal{R}$. A deep neural network becomes less prone to overfit the training data by following a data augmentation process. Among other regularization techniques such as weight decay, dropout, batch normalization, and transfer learning [21,23], data augmentation is an alternative and efficient way to reduce overfitting.

To achieve this goal, data augmentation settings need to be carefully chosen. However, conventionally data augmentation involves many manual or random choices. The main idea is to improve class representation inside of a dataset. For that, it is important to understand what kind of effect the augmentation process needs to develop. For instance, when training a convolution neural network to be as shift-invariant as much as possible concerning images, adding rotation, shifts, resizing, or re-scaling improves the number of examples with image variations. On the other hand, inappropriate choices of data augmentation settings probably lead to no effect or even detrimental effect [7,5]. To skip the manual augmentation process, different techniques have been proposed in deep learning literature. In [5], the authors proposed a procedure called AutoAugment to automatically search for the best data augmentation setting from training data properties. Later, the authors proposed a new strategy called Randaugment [6]. Randaugment greatly reduces the computational expense of automated augmentation by simplifying the search space. Ultimately, these automated data augmentation processes require optimization algorithms such as reinforcement learning.

2.3 Datasets

In our experiments, we consider two publicly available software masked AES datasets.

`ASCADr` ASCAD database [1] provides side-channel measurements collected from different software AES implementations: AES protected with first-order Boolean masking running on an 8-bit Atmega device [3], and AES protected with Boolean, affine, and shuffling running on a 32-bit STM32 platform [4]. The former is considered in our experiments, and it contains two main trace sets: (1) trace set with 60 000 traces, where each power measurement contains 100 000 samples points, and all traces contains the same fixed key, and (2) trace set with 300 000 traces,

[3] `https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1`

[4] `https://github.com/ANSSI-FR/ASCAD/tree/master/STM32_AES_v2`

each measurement containing 250 000 sample points, with first 200 000 containing random keys and the remainder 100 000 containing a fixed key. We consider this last dataset with 300 000 measurements, hereby called ASCADr. In our experiments, we take the trimmed version of ASCADr, which contains 1 400 sample points per trace and represents the power consumption of the third key byte j ($j \in [0, 15]$) of the S-Box output in the first encryption round. Therefore, each trace $\mathtt{x}_i$ is labeled according to $\mathtt{y}_i = \mathtt{S\text{-}Box}(\mathtt{d}_2 \oplus \mathtt{k}_2)$ when we consider the Identity leakage model or $\mathtt{y}_i = HW(\mathtt{S\text{-}Box}(\mathtt{d}_2 \oplus \mathtt{k}_2))$ when we apply the Hamming weight leakage model. We use 200 000 traces for training, 5 000 for validation, and another 5 000 as the attack set.

DPAv4.2 The DPAcontest v4.2 dataset (DPAv4.2) [5] is the second implementation available in the DPAcontest v4. It is an improved version implemented in software on an 8-bit Atmel ATMega-163 smart card and corrects several leaks identified in its previous generation. This dataset represents the power consumption of the first AES encryption round, and the AES implementation is protected with RSM (Rotate S-box Masking). The dataset contains a total of 80 000 traces, and each of them contains 1 704 402 sample points. In our experiments, we trim the dataset to the interval representing the processing of the twelfth S-box byte j ($j \in [0, 15]$), resulting in 2 000 samples per trace. We use 70 000 traces for training (which contains 14 different keys), 5 000 for validation, and another 5 000 as the attack set. Each trace $\mathtt{x}_i$ is labeled according to $\mathtt{y}_i = \mathtt{S\text{-}Box}(\mathtt{d}_{11} \oplus \mathtt{k}_{11})$ when we consider the Identity leakage model or $\mathtt{y}_i = HW(\mathtt{S\text{-}Box}(\mathtt{d}_{11} \oplus \mathtt{k}_{11}))$ when we apply the Hamming weight leakage model.

3 Related Works

Data augmentation has been widely applied to the SCA context. In [3], data augmentation was considered to mitigate trace desynchronization effects caused by jitter effects. The results showed significant improvements in profiling attacks when compared to Gaussian template attacks. In that case, the authors applied two customized data augmentation techniques based on shift deformation and add-remove deformation of side-channel measurements. In [9], the authors considered a regularization technique that artificially adds Gaussian noise to the training set. Results showed significant key recovery improvements in the attack phase. Although this process only modifies the existing training set without augmenting the training set during model training, we still consider this work as data augmentation due to the modifications applied to input traces. In [16], the authors applied SMOTE, a data augmentation technique to suppress imbalanced dataset limitations. The authors of [11] applied *mixup* [27] technique for data augmentation. Mukhtar et al. [13] considered Generative Adversarial Networks (GANs) and Siamese networks to generate new side-channel traces for data augmentation. While this approach works well, due to its black-box character, it

[5] https://www.dpacontest.org/v4/42_doc.php

becomes more difficult to evaluate the effect of a specific change. In [14], the authors demonstrated that data augmentation based on random shifts could act as a strong regularizer for label correction in an iterative framework.

In the context of SCA, data augmentation essentially solves three main problems: (1) it suppresses the lack of training data for better class representations (also to suppress class imbalance) [13], (2) it augments the training set to cover the effects of existing hiding countermeasures better (e.g., cover a wider range of trace shift positions due to misalignment or jitter) [3], and (3) it regularizes the model to prevent overfitting [15]. For SCA, data augmentation also creates an adversarial training effect [8] on the model [20]. Indeed, hiding countermeasures that are expected to be presented in side-channel measurements collected from the target device contain modifications (e.g., desynchronization, additive noise) that aim at perturbing the prediction of the trained model. Training with data augmentation leads to a model that is more robust to unseen modifications that can exist in measurements from different targets.

However, it is still an open question of how to customize a data augmentation for a specific dataset. More specifically, for each considered hiding countermeasure, data augmentation requires defining optimal configuration hyperparameters. In Section 4, we provide an analysis methodology to evaluate this open question, and in Section 5, we provide experimental results for different datasets.

4 Analysis Methodology

In this section, we describe the analysis methodology applied in our experiments. The proposed methodology defines a grid search to identify what is the best data augmentation setting for specific countermeasures present in side-channel measurements.

The analysis starts by taking clean side-channel traces, i.e., raw side-channel measurements, where we assume hiding countermeasures such as noise and desynchronization are not active to protect the underlying device under test. Obviously, as we are dealing with real side-channel measurements, some level of noise is still present. However, the Signal-to-Noise Ratio (SNR) is sufficiently high to assume that side-channel measurements contain irrelevant noise. Hiding countermeasures are artificially emulated by adding either Gaussian noise or desynchronization to the raw measurements. This is done by choosing different hiding countermeasures hyperparameters such as standard deviation for the added Gaussian distribution and the maximum number of shifted samples in side-channel traces in case of desynchronization.

Next, we perform the hyperparameter search to find the best possible CNN models that can recover the target key in a profiling attack scenario, even in the presence of added hiding countermeasures. Table 1 shows the hyperparameter options from where each CNN model is randomly configured during the search. In case when the best-found neural network is not capable of successfully retrieving the key (i.e., $N_{\mathsf{ge}^*=1} > V$), the best model will be the one that presents lower guessing entropy ge^*. This analysis will serve as a baseline comparison for the experiments with data augmentation. Note that the early stopping process is not considered during the hyperparameter search process. To eventually implement early stopping, we would have to set an early stopping metric such as guessing entropy, which would add significant overheads to the search process. Therefore, every model is trained for a total of 100 epochs, as this number of epochs is in accordance with related works [1,25,26] and, for a majority of cases, enough to find a CNN model with $N_{\texttt{ge}^*=1} \leq V$.

After the random search process, we search for the best data augmentation configuration. We start from the best-found CNN models obtained with the hyperparameter search, and we train these models from scratch with data augmentation by considering a grid of different hyperparameters. For that, we consider the data augmentation that implements the same effects provided by the given hiding countermeasure. This way, data augmentation involves applying Gaussian noise to training data or desynchronization. For the Gaussian noise case, we test different standard deviations to see if there is an optimal value that provides better performance. In the same scope, we test different desynchronization levels, i.e., the maximum number of randomly selected shifted samples in side-channel traces during model training. The idea is again to identify if, for a given desynchronization provided by hiding countermeasures, there is an optimal range of sample shifts for data augmentation. We also evaluate if there is a minimum number of augmented traces that provide better results. For that, we train the best-found CNN models with different numbers of augmented traces added to the original training set.

To summarize, our methodology implements four main steps:

1. Add hiding countermeasures to the raw side-channel measurements.
2. Deploy random hyperparameter search to identify the best CNN model for each hiding countermeasure scenario (the hyperparameter range is in Table 1).
3. Investigate the best data augmentation hyperparameters (e.g., standard deviation or maximum trace shifts) through a grid search.
4. Investigate the minimum number of augmented side-channel traces during neural network training that improves CNN performance.

4.1 Adding Hiding Countermeasures

We emulate hiding countermeasures on raw side-channel traces. We explore two cases: desynchronization and Gaussian noise. Desynchronization emulates the effect of hiding countermeasures aimed at providing trace misalignment. The Gaussian noise emulates the effect of additive noise provided by the target to reduce the SNR of measurements. For that, we define the following hyperparameters:

- δ_{hid}: maximum number of trace sample shifts. The shifts that are applied to each measurement are drawn from a normal distribution with a mean equal to $\delta_{hid}/2$. The blue lines in Figures 1a and 1b refer to the distribution of shifts when $\delta_{hid} = 25$ and $\delta_{hid} = 200$, respectively.

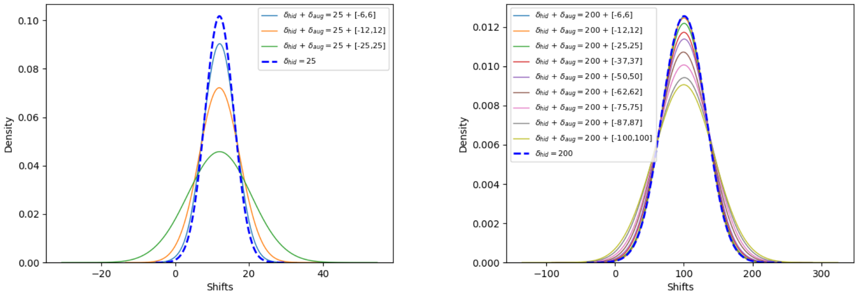
- σ_{hid}: standard deviation considered to define a Gaussian distribution from where we obtain a noise trace that is added to raw measurements. The mean of the distribution is zero. Figures 2a and 2b show the SNR analysis for the ASCADr and DPAv42 datasets without adding Gaussian noise. We can see that the max value for the ASCADr dataset is 1.52 when SNR is computed for the intermediate $v = \texttt{S-Box}(d_2 \oplus k_2) \oplus m_2$. The max value for the DPAv42 dataset is 4.14 when the intermediate is $v = \texttt{S-Box}(d_{12} \oplus k_{12}) \oplus m_{12}$. When the Gaussian noise countermeasure is added to these two datasets, and the standard deviation σ_{hid} changes from 1 to 6, the max values reduce accordingly in Table 2. Note that SNR or intermediate variables are significantly reduced.

Table 1: Hyperparameter variations in the CNN architecture.

Hyperparameters	**Options**
neurons	{20, 40, 50, 100, 150, 200, 300, 400}
batch_size	{100, 200, 400}
layers	{1, 2}
filters	{4, 8, 12, 16}
kernel_size	{10, 20, 30, 40}
strides	{5, 10, 15, 20}
pool_type	{"Average", "Max"}
pool_size	2
conv_layers	{1, 2, 3, 4}
activation	{"elu", "selu", "relu"}
learning_rate	{0.005, 0.0025, 0.001, 0.0005, 0.00025, 0.0001, 0.00005, 0.000025, 0.00001}
weight_init	{"random_uniform", "he_uniform", "glorot_uniform", "random_normal", "he_normal", "glorot_normal"}
optimizer	{"Adam", "RMSprop"}

4.2 Data Augmentation Hyperparameters

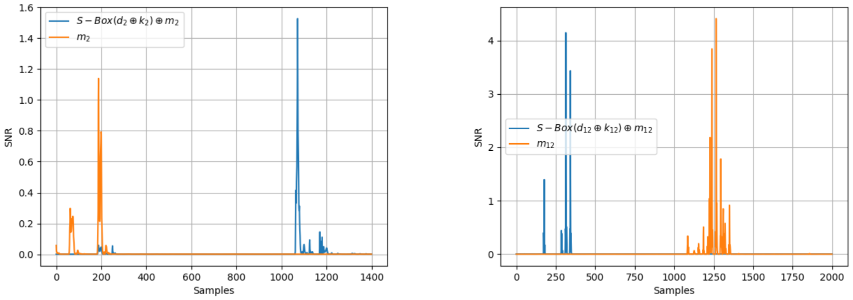
For our analysis, the data augmentation strategy requires the definition of the following hyperparameters:



(a) Trace desynchronization distribution for different values of δ_{aug} when measurements contain desynchronization of $\delta_{hid} = 25$.

(b) Trace desynchronization distribution for different values of δ_{aug} when measurements contain desynchronization of $\delta_{hid} = 200$.

Fig. 1: Trace desynchronization distribution for different values of augmentation shifts $[-\delta_{aug}, \delta_{aug}]$.



(a) SNR analysis for the `ASCADr` dataset without countermeasure for masked `S-Box` output and corresponding mask.

(b) SNR analysis for the `DPAv42` dataset without countermeasure for masked `S-Box` output and corresponding mask.

Fig. 2: SNR analysis without countermeasure

- **Augmented hyperparameter**: this hyperparameter refers to the data augmentation type that is applied to training data. If the data augmentation type is Gaussian noise, the statistical hyperparameter to be tuned is the standard deviation, σ_{aug}, of the applied normal distribution with zero mean. In case the data augmentation type is desynchronization, the statistical hyperparameter is the range of shifts, $[-\delta_{aug}, \delta_{aug}]$, applied to the training data. We randomly shift each trace to the left and to the right by randomly taking the shift value from a normal distribution with mean zero and minimum value being $-\delta_{aug}$ and maximum value being δ_{aug}. Note that random shifts

Table 2: The max values change in SNR analysis for two datasets with Gaussian noise countermeasure. The mean of the distribution is zero. The standard deviation σ_{hid} changing from 1 to 6.

$\sigma_{hid} =$	0	1	2	3	4	5	6
ASCADr							
$v = \texttt{S-Box}(d_2 \oplus k_2) \oplus m_2$	1.52	1.21	0.79	0.51	0.38	0.30	0.25
$v = m_2$	1.13	1.07	0.94	0.78	0.64	0.50	0.41
DPAv4.2							
$v = \texttt{S-Box}(d_{12} \oplus k_{12} \oplus m_{12})$	4.14	3.74	2.89	2.13	1.55	1.15	0.87
$v = m_{12}$	4.40	3.92	2.97	2.21	1.66	1.26	0.98

during data augmentation are always selected from a normal distribution, and the mean of the distribution is zero. Figures 1a and 1b illustrate the final desynchronization distributions after we apply the data augmentation shifts to trace sets containing $\delta_{hid} = 25$ and $\delta_{hid} = 200$, respectively. Note how the final distribution, given by $\delta_{hid} + [-\delta_{aug}, \delta_{aug}]$, provides a larger range of possible trace shifts during the training phase. This larger range is more salient when $\delta_{hid} = 25$ than $\delta_{hid} = 200$.

- **Augmented traces per epoch**: this hyperparameter refers to the number of augmented training side-channel measurements that are generated for each epoch. In this case, augmented traces are different as they are randomly generated for each epoch. Note that the resulting training set consists of original traces plus augmented ones.

5 Experimental Results

In this section, we provide experimental results by applying our analysis methodology to the two datasets described in Section 2.3.

5.1 Desynchronization Countermeasure

ASCADr Tables 3 and 4 provide results for the ASCADr dataset labeled with the Identity leakage model and Hamming weight leakage model, respectively. As specified by the table's header, the training is always conducted for the 200 000 traces plus the augmented traces. When the augmented traces are denoted by 0 (third column of the table), we indicate the $N_{\texttt{ge}^*=1}$ value (i.e., the number of attack traces required to reach $\texttt{ge}^* = 1$) for the baseline model trained *without* data augmentation.

Note that for each different δ_{hid} value, the CNN architecture is different, and it is obtained from a random search. Then we deploy a new training for this CNN model by considering data augmentation with a different number of augmented traces (from 20 000 to 200 000 augmented traces - from 10% of the number of the original traces up to 100%). For each number of these augmented traces, the model is trained with a different range of shifts $[-\delta_{aug}, \delta_{aug}]$. We always set this value to ensure that $2 \times \delta_{aug} \leq \delta_{hid}$.

Results shown in Table 3 demonstrate the efficiency of data augmentation for different CNN architectures with the ASCADr dataset and the Identity leakage model. The $N_{\texttt{ge}^*=1}$ value obtained for the baseline model (third table column) is always higher than the lowest value obtained with the best $N_{\texttt{ge}^*=1}$ when data augmentation is active during training. Specifically, the case when $\delta_{hid} = 25$ is very representative. When this CNN model is trained without data augmentation, we obtain $N_{\texttt{ge}^*=1} > 3\,000$, indicating that this model cannot successfully recover the key with less than 3 000 traces. When data augmentation with 120 000 augmented traces is applied during training (these traces are randomly generated for each epoch), with $\delta_{aug} = 25$, the correct key candidate is recovered with only 39 traces. Moreover, when $\delta_{hid} = 175$, which indicates a more aggressive desynchronization level, the baseline model without data augmentation still successfully recovers the correct key with 2 195 traces. However, after applying data augmentation with 180 000 augmented traces at each training epoch and $[-\delta_{aug}, \delta_{aug}] = [-87, 87]$, the correct key is recovered with only 76 traces, which is a significant improvement. Finally, when δ_{hid} equals 200, at the highest level of trace desynchronization in our experiments, we get $N_{\texttt{ge}^*=1} = 304$ for the baseline model and $N_{\texttt{ge}^*=1} = 44$ for augmentation with $[-\delta_{aug}, \delta_{aug}] = [-12, 12]$ and 180 000 augmented traces.

The results in Table 4 show the performance of different CNN models with different data augmentation configurations when the ASCADr dataset is labeled with the Hamming weight leakage model. We also first choose the best CNN model through a random search under different δ_{hid} without augmentation. Then, for each δ_{hid}, we use the same CNN model to conduct the training process with 200 000 raw traces plus a different number of augmented traces. We can see the improvement from data augmentation since the $N_{\texttt{ge}^*=1}$ value obtained for the baseline model is higher than the lowest value obtained with the best $N_{\texttt{ge}^*=1}$ for different δ_{aug} in most cases. Take $\delta_{hid} = 100$ for example. We get $N_{\texttt{ge}^*=1} = 1\,898$ when the CNN model is trained without augmentation. When augmentation is implemented, the correct key candidate is recovered with fewer traces in each δ_{aug} when the augmented trace number is greater than 40 000. For $\delta_{hid} = 125$, we often use fewer traces for every δ_{aug} than the baseline model when the augmented trace number is greater than 80 000. Moreover, we can see that augmentation works with even higher desynchronization levels. When $\delta_{hid} = 200$, the baseline model recovers the correct key candidate with 2 677 traces. By adding 120 000 traces at each training epoch and $[-\delta_{aug}, \delta_{aug}] = [-50, 50]$, we can recover the correct key with only 533 traces.

DPAv4.2 Tables 5 and 6 demonstrate results for the DPAv4.2 dataset with desynchronization countermeasure adopted with the Identity leakage model and the

Table 3: Number of attack traces to reach guessing entropy equal to 1. Results obtained with the ASCADr dataset and the Identity leakage model. Neural networks are trained with data augmentation by *generating different augmented traces at each epoch.*

		200k original traces +										
δ_{hid}	$[-\delta_{aug}, \delta_{aug}]$	**0**	**20k**	**40k**	**60k**	**80k**	**100k**	**120k**	**140k**	**160k**	**180k**	**200k**
25	[-6, 6]	> 3000	-	-	444	94	623	84	97	1477	73	68
	[-12, 12]		-	-	181	268	49	90	80	82	49	57
	[-25, 25]		-	-	1952	62	-	39	59	77	59	54
50	[-6, 6]	114	-	-	-	96	377	77	44	64	51	55
	[-12, 12]		-	105	-	152	107	58	64	55	53	33
	[-25, 25]		-	-	-	119	-	74	41	62	113	92
	[-37, 37]		-	-	-	-	-	215	92	-	81	69
75	[-6, 6]	317	-	549	208	89	111	65	72	116	92	71
	[-12, 12]		-	271	157	120	76	66	86	103	103	90
	[-25, 25]		-	815	229	139	86	115	101	58	69	59
	[-37, 37]		-	302	286	179	116	121	51	71	71	100
	[-50, 50]		-	980	230	224	140	84	64	56	92	100
100	[-6, 6]	774	-	-	-	-	1090	1172	859	358	481	251
	[-12, 12]		-	-	-	-	350	362	317	553	370	305
	[-25, 25]		-	-	-	-	825	370	479	275	351	378
	[-37, 37]		-	-	-	876	624	598	667	368	353	498
	[-50, 50]		-	-	-	1741	642	423	346	509	484	472
	[-62, 62]		-	-	-	761	634	333	496	645	254	688
125	[-6, 6]	252	228	148	124	149	119	160	104	121	143	118
	[-12, 12]		183	143	129	80	50	88	104	120	100	58
	[-25, 25]		182	84	75	105	91	65	89	84	97	112
	[-37, 37]		175	122	79	122	55	139	122	109	64	87
	[-50, 50]		447	158	97	95	63	91	73	82	66	98
	[-62, 62]		327	182	88	123	64	80	107	92	70	48
	[-75, 75]		162	249	82	97	83	75	106	99	75	78
150	[-6, 6]	1615	-	1002	738	378	903	339	230	335	462	378
	[-12, 12]		-	719	462	551	264	442	322	489	276	246
	[-25, 25]		-	1202	508	318	243	272	306	302	252	255
	[-37, 37]		-	1374	526	339	280	223	335	204	311	156
	[-50, 50]		-	-	254	337	405	254	427	281	187	286
	[-62, 62]		-	653	322	277	248	354	197	276	248	286
	[-75, 75]		-	2117	861	302	273	229	355	184	162	238
	[-87, 87]		-	1008	-	345	279	195	298	396	218	173
175	[-6, 6]	2195	2168	412	491	244	286	248	293	602	300	228
	[-12, 12]		1117	923	1136	324	234	183	284	319	157	198
	[-25, 25]		1310	970	694	220	226	184	159	150	266	137
	[-37, 37]		2068	894	978	333	199	173	127	146	131	162
	[-50, 50]		2986	1167	1289	291	335	197	117	120	108	132
	[-62, 62]		2787	1785	1365	161	254	202	260	183	237	107
	[-75, 75]		2638	1802	662	462	279	248	114	129	103	86
	[-87, 87]		1393	2995	698	673	213	217	175	128	76	104
	[-100, 100]		2546	-	847	453	191	202	153	80	219	126
200	[-6, 6]	304	-	-	-	-	282	176	222	87	69	107
	[-12, 12]		-	-	-	-	220	246	106	54	44	74
	[-25, 25]		-	-	-	211	143	153	239	698	57	75
	[-37, 37]		-	-	-	-	365	201	255	97	107	50
	[-50, 50]		-	-	-	-	265	129	111	120	85	88
	[-62, 62]		-	-	-	-	-	165	203	106	69	68
	[-75, 75]		-	-	-	-	-	-	179	68	94	73
	[-87, 87]		-	-	-	-	-	296	180	52	109	79
	[-100, 100]		-	-	-	-	-	176	146	371	59	-

Table 4: Number of attack traces to reach guessing entropy equal to 1. Results obtained with the ASCADr dataset and the Hamming weight leakage model. Neural networks are trained with data augmentation by *generating different augmented traces at each epoch*.

		200k original traces +										
δ_{hid}	$[-\delta_{aug}, \delta_{aug}]$	**0**	**20k**	**40k**	**60k**	**80k**	**100k**	**120k**	**140k**	**160k**	**180k**	**200k**
25	[-6, 6]	1053	2288	-	1495	-	936	490	970	722	634	666
	[-12, 12]		2166	-	-	598	1317	1185	609	792	-	-
	[-25, 25]		-	-	-	-	-	-	2470	-	-	-
50	[-6, 6]	> 3000	-	2621	2454	2016	2020	2097	1260	1775	1344	-
	[-12, 12]		-	-	2576	1897	2771	-	2809	-	-	-
	[-25, 25]		-	-	-	-	-	-	-	-	-	-
	[-37, 37]		-	-	-	-	-	-	-	-	-	-
75	[-6, 6]	1238	2599	2384	1562	1495	1608	1490	1627	1688	1913	2670
	[-12, 12]		1928	2512	1482	1740	2029	1887	2145	1406	2331	1804
	[-25, 25]		2972	2402	1750	566	2232	1426	1275	1957	1442	1419
	[-37, 37]		2599	1997	1577	1471	1255	802	1921	1019	989	1414
	[-50, 50]		2375	-	1414	1278	1192	1489	1215	747	1358	1194
100	[-6, 6]	1898	2818	2372	988	789	758	590	762	504	629	500
	[-12, 12]		-	676	1017	810	666	749	604	680	654	583
	[-25, 25]		1614	686	1090	537	775	759	499	607	658	522
	[-37, 37]		-	1775	754	1170	499	681	780	545	303	427
	[-50, 50]		-	1534	691	938	737	889	683	727	464	797
	[-62, 62]		1192	2596	1134	1036	810	1145	836	757	930	692
125	[-6, 6]	2509	-	-	1175	1564	1438	1018	660	1475	1518	1211
	[-12, 12]		2007	2007	1764	1706	836	1885	1404	1675	705	721
	[-25, 25]		2259	1067	1613	2555	936	993	1201	1016	742	820
	[-37, 37]		1389	-	1417	866	1225	788	603	823	841	869
	[-50, 50]		-	1504	2565	1460	1096	1593	1264	660	787	842
	[-62, 62]		2139	920	1557	903	-	648	1787	971	904	810
	[-75, 75]		-	2239	1359	-	1299	2495	1063	-	1207	1569
150	[-6, 6]	851	2036	2138	1000	945	590	889	925	908	625	564
	[-12, 12]		1254	1782	683	690	707	661	717	691	474	571
	[-25, 25]		821	1427	922	645	929	568	540	595	584	729
	[-37, 37]		1485	1077	895	458	571	512	357	693	641	515
	[-50, 50]		1081	1132	617	739	791	406	586	716	322	465
	[-62, 62]		1875	1140	727	850	445	613	521	948	561	689
	[-75, 75]		1041	1526	486	718	1187	565	678	531	1197	596
	[-87, 87]		982	1163	794	962	844	702	914	709	2771	748
175	[-6, 6]	592	679	723	460	496	568	564	484	465	521	319
	[-12, 12]		639	639	492	527	592	429	446	508	400	459
	[-25, 25]		621	598	552	376	515	429	745	405	419	420
	[-37, 37]		686	422	486	633	516	416	443	407	465	554
	[-50, 50]		677	426	546	660	395	586	384	450	408	466
	[-62, 62]		705	574	651	843	494	495	689	-	433	549
	[-75, 75]		674	800	471	665	736	580	-	397	554	404
	[-87, 87]		1142	362	534	323	586	635	487	391	597	318
	[-100, 100]		1025	837	645	774	434	537	597	470	599	412
200	[-6, 6]	2677	-	2067	1009	1413	1030	737	712	804	687	569
	[-12, 12]		-	-	2068	829	994	1139	892	719	771	669
	[-25, 25]		1056	-	2151	810	1537	730	818	677	1052	541
	[-37, 37]		-	2157	2053	1347	1098	1185	1327	703	846	1058
	[-50, 50]		1007	-	1058	943	-	533	658	860	913	1038
	[-62, 62]		-	1469	1203	2098	-	709	899	876	830	887
	[-75, 75]		1617	1480	-	890	984	1099	576	952	839	829
	[-87, 87]		-	924	2263	1508	1461	2018	865	999	636	1006
	[-100, 100]		-	1436	-	-	1088	1246	1629	844	1010	709

Hamming weight leakage model, respectively. The training is always conducted for 70 000 traces plus the augmented traces. The augmented traces denoted by 0 indicate the number of attack traces required to reach $\texttt{ge}^* = 1$ for the baseline model trained *without* data augmentation. For each different δ_{hid} value, the CNN architecture is obtained from a random search with 70 000 traces. Later, new training is adopted for this CNN model with data augmentation for different δ_{aug} with 70 000 original traces plus 7 000 to 70 000 augmented traces (from 10% of the number of original traces to 100%). The desynchronization levels δ_{aug} is also set to $2 \times \delta_{aug} \leq \delta_{hid}$.

Table 5 gives results for the `DPAv4.2` dataset with the Identity leakage model. For $\delta_{hid} = \{25, 50, 75, 125, 150\}$, we observe that the $N_{\texttt{ge}^*=1}$ value of the baseline model is often higher than the lowest value obtained with the best $N_{\texttt{ge}^*=1}$ when data augmentation is active during training. However, there are also some cases where we get $N_{\texttt{ge}^*=1} > 3\,000$ when the CNN model is trained with data augmentation. The case when $\delta_{hid} = 150$ shows how data augmentation improves a CNN that, without data augmentation, requires 141 attack traces to reach $\texttt{ge}^* = 1$. After augmentation is applied, it requires a single attack trace when at least 35 000 augmented traces are considered. When $\delta_{hid} = \{175, 200\}$, we cannot get the correct key using the chosen model under 3 000 traces without augmentation. We also cannot recover the correct key using augmentation techniques. This means that when desynchronization is at a high level for this dataset and leakage model, it is not easy to recover the correct key successfully, whether or not augmentation is adopted.

The results in Table 6 illustrate the performance of different CNN models with different data augmentation configurations for the `DPAv4.2` dataset labeled with the Hamming weight leakage model. We also observe that the $N_{\texttt{ge}^*=1}$ value obtained for the baseline model is always higher than the lowest value obtained with the best $N_{\texttt{ge}^*=1}$ when data augmentation is adopted during training. This is even true for $\delta_{hid} = 200$, the highest level of Gaussian noise. The baseline model without data augmentation gets the correct key successfully with 1 371 traces. However, after applying data augmentation with $[-\delta_{aug}, \delta_{aug}] = [-100, 100]$ and using 63 000 augmented traces at each training epoch, the correct key is recovered with only 21 traces.

5.2 Gaussian Noise Countermeasure

`ASCADr` Tables 7 and 8 provide results for the `ASCADr` dataset for Gaussian noise countermeasure with the Identity leakage model and the Hamming weight leakage model, respectively. The term σ_{hid} refers to the standard deviation in Gaussian noise (with zero mean) applied to the raw traces for a hiding countermeasure. The term σ_{aug} denotes the standard deviation in Gaussian noise applied to the augmented traces. The training is always conducted for the 200 000 traces plus the augmented traces. The augmented traces denoted by 0 indicate the number of attack traces required to reach $\texttt{ge}^* = 1$ for the baseline model trained *without* data augmentation. Again, for each different σ_{hid} value, the CNN architecture is different, and it is obtained from the best one from a random search.

Table 5: Number of attack traces to reach guessing entropy equal to 1. Results obtained with the DPAv4.2 dataset and the Identity leakage model. Neural networks are trained with data augmentation by *generating different augmented traces at each epoch.*

		70k original traces +										
δ_{hid}	$[-\delta_{aug}, \delta_{aug}]$	**0**	**7k**	**14k**	**21k**	**28k**	**35k**	**42k**	**49k**	**56k**	**63k**	**70k**
25	[-6, 6]	2712	-	276	478	362	335	147	315	261	170	220
	[-12, 12]		-	522	1233	1025	-	-	-	-	-	-
	[-25, 25]		-	1383	1312	-	-	-	-	-	-	-
50	[-6, 6]	991	-	2174	252	-	548	-	679	-	-	-
	[-12, 12]		-	-	138	-	-	-	201	-	188	440
	[-25, 25]		-	-	-	98	-	70	107	372	116	241
	[-37, 37]		-	-	-	-	-	55	225	131	56	199
75	[-6, 6]	108	141	47	31	32	9	6	8	4	4	4
	[-12, 12]		112	29	6	3	3	7	3	11	23	4
	[-25, 25]		96	44	21	4	4	3	2	1	5	5
	[-37, 37]		159	45	23	7	3	2	1	8	2	28
	[-50, 50]		114	36	5	5	4	9	2	25	4	2
100	[-6, 6]	317	640	300	141	124	97	92	145	45	56	94
	[-12, 12]		607	204	118	68	59	59	20	22	27	28
	[-25, 25]		680	86	61	25	36	6	47	17	12	51
	[-37, 37]		353	178	75	16	18	5	11	15	33	23
	[-50, 50]		657	192	58	98	15	17	13	34	45	9
	[-62, 62]		387	276	82	126	22	16	5	6	24	71
125	[-6, 6]	223	1598	437	820	263	328	400	241	153	213	141
	[-12, 12]		964	568	160	386	285	215	236	118	205	153
	[-25, 25]		1585	1043	564	638	438	271	262	255	429	733
	[-37, 37]		1212	2253	838	1132	782	454	178	287	214	492
	[-50, 50]		1650	399	1076	515	292	236	303	209	912	613
	[-62, 62]		1479	664	341	657	373	528	294	1004	655	559
	[-75, 75]		1033	492	574	1750	463	745	275	886	341	1336
150	[-6, 6]	141	32	101	-	27	-	3	13	-	-	1
	[-12, 12]		-	-	107	-	3	19	2	8	3	2
	[-25, 25]		-	-	-	-	21	2	1	1	1	1
	[-37, 37]		41	-	-	19	-	-	1	2	2	1
	[-50, 50]		1330	-	-	2	-	2	3	1	2	1
	[-62, 62]		-	-	-	156	12	-	1	1	1	1
	[-75, 75]		-	-	23	-	3	41	1	1	1	1
	[-87, 87]		-	-	-	-	1	14	-	59	15	1
175	[-6, 6]	> 3000	-	-	-	-	-	-	-	-	-	-
	[-12, 12]		-	-	-	-	-	-	-	-	-	-
	[-25, 25]		-	-	-	-	-	-	-	-	-	-
	[-37, 37]		-	-	-	-	-	-	-	-	-	-
	[-50, 50]		-	-	-	-	-	-	-	-	-	-
	[-62, 62]		-	-	-	-	-	-	-	-	-	-
	[-75, 75]		-	-	-	-	-	-	-	-	-	-
	[-87, 87]		-	-	-	-	-	-	-	-	-	-
	[-100, 100]		-	-	-	-	-	-	-	-	-	-
200	[-6, 6]	> 3000	-	-	-	-	-	-	-	-	-	-
	[-12, 12]		-	-	-	-	-	-	-	-	-	-
	[-25, 25]		-	-	-	-	-	-	-	-	-	-
	[-37, 37]		-	-	-	-	-	-	-	-	-	-
	[-50, 50]		-	-	-	-	-	-	-	-	-	-
	[-62, 62]		-	-	-	-	-	-	-	-	-	-
	[-75, 75]		-	-	-	-	-	-	-	-	-	-
	[-87, 87]		-	-	-	-	-	-	-	-	-	-
	[-100, 100]		-	-	-	-	-	-	-	-	-	-

Table 6: Number of attack traces to reach guessing entropy equal to 1. Results obtained with the `DPAv4.2` dataset and the Hamming weight leakage model. Neural networks are trained with data augmentation by *generating different augmented traces at each epoch.*

		70k original traces +										
δ_{hid}	$[-\delta_{aug}, \delta_{aug}]$	**0**	**7k**	**14k**	**21k**	**28k**	**35k**	**42k**	**49k**	**56k**	**63k**	**70k**
25	[-6, 6]	714	760	747	673	439	203	589	393	216	286	316
	[-12, 12]		563	486	398	741	484	467	637	460	-	2515
	[-25, 25]		481	519	654	420	352	158	551	582	478	2032
50	[-6, 6]	411	359	367	403	375	197	187	244	222	219	297
	[-12, 12]		538	145	237	206	161	164	351	184	214	275
	[-25, 25]		366	273	188	207	341	204	160	426	211	293
	[-37, 37]		410	199	213	182	195	180	150	343	180	195
75	[-6, 6]	417	1587	782	13	961	-	-	659	-	-	2557
	[-12, 12]		1778	1329	-	462	-	762	-	-	-	1229
	[-25, 25]		1436	1315	9	64	33	-	303	-	-	-
	[-37, 37]		1031	426	-	-	-	178	-	-	-	-
	[-50, 50]		1811	481	-	254	545	13	-	-	-	-
100	[-6, 6]	394	462	401	180	36	379	299	39	-	-	25
	[-12, 12]		700	640	355	25	-	127	40	154	-	352
	[-25, 25]		162	159	19	145	240	141	-	-	-	11
	[-37, 37]		497	225	149	-	45	31	16	-	-	20
	[-50, 50]		144	240	219	36	175	-	53	-	-	12
	[-62, 62]		606	314	188	1713	419	31	-	-	-	73
125	[-6, 6]	460	492	188	140	28	53	64	54	76	64	253
	[-12, 12]		199	42	49	71	37	62	49	17	22	18
	[-25, 25]		105	48	18	31	15	17	169	28	18	15
	[-37, 37]		237	27	38	51	36	27	15	22	20	19
	[-50, 50]		478	21	27	16	10	16	15	15	20	31
	[-62, 62]		834	69	51	15	19	23	14	12	18	15
	[-75, 75]		434	24	20	16	15	15	9	21	10	12
150	[-6, 6]	452	853	351	16	13	-	15	12	-	-	-
	[-12, 12]		634	264	14	10	196	23	-	10	23	-
	[-25, 25]		390	10	-	-	-	-	10	-	-	8
	[-37, 37]		495	10	-	-	-	-	-	-	-	-
	[-50, 50]		24	-	16	12	12	-	-	-	-	-
	[-62, 62]		35	14	12	-	-	-	-	-	-	-
	[-75, 75]		224	16	-	-	-	-	-	-	-	-
	[-87, 87]		26	164	-	32	-	-	-	-	-	-
175	[-6, 6]	1746	1501	1264	1131	943	1310	1185	852	629	1033	555
	[-12, 12]		1858	1307	732	606	1131	697	766	698	314	574
	[-25, 25]		1009	617	988	462	439	460	296	305	662	332
	[-37, 37]		1134	684	608	399	556	672	271	250	529	273
	[-50, 50]		1177	875	465	464	405	322	746	371	174	333
	[-62, 62]		1150	638	571	446	337	316	284	361	260	314
	[-75, 75]		492	709	677	497	379	181	253	315	140	271
	[-87, 87]		1134	599	529	277	293	391	265	246	264	206
	[-100, 100]		500	866	619	475	382	273	291	203	283	196
200	[-6, 6]	1371	-	1621	1501	1961	1379	993	1283	1113	1415	1433
	[-12, 12]		1738	2731	1366	1576	645	1036	1149	1228	550	1151
	[-25, 25]		1193	930	2038	858	679	752	1066	1215	763	702
	[-37, 37]		1674	674	979	757	718	689	490	702	276	235
	[-50, 50]		1983	1004	778	531	486	371	569	795	60	50
	[-62, 62]		1507	1124	1329	1374	204	485	446	102	422	123
	[-75, 75]		1484	1108	1872	500	191	86	347	59	76	87
	[-87, 87]		1451	633	877	738	562	135	61	200	34	38
	[-100, 100]		1634	1242	640	849	997	338	24	35	21	22

Then a new training is deployed for this CNN model with the data augmentation and different numbers of augmented traces. For each number of the augmented traces, the model is trained with Gaussian noise with different standard deviations σ_{aug}. We set this value to ensure $0.5 \leq \sigma_{aug} \leq \sigma_{hid} + 1$. The minimum value of 0.5 for σ_{aug} is to ensure that σ_{aug} is tested at least for a value that is lower than the minimum value considered for σ_{hid}, which is 1.0.

Table 7 presents the efficiency of data augmentation for different CNN architectures with the Identity leakage model. When $\sigma_{hid} = \{1.0, 2.0, 3.0\}$, the $N_{\mathtt{ge}^*=1}$ value obtained for the baseline model is always higher than the lowest value obtained with the best $N_{\mathtt{ge}^*=1}$ when data augmentation is active during training. Take $\sigma_{hid} = 1.0$ for example. When the CNN model is trained without data augmentation, the baseline model can successfully recover the key with 514 traces. When data augmentation with 100 000 augmented traces is applied during training and $\sigma_{aug} = 0.5$, the correct key is recovered with 200 traces. However, if $N_{\mathtt{ge}^*=1} > 3\,000$ is obtained for the baseline model, we observe different scenarios. When $\sigma_{hid} = \{4.0, 6.0\}$, the baseline model cannot successfully recover the key with less than 3 000 traces, and neither can the CNN model do when data augmentation is applied. This suggests that random search should be applied again to return another best CNN model. When $\sigma_{hid} = 5.0$, the baseline model cannot successfully recover the key with less than 3 000 traces. However, when $\sigma_{aug} = \{1.0, 2.0\}$ is adopted, the key can be recovered.

Table 8 presents the efficiency of data augmentation for different CNN architectures with the Hamming weight leakage model. When $\sigma_{hid} = 1.0$, we do not see the performance improvement from data augmentation except in one case with $\sigma_{aug} = 0.5$ and 200 000 augmented traces. When $\sigma_{hid} = 2.0$, there is not a single case where data augmentation can reduce the traces needed to recover the key successfully. We see the improvement from augmentation for $\sigma_{hid} = 3.0$. For example, we obtain $N_{\mathtt{ge}^*=1} = 2\,136$ from the baseline model without data augmentation. When data augmentation with 200 000 augmented traces with $\sigma_{aug} = 0.5$ is applied during training, the correct key candidate is recovered with 1 431 traces. For $\sigma_{hid} = 4.0$, the $N_{\mathtt{ge}^*=1}$ value obtained for the baseline model is always higher than the lowest value obtained with the best $N_{\mathtt{ge}^*=1}$ when data augmentation with $\sigma_{aug} = \{0.5, 1.0\}$ is applied during training. when $\sigma_{hid} = \{5.0, 6.0\}$, the baseline model cannot successfully recover the key with less than 3 000 traces, and neither can the CNN model do when augmentation is applied. This indicates that when Gaussian noise is at a high level, and the SNR is low, it is not easy to recover the correct key successfully, regardless of the fact that data augmentation is used.

`DPAv4.2` Tables 9 and 10 illustrate results for the `DPAv4.2` dataset with Gaussian noise countermeasure applied with the Identity leakage model and the Hamming weight leakage model, respectively. The mean value of Gaussian noise is fixed at 0. The term σ_{hid} refers to the standard deviation in Gaussian noise used to the raw traces for a hiding countermeasure. The term σ_{aug} indicates the standard deviation in Gaussian noise applied to the augmented traces. The training

Table 7: Number of attack traces to reach guessing entropy equal to 1. Results obtained with the `ASCADr` dataset and the Identity leakage model. Neural networks are trained with data augmentation by *generating different augmented traces at each epoch.*

		70k original traces +										
σ_{hid}	σ_{aug}	**0**	**20k**	**40k**	**60k**	**80k**	**100k**	**120k**	**140k**	**160k**	**180k**	**200k**
1.0	0.5	514	642	345	311	598	200	350	304	258	203	288
	1.0		556	416	376	592	336	318	218	351	274	233
	2.0		736	476	376	495	330	273	544	391	304	306
2.0	0.5	1821	-	1393	964	-	-	860	396	560	-	449
	1.0		-	-	-	776	424	344	692	702	577	710
	2.0		-	1976	2673	850	-	-	622	279	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
3.0	0.5	828	-	-	-	-	-	819	-	-	-	1148
	1.0		-	-	-	-	-	-	665	525	317	604
	2.0		-	-	-	-	-	-	-	818	277	751
	3.0		-	-	-	-	-	-	-	-	1822	-
	4.0		-	-	-	-	-	-	-	-	-	-
4.0	0.5	> 3000	-	-	-	-	-	-	-	-	-	-
	1.0		-	-	-	-	-	-	-	-	-	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
5.0	0.5	> 3000	-	-	-	-	-	-	-	-	-	-
	1.0		-	-	-	2544	-	-	868	2478	2938	1950
	2.0		-	-	-	2663	-	2952	2412	2998	-	2169
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
	6.0		-	-	-	-	-	-	-	-	-	-
6.0	0.5	> 3000	-	-	-	-	-	-	-	-	-	-
	1.0		-	-	-	-	-	-	-	-	-	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
	6.0		-	-	-	-	-	-	-	-	-	-
	7.0		-	-	-	-	-	-	-	-	-	-

Table 8: Number of attack traces to reach guessing entropy equal to 1. Results obtained with the `ASCADr` dataset and the Hamming weight leakage model. Neural networks are trained with data augmentation by *generating different augmented traces at each epoch.*

		70k original traces +										
σ_{hid}	σ_{aug}	**0**	**20k**	**40k**	**60k**	**80k**	**100k**	**120k**	**140k**	**160k**	**180k**	**200k**
1.0	0.5	610	1149	698	839	1221	823	908	1112	1174	860	606
	1.0		865	789	1775	1091	928	1001	899	979	1476	1060
	2.0		1265	1334	1874	1528	2004	1958	1977	2212	2041	1946
2.0	0.5	1357	1547	1555	2114	1867	1964	2075	1650	2172	1946	2425
	1.0		1900	1370	2275	1950	1864	2196	1994	1914	1695	1842
	2.0		1460	1686	1653	1924	1682	2561	2512	2208	2683	2397
	3.0		2513	2469	2359	2215	2852	2797	2840	-	-	-
3.0	0.5	2136	1774	2015	2062	2149	1859	1698	1869	1576	1915	1431
	1.0		2338	2153	1949	2255	1952	2451	1946	2103	1889	2068
	2.0		1775	2746	2284	2438	-	-	-	-	-	-
	3.0		2798	2799	-	2989	-	-	-	-	-	-
	4.0		2844	-	-	-	-	-	-	-	-	-
4.0	0.5	2953	2034	2413	2698	2398	1608	2403	2111	2548	2662	2594
	1.0		2993	-	2319	2370	2788	2544	2971	2654	2891	2709
	2.0		-	2630	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
5.0	0.5	2758	-	-	-	-	-	-	-	-	-	-
	1.0		-	-	-	-	-	-	-	-	-	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
	6.0		-	-	-	-	-	-	-	-	-	-
6.0	0.5	> 3000	-	-	-	-	-	-	-	-	-	-
	1.0		-	-	-	-	-	-	-	-	-	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
	6.0		-	-	-	-	-	-	-	-	-	-
	7.0		-	-	-	-	-	-	-	-	-	-

is always conducted for the 70 000 traces plus the augmented traces. The augmented traces denoted by 0 indicate the number of attack traces required to reach $\texttt{ge}^* = 1$ for the baseline model trained *without* data augmentation. For each different σ_{hid} value, the CNN architecture is obtained from a random search. Later, a new training is adopted for this CNN model with data augmentation. For each of these augmented traces, the model is trained with Gaussian noise with different standard deviations σ_{aug}, which is set to $0.5 \leq \sigma_{aug} \leq \sigma_{hid} + 1$.

Table 9 illustrates the efficiency of data augmentation for the `DPAv4.2` dataset with the Identity leakage model. When $\sigma_{hid} = \{1.0, 2.0, 3.0\}$, the $N_{\texttt{ge}^*=1}$ value obtained for the baseline model is always higher than the lowest value obtained with the best $N_{\texttt{ge}^*=1}$ when data augmentation is active during training. Take $\sigma_{hid} = 1.0$ for example. When the CNN model is trained without data augmentation, the model can successfully recover the key with 54 traces. When data augmentation with 42 000 augmented traces is applied during training and $\sigma_{aug} = 0.5$, the correct key candidate is recovered with 24 traces. However, if $N_{\texttt{ge}^*=1} > 3\,000$ is obtained from the baseline model, we can see different cases. When $\sigma_{hid} = \{4.0, 6.0\}$, the baseline model cannot successfully recover the key with less than 3 000 traces, and neither can the CNN model when augmentation is applied. When $\sigma_{hid} = 5.0$, the baseline model cannot successfully recover the key with less than 3 000 traces. We obtain $N_{\texttt{ge}^*=1} = 1\,884, 1\,794$ when 42 000 training augmented trace and $\sigma_{aug} = 0.5$, and 63 000 training augmented trace and $\sigma_{aug} = 0.1$ are applied, respectively.

Table 10 presents the efficiency of data augmentation for the `DPAv4.2` dataset with the Hamming weight leakage model. When $\sigma_{hid} = \{1.0, 4.0\}$, we do not observe the performance improvement from data augmentation. When $\sigma_{hid} = \{2.0, 3.0\}$, the $N_{\texttt{ge}^*=1}$ value obtained for the baseline model is always higher than the lowest value obtained with the best $N_{\texttt{ge}^*=1}$ when data augmentation is active during training. When $\sigma_{hid} = 5.0$, the CNN model can successfully recover the key with 2 025 traces. When data augmentation with 21 000 augmented traces and $\sigma_{aug} = 0.5$ is applied during training, the correct key candidate is recovered with only 1 273 traces. For $\sigma_{hid} = 6.0$, we obtain $N_{\texttt{ge}^*=1} > 3\,000$, and get $N_{\texttt{ge}^*=1} = 2\,479$ when data augmentation is applied with 35 000 augmented traces with $\sigma_{aug} = 1.0$.

5.3 Discussion

Based on the obtained results, some general guidelines can be given:

- Is there a single best data augmentation setting for all cases?
  We see that different settings (datasets, neural network architectures, leakage models) require different data augmentation settings, making the hyperparameter tuning even more complex. At the same time, we deem this effort well spent as the attack performance can improve significantly when careful data augmentation is conducted.
- What countermeasure is more difficult?
  We observe that the Gaussian noise countermeasure is more difficult to break

Table 9: Number of attack traces to reach guessing entropy equal to 1. Results obtained with the `DPAv4.2` dataset and the Identity leakage model. Neural networks are trained with data augmentation by *generating different augmented traces at each epoch*.

		70k original traces +										
σ_{hid}	σ_{aug}	**0**	**7k**	**14k**	**21k**	**28k**	**35k**	**42k**	**49k**	**56k**	**63k**	**70k**
1.0	0.5	54	33	43	38	46	47	24	29	33	44	42
	1.0		38	30	44	43	49	42	49	63	42	60
	2.0		39	54	-	-	113	109	67	105	93	-
2.0	0.5	99	95	87	69	66	46	6	29	8	6	8
	1.0		69	66	95	43	45	7	22	11	10	7
	2.0		142	174	119	221	116	197	219	-	329	-
	3.0		249	519	456	-	208	-	-	-	-	-
3.0	0.5	2426	1671	1149	878	756	983	895	617	392	672	299
	1.0		1594	-	-	1642	-	1765	972	821	868	411
	2.0		-	2192	-	-	2443	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
4.0	0.5	> 3000	-	-	-	-	-	-	-	-	-	-
	1.0		-	-	-	-	-	-	-	-	-	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
5.0	0.5	> 3000	-	-	-	-	-	1884	-	-	-	-
	1.0		-	-	-	-	-	-	-	-	1794	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
	6.0		-	-	-	-	-	-	-	-	-	-
6.0	0.5	> 3000	-	-	-	-	-	-	-	-	-	-
	1.0		-	-	-	-	-	-	-	-	-	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
	6.0		-	-	-	-	-	-	-	-	-	-
	7.0		-	-	-	-	-	-	-	-	-	-

Table 10: Number of attack traces to reach guessing entropy equal to 1. Results obtained with the `DPAv4.2` dataset and the Hamming weight leakage model. Neural networks are trained with data augmentation by *generating different augmented traces at each epoch.*

		70k original traces +										
σ_{hid}	σ_{aug}	**0**	**7k**	**14k**	**21k**	**28k**	**35k**	**42k**	**49k**	**56k**	**63k**	**70k**
1.0	0.5	15	601	-	70	80	-	101	42	29	52	-
	1.0		-	-	60	-	334	786	196	59	-	98
	2.0		-	-	160	577	-	-	-	-	-	-
2.0	0.5	64	541	31	42	75	90	39	41	47	38	54
	1.0		59	68	75	46	46	82	92	70	76	54
	2.0		138	64	114	879	318	264	275	439	593	343
	3.0		831	1431	674	1987	2991	-	-	-	1149	2408
3.0	0.5	719	2085	2206	1948	2084	1701	2024	805	1086	696	1076
	1.0		2752	1282	2686	992	2490	1129	1122	1511	1187	833
	2.0		2758	1944	-	2225	-	1875	1281	-	-	-
	3.0		-	-	2079	-	-	-	-	-	-	-
	4.0		1827	-	-	2989	-	-	-	-	-	-
4.0	0.5	961	-	-	1452	-	1810	-	1396	1392	1128	1215
	1.0		-	-	-	1844	-	-	1794	-	-	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
5.0	0.5	2025	-	-	1273	-	2691	-	-	-	2107	2778
	1.0		-	-	-	-	-	-	-	-	-	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
	6.0		-	-	-	-	-	-	-	-	-	-
6.0	0.5	> 3000	-	-	-	-	-	-	-	-	-	-
	1.0		-	-	-	-	2479	-	-	-	-	-
	2.0		-	-	-	-	-	-	-	-	-	-
	3.0		-	-	-	-	-	-	-	-	-	-
	4.0		-	-	-	-	-	-	-	-	-	-
	5.0		-	-	-	-	-	-	-	-	-	-
	6.0		-	-	-	-	-	-	-	-	-	-
	7.0		-	-	-	-	-	-	-	-	-	-

using data augmentation.
For both datasets, we can use data augmentation to get significant improvement for recovering the correct key under desynchronization countermeasure, even when δ_{hid} is at a high level, such as 175 or 200. This is because of the shift-invariant property of CNN, which can extract the points of interest in traces even when the misalignment of traces is large.
When the Gaussian noise countermeasure is applied, usually, we can see some improvement when using data augmentation for $\sigma_{hid} < 5$. When increasing the σ_{hid} to 5 or 6, we often cannot recover the correct key using the baseline model or data augmentation techniques because of the low SNR level and the shift-invariant property of CNN, which cannot contribute to the reduction of noise.

- Is there a range for the efficiency of data augmentation?
  For the desynchronization countermeasure, we often observe the performance improvement from data augmentation when the number of augmented traces is above some value. Take the ASCAD dataset with $\delta_{hid} = 100$, for example. The $N_{\texttt{ge}^*=1}$ from data augmentation is always lower than that from the baseline model when the number of augmented traces is larger than 120 000 and 40 000 for the Identity and Hamming weight leakage model, respectively. For the DPAv4.2 dataset with $\delta_{hid} = 100$, the data augmented trace range is greater than 7 000 and 28 000 for the two leakage models.
  At the same time, for the Gaussian noise countermeasure, we do not observe this phenomenon.

6 Conclusions and Future Work

In this paper, we evaluated the influence of data augmentation on deep learning-based SCA and verified to what extent it can reduce the protective effect of hiding countermeasures. We applied our analysis to two public datasets with masked AES implementations. We apply desynchronization and Gaussian noise to the original measurements to create a hiding countermeasure effect. We first add the hiding countermeasure to the chosen datasets and then deploy a hyperparameter random search to obtain the best CNN model for each hiding countermeasure case. Later, to investigate how to properly implement data augmentation for specific models, we deploy new training for each CNN model by considering data augmentation with different numbers of augmented traces and different data augmentation hyperparameters, such as range of trace shifts and standard deviations. Our results show that data augmentation can decrease the efficiency of hiding countermeasures to protect the secret key for different datasets. In particular, we can improve a CNN model generalization by making the model trained with data augmentation to recover the key with less than 50 attacked traces for the ASCADr dataset and a single attack trace for the DPAv4.2 dataset. These are the best results against trace desynchronization reported in the literature so far for these datasets. However, different data augmentation configurations are required for specific neural network architectures to provide the best behavior.

More datasets and different neural network architectures will be studied in future work. Additionally, more countermeasures and augmentation techniques, such as time warping and SMOTE, can be adopted. Here, we investigated hiding countermeasures techniques separately. We will also investigate how combined data augmentation strategies could defeat the combination of multiple hiding countermeasures.

References

1. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ascad database. Journal of Cryptographic Engineering **10**(2), 163–188 (2020)
2. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International workshop on cryptographic hardware and embedded systems. pp. 16–29. Springer (2004)
3. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 45–68. Springer (2017)
4. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3, https://doi.org/10.1007/3-540-36400-5_3
5. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation strategies from data. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 113–123 (2019)
6. Cubuk, E.D., Zoph, B., Shlens, J., Le, Q.V.: Randaugment: Practical automated data augmentation with a reduced search space. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. pp. 702–703 (2020)
7. Fawzi, A., Samulowitz, H., Turaga, D., Frossard, P.: Adaptive data augmentation for image classification. In: 2016 IEEE international conference on image processing (ICIP). pp. 3688–3692. Ieee (2016)
8. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6572
9. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 148–179 (2019)
10. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25, https://doi.org/10.1007/3-540-48405-1_25

11. Luo, Z., Zheng, M., Wang, P., Jin, M., Zhang, J., Hu, H.: Towards strengthening deep learning-based side channel attacks with mixup. In: 20th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2021, Shenyang, China, October 20-22, 2021. pp. 791–801. IEEE (2021). https://doi.org/10.1109/TrustCom53373.2021.00114, `https://doi.org/10.1109/TrustCom53373.2021.00114`
12. Mikołajczyk, A., Grochowski, M.: Data augmentation for improving deep learning in image classification problem. In: 2018 international interdisciplinary PhD workshop (IIPhDW). pp. 117–122. IEEE (2018)
13. Mukhtar, N., Batina, L., Picek, S., Kong, Y.: Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices. In: Galbraith, S.D. (ed.) Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13161, pp. 297–321. Springer (2022). https://doi.org/10.1007/978-3-030-95312-6_13, `https://doi.org/10.1007/978-3-030-95312-6_13`
14. Perin, G., Chmielewski, L., Batina, L., Picek, S.: Keep it unsupervised: Horizontal attacks meet deep learning. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(1), 343–372 (2021). https://doi.org/10.46586/tches.v2021.i1.343-372, `https://doi.org/10.46586/tches.v2021.i1.343-372`
15. Perin, G., Chmielewski, L., Picek, S.: 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). https://doi.org/10.13154/tches.v2020.i4.337-364, `https://tches.iacr.org/index.php/TCHES/article/view/8686`
16. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 209–237 (Nov 2018). https://doi.org/10.13154/tches.v2019.i1.209-237, `https://tches.iacr.org/index.php/TCHES/article/view/7339`
17. Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: Sok: Deep learning-based physical side-channel analysis. IACR Cryptol. ePrint Arch. p. 1092 (2021), `https://eprint.iacr.org/2021/1092`
18. Picek, S., Samiotis, I.P., Kim, J., Heuser, A., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 157–176. Springer (2018)
19. Pu, S., Yu, Y., Wang, W., Guo, Z., Liu, J., Gu, D., Wang, L., Gan, J.: Trace augmentation: What can be done even before preprocessing in a profiled sca? In: International Conference on Smart Card Research and Advanced Applications. pp. 232–247. Springer (2017)
20. Rijsdijk, J., Wu, L., Perin, G.: Reinforcement learning-based design of side-channel countermeasures. In: Batina, L., Picek, S., Mondal, M. (eds.) Security, Privacy, and Applied Cryptography Engineering - 11th International Conference, SPACE 2021, Kolkata, India, December 10-13, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13162, pp. 168–187. Springer (2021). https://doi.org/10.1007/978-3-030-95085-9_9, `https://doi.org/10.1007/978-3-030-95085-9_9`
21. Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. Journal of big data **6**(1), 1–48 (2019)

22. Standaert, F.X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 443–461. Springer (2009)
23. Wang, J., Perez, L., et al.: The effectiveness of data augmentation in image classification using deep learning. Convolutional Neural Networks Vis. Recognit **11**, 1–8 (2017)
24. van Woudenberg, J.G.J., Witteman, M.F., Bakker, B.: Improving differential power analysis by elastic alignment. In: Kiayias, A. (ed.) Topics in Cryptology – CT-RSA 2011. pp. 104–119. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
25. Wouters, L., Arribas, V., Gierlichs, B., Preneel, B.: Revisiting a methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 147–168 (Jun 2020). https://doi.org/10.13154/tches.v2020.i3.147-168, `https://tches.iacr.org/index.php/TCHES/article/view/8586`
26. Zaid, G., Bossuet, L., Dassance, F., Habrard, A., Venelli, A.: Ranking loss: Maximizing the success rate in deep learning side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(1), 25–55 (2021). https://doi.org/10.46586/tches.v2021.i1.25-55, `https://doi.org/10.46586/tches.v2021.i1.25-55`
27. Zhang, H., Cissé, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net (2018), `https://openreview.net/forum?id=r1Ddp1-Rb`
28. Zhou, Y., Standaert, F.: Deep learning mitigates but does not annihilate the need of aligned traces and a generalized resnet model for side-channel attacks. J. Cryptogr. Eng. **10**(1), 85–95 (2020). https://doi.org/10.1007/s13389-019-00209-3, `https://doi.org/10.1007/s13389-019-00209-3`