Gambling for Success: The Lottery Ticket Hypothesis in Deep Learning-based SCA

Guilherme Perin[1], Lichao Wu[1] and Stjepan Picek[1]

Delft University of Technology, The Netherlands

Abstract. Deep learning-based side-channel analysis (SCA) became the de facto standard in the profiling SCA. Still, this does not mean it is trivial to find neural networks that perform well for any setting. Based on the developed neural network architectures, we can distinguish between small neural networks that are easier to tune and less prone to overfitting but can have insufficient capacity to model the data. On the other hand, large neural networks would have sufficient capacity but can overfit and are more difficult to tune. This always brings an interesting trade-off between simplicity and performance.

This paper proposes using a pruning strategy and recently proposed Lottery Ticket Hypothesis to improve the deep learning-based SCA. We demonstrate that we can find smaller neural networks that perform on the level of larger networks, where we manage to reduce the number of weights by more than 90% on average. Additionally, we show that pruning can help prevent overfitting and the effects of imbalanced data, reaching top attack performance for small networks when larger networks do not manage to break the target at all.

Keywords: Side-channel Analysis · Deep learning · Lottery Ticket Hypothesis · Pruning

1 Introduction

There are several side-channel analysis (SCA) approaches to exploit various sources of information leakage in the device in the last few decades, like timing [Koc96], power [KJJ99], and electromagnetic (EM) emanation [QS01]. One standard division of side-channel analyses is into non-profiling and profiling attacks. Non-profiling attacks require fewer assumptions but often require thousands of measurements (traces) to break a target, especially if protected with countermeasures. Profiling attacks are considered one of the strongest possible attacks as the attacker has control over a clone device to build its complete profile [CRR02]. The attacker then uses this profile to target other similar devices to recover the secret information. The history of profiling side-channel analysis (SCA) is relatively short, and in less than 20 years, we can recognize several research directions. The first direction used techniques like (pooled) template attack [CRR02, CK14] or stochastic models [SLP05] and managed to improve the attack performance over non-profiling attacks significantly. Then, the second direction moved toward machine learning in SCA, and again, a plethora of results [PHJ+17, HZ12, LPB+15] indicated that machine learning could outperform other profiling SCAs. More recently, as the third direction, we see a change of focus to deep learning techniques. Intuitively, we can find at least two reasons for this: 1) deep learning can break targets protected with countermeasures, and 2) deep learning does not require pre-processing like feature selection [PHJB19] or dimensionality reduction [APSQ06]. While the SCA community progressed quite far in the deep learning-based SCA in just a few years, there are many knowledge gaps. One example would be

how to successfully and consistently find neural networks that manage to break various targets.

Thus, we still need to find approaches that allow obtaining neural networks that perform well for various settings. Even more, we would want to have an approach that manages to transform a good performing architecture for one setting to a good performing architecture for some different setting. Finally, it would be ideal if the top-performing architectures could be small (so they are more computationally efficient, and hopefully, easier to understand). Unfortunately, this is not an easy task as the search space of all possible neural networks is huge, and there are no general guidelines on how to construct a neural network that will break the target. Current efforts mainly concentrate on finding better hyperparameters by random search, Bayesian optimization [WPP20], reinforcement learning [RWPP21], or following a specific methodology [ZBHV19]. Still, there are alternatives to how to provide neural networks that are small and perform well.

In the machine learning domain, there is a technique called pruning that refers to a systematical removal of parameters from existing neural networks. Commonly, pruning is used on large neural networks that show good performance, and the goal is to produce a smaller network with similar performance. While pruning [Jan89, BOFG20] is a rather standard technique in deep learning, it has not been investigated before for SCA to the best of our knowledge. Similarly, the lottery ticket hypothesis [FC18] attracted quite some attention in the machine learning community, but none (as far as we know) in the SCA community. On the other hand, there are several attempts at creating methodologies for deep learning-based SCA [ZBHV19, WAGP20], but there are some issues. First, it is not easy to use those methodologies and generalize for other datasets or neural network architectures. Second, the conflicting results among those methodologies indicate it is difficult to find a single approach that works the best for everything.

This paper applies the recent *Lottery Ticket Hypothesis* in the profiling side-channel analysis. After training a (large) neural network, we apply the pruning process by removing the activity of small weights from the neural network. We then re-initialize the pruned neural network with the same initial weights set for the original large neural network. The pruned and re-initialized network shows equal and, most of the time, superior performance compared to the large trained network. The results demonstrate that when the large network cannot reach a successful attack (low guessing entropy), applying the lottery ticket hypothesis leads to a successful key recovery, even when the number of profiling traces is low. More importantly, we verify that when training a large deep neural network provides guessing entropy close to a random guess, a pruned and re-initialized neural network can successfully recover the key. Our two main contributions are:

1. We introduce the pruning approach into profiling SCA, which enables us to propose a "methodology" that can work on top of other approaches. Our approach is applicable to any neural network, regardless of whether it is selected randomly or obtained through some other methodology. Naturally, depending on how good is the original network, the results from our approach can differ. As reported in this paper, we can find smaller and better performing networks by using the pruning methodology, even when the original network does not work.
2. We investigate how pruning can be a useful strategy when using a single neural network to attack the whole key (and not just a single key byte as commonly reported in the literature). Indeed, pruning and re-initializing proved to be very powerful options for adjusting the neural network to different settings.

2 Background

In this section, we first present the notation we use. Afterward, we discuss the profiling SCA with machine learning. Finally, we discuss the SCA datasets we use in our experiments.

2.1 Notation

Let 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. Next, let k be a key candidate that takes its value from the keyspace $\mathcal{K}$, and k^* the correct key. We define a dataset as a collection of traces $\mathbf{T}$, where each trace $\mathbf{t}_i$ is associated with an input value (plaintext or ciphertext) $\mathbf{d}_i$ and a key $\mathbf{k}_i$. When considering only a specific key byte j, we denote it as $k_{i,j}$, and input byte as $d_{i,j}$.

The dataset consists of $|T|$ traces. From there, we take N traces for the profiling set, V for the validation set, and Q for the attack set. Finally, $\boldsymbol{\theta}$ denotes the vector of parameters to be learned in a profiling model, and $\mathcal{H}$ denotes the hyperparameters defining the profiling model.

2.2 Machine Learning Techniques

We consider two common choices in machine learning-based SCA: multilayer perceptron (MLP) and convolutional neural networks (CNNs). For details on the performance of those techniques in SCA, we refer interested readers to Section 3.

2.2.1 Multilayer Perceptron

The multilayer perceptron is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. MLP consists of multiple layers (at least three: an input layer, one hidden layer, and one output layer) of nodes in a directed graph, where each layer is fully connected to the next one.

2.2.2 Convolutional Neural Networks

Convolutional neural networks commonly consist of three types of layers: 1) convolution layers, 2) pooling layers, and 3) fully-connected layers. The convolution layer computes neurons' output 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.

2.3 Supervised Machine Learning in Profiling SCA

Supervised machine learning considers the machine learning task of learning a function f mapping an input X to the output Y ($f : \mathcal{X} \rightarrow Y$)) based on 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, which corresponds to SCA's profiling and attack phases. Thus, in the rest of this paper, we use the terms profiling/training and attack/testing interchangeably. As the function f, we consider a deep neural network with the *Softmax* output layer.

1. The goal of the training phase is to learn parameters $\boldsymbol{\theta}'$ that minimize the empirical risk represented by a loss function L on a dataset T of size N.
2. In the attack phase, the goal is to make predictions about the classes

$$y(t_1, k^*), \ldots, y(t_Q, k^*),$$

where k^* represents the secret (unknown) key on the device under the attack (or the key byte). 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 number of classes c depends on the leakage model as the class v is derived from the key and input through a cryptographic function and a leakage model). Every element $\mathbf{p}_{i,v}$ of matrix P is a vector of all class probabilities for a specific trace $\mathbf{x}_i$ ($\sum_v^c \mathbf{p}_{i,v} = 1, \forall i$. To reach the probability that a certain key k is the correct one, we use the maximum log-likelihood 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.

Note that with machine learning algorithms, we are interested in reaching good generalization, where the generalization refers to how well the concepts learned by a machine learning model apply to previously unseen examples. At the same time, we aim to avoid underfitting and overfitting. Overfitting happens when a model learns the detail and noise in the training data, negatively impacting the model's performance on unseen data. Underfitting happens with a model that cannot model the training data or generalize to unseen data.

In SCA, an adversary is not interested in predicting the classes in the attack phase but obtaining the secret key k^*. To estimate the effort required to obtain the key, we will use the guessing entropy (GE) or success rate metrics [SMY09]. An attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in decreasing order of probability, which means that g_1 is the most likely key candidate and $g_{|\mathcal{K}|}$ the least likely key candidate. The success rate is the average probability that the secret key k^* is the first element of the key guessing vector $\mathbf{g}$. Guessing entropy is the average position of k^* in $\mathbf{g}$. Commonly, averaging is done over 100 independent experiments to obtain statistically significant results. While we defined here guessing entropy for the whole key, it can also be observed for separate key bytes, in which case it is called partial guessing entropy. We will use the notion of guessing entropy for both the whole key and key byte, as the meaning will be clear from the attack context.

2.4 Datasets and Leakage Models

During the execution of the cryptographic algorithm, the processing of sensitive information produces a certain leakage. In this paper, we consider the Hamming weight leakage model.There, the attacker assumes the leakage is proportional to the sensitive variable's Hamming weight. When considering a cipher that uses an 8-bit S-box, this leakage model results in nine classes ($c = 9$).

2.4.1 ASCAD Datasets.

The first target platform we consider is an 8-bit AVR microcontroller running a masked AES-128 implementation [BPS+20]. There are two versions of the ASCAD dataset. The first version of the ASCAD dataset has a fixed key and 50 000 traces for profiling and 10 000 for testing.The second version of the ASCAD dataset has random keys, and it consists of 200 000 traces for profiling and 100 000 for testing. For both versions, we attack the key byte 3 unless specified differently. For the ASCAD dataset, the third key byte is the first masked byte. For ASCAD with the fixed key, we use a pre-selected window of 700 features, while for ASCAD with random keys, the window size equals 1 400 features. These datasets are available at https://github.com/ANSSI-FR/ASCAD.

2.4.2 CHES CTF Dataset.

This dataset refers to the CHES Capture-the-flag (CTF) AES-128 dataset, released in 2018 for the Conference on Cryptographic Hardware and Embedded Systems (CHES). The traces consist of masked AES-128 encryption running on a 32-bit STM microcontroller. We use 45 000 traces for the training set, which contains a fixed key. The attack set consists of 5 000 traces. The key used in the training and validation set is different from the key configured for the test set. Each trace has 2 200 features. This dataset is available at https://chesctf.riscure.com/2018/news.

3 Related Works

The goal of finding neural networks that perform well in SCA is probably the most explored direction in machine learning-based SCA. The first works commonly considered multilayer perceptron and reported good results even though there were not many available details about hyperparameter tuning or the best-obtained architectures [GHO15, MHM14, YZLC12, HPGM16]. In 2016, Maghrebi et al. made a significant step forward in the profiling SCA as they investigated the performance of convolutional neural networks [MPP16]. Since the results were promising, this paper started a series of works where deep learning techniques (most dominantly MLP and CNNs) were used to break various targets with as little as possible attack traces.

Soon after, works from Cagli et al. [CDP17], Picek et al. [PHJ+18], and Kim et al. [KPH+19] demonstrated that deep learning could efficiently break implementations protected with countermeasures. While those works also discuss hyperparameter tuning, it was still not straightforward to understand the effort required to find the neural networks that performed well. This effort became somewhat clearer after Benadjila et al. discussed hyperparameter tuning for the ASCAD dataset [BPS+20]. Indeed, while considering only a subset of possible hyperparameters, the tuning process was far from trivial.

Zaid et al. proposed a methodology for CNNs for profiling SCA [ZBHV19]. While the methodology has limitations, the results obtained are significant as we reached top performance with never smaller deep learning architectures. This direction is further investigated by Wouters et al. [WAGP20] who reported some issues with [ZBHV19] but managed to find even smaller neural networks that perform similarly well. Perin et al. conducted a random search in pre-defined ranges to build deep learning models to form ensembles [PCP20]. Their findings showed that even random search (when working on some reasonable range of hyperparameters) could find neural networks that perform extremely well.

Thus, while the methodologies mentioned in the previous paragraph work as evident through the excellent attack performance, there are still questions that are left unanswered. What is clear is that we can reach good results with (relatively) small neural networks. What remains to be answered is how to adapt those methodologies for different datasets, or can we find even smaller neural network architectures that perform as well (or even better). We aim to provide the answers to those questions in this work.

4 The Lottery Ticket Hypothesis (LTH)

The *Lottery Ticket Hypothesis* was originally proposed by Franke and Carbin in [FC18]: *"**Lottery Ticket Hypothesis:** a randomly initialized dense neural network contains a sub-network that is initialized such that - when trained in isolation - it can match the test accuracy of the original network after training for at most the same number of iterations".*

Simplified, the hypothesis assumes that randomly initialized deep neural networks contain sub-networks that, when trained in isolation (without considering other parts of

the network), reach test accuracy comparable to the original network in a similar number of iterations. The sub-networks are obtained by pruning the original network. The authors of [FC18] observed that the sub-network obtained after pruning (with a sparsity level of $P_{\%}$) provides superior performance when re-initialized with the weights used to initialize the original weights. These top-performing sub-networks are then called the *winning tickets*. The sparsity denotes the percentage of the removed network (e.g., 90% sparsity on an MLP consisting of 100 neurons would remove 90 neurons).

4.1 One-shot Pruning

Pruning is a popular deep learning method to reduce a trained model's size while keeping efficient computation during inference time and with minimal loss in accuracy. One of the main reasons for pruning is compression, allowing complex trained models to run on devices with limited resources. However, to find an efficient pruned network, the large overparameterized baseline model (the baseline model refers to the trained neural network architecture that is not pruned) still needs to be trained before applying pruning to remove unnecessary weights.

As we show in the experimental results section, the lottery ticket hypothesis also works when considering other than accuracy performance metrics (e.g., success rate, guessing entropy). What is more, once the efficient pruned model is created, the overparameterized model needs to be trained for one AES key byte, and the pruned model can be re-initialized and trained (with more efficiency) for the remaining key bytes. Therefore, the lottery ticket hypothesis reduces complexity in the application of deep learning in profiling SCA. We give the one-shot pruning procedure for LTH in Algorithm 1.

Algorithm 1 One-shot Pruning

procedure ONE-SHOT PRUNING(original neural network f, original dataset x, random initial weights θ_0, training epoch θ_j, trained weight θ_j, pruning ratio $P_{\%}$, mask m)
  $\theta_j \leftarrow$ Pretrain Model $f(x,\theta_0)$ for j epochs
  $m \leftarrow$ Prune $P_{\%}$ of the smallest weights from θ_j
  for i=1 to j **do**
    Train $f(x,\theta_0 \odot m)$
  end for
end procedure

In the pruning process, the "smallest weights" are selected according to their absolute magnitude. In a neural network, the activations in a forward propagation are mostly affected by larger weight values. Therefore, pruning the smallest weights remove those weights that are not significantly impacting the predictions.

In Figure 1, we depict one-shot pruning procedure. The first part of the figure displays the "usual" procedure where the weights at the beginning of the training process are different from those at epoch A and epoch B. The lower figure shows the setup when we prune the smallest weights, and we are left with the choice of whether we randomly initialize the remaining weights or re-initialize them from the original weights.

4.2 Winning Tickets in Profiling SCA

In [FC18], a *winning ticket* is defined as a sub-network that, when trained in isolation (after being re-initialized with the same baseline model initial weights), provides classification accuracy equivalent or superior to the baseline model. For profiling SCA, we define a winning ticket as a sub-network that provides a test guessing entropy lower than or equivalent to the guessing entropy obtained from the original baseline model. Note that

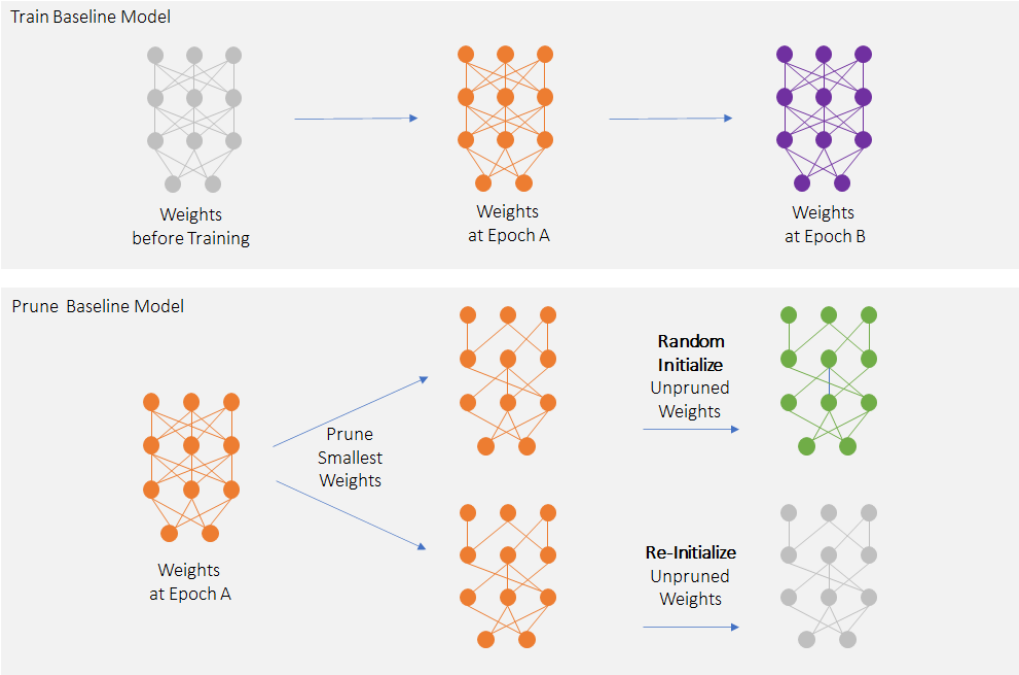


Figure 1: One-shot pruning procedure for LTH.

hyperparameters defined for the baseline model (number of training parameters, number of epochs) and the number of profiling traces directly affect the chances to identify a winning ticket as demonstrated in Section 5.

5 Experimental Results

This section provides experimental results with different neural network architectures and three publicly available SCA datasets. First, we describe the chosen MLP and CNN models. Next, we demonstrate that the lottery ticket hypothesis procedure can provide, in numerous cases, superior results compared to the performance of the baseline model.

5.1 Baseline Neural Networks

In our experiments, we define six different baseline models: three MLPs and three CNNs. Here, the main idea is to demonstrate how pruning and weight re-initialization (the lottery ticket hypothesis) provide different side-channel analysis results if the baseline model varies in size or capacity. The models are selected based on the sizes of commonly used architectures from the related works.

Table 1 lists the hyperparameter configurations for MLP4, MLP6, and MLP8 models. The main idea is to verify how pruning and re-initialization work for MLP architectures with different numbers of dense layers and, consequently, different number of trainable parameters. Note that we have not selected very large neural network models. All of them contain less than one million trainable parameters. Here, the goal is to demonstrate that even an already not so large model can be significantly reduced according to the lottery ticket hypothesis procedure presented in Algorithm 1 and still keep or provide superior profiling SCA results. The principle also holds for chosen CNN models. Table 2 shows the three CNN architectures, named CNN3, CNN4, and CNN4-2. We defined relatively

small CNNs, which appear to be sufficient to break the evaluated datasets. CNN3 has only one convolution layer, while CNN4 and CNN4-2 contain two convolution layers each. In particular, CNN4-2 has larger dense layers compared to CNN4. It is important to notice that we are defining the same models for three different datasets, and it is expected that for baseline models (without pruning), the performance might not be optimal for all evaluated datasets. Although it is out of this paper's scope to identify one model that generalizes well for all scenarios, we demonstrate that applying the lottery ticket hypothesis procedure is a step forward in this important deep learning-based profiling SCA research direction.

Table 1: MLP architectures (batch size 400, learning rate 0.001, ADAM, *selu* activation functions). Number of parameters vary for different datasets due to different input layer dimensions.

Layer	**MLP4**	**MLP6**	**MLP8**
Dense_1	200 neurons	200 neurons	200 neurons
Dense_2	200 neurons	200 neurons	200 neurons
Dense_3	200 neurons	200 neurons	200 neurons
Dense_4	200 neurons	200 neurons	200 neurons
Dense_5	-	200 neurons	200 neurons
Dense_6	-	200 neurons	200 neurons
Dense_7	-	-	200 neurons
Dense_8	-	-	200 neurons
Softmax	9 neurons	9 neurons	9 neurons
Parameters (ASCAD Random Keys)	402 609	483 009	563 409
Parameters (ASCAD Fixed Key)	262 609	343 009	423 409
Parameters (CHES CTF 2018)	562 609	643 009	723 409

Table 2: CNN architectures (batch size 400, learning rate 0.001, ADAM, *selu* activation function). Number of parameters vary for different datasets due to different input layer dimensions.

Layer	**CNN3**	**CNN4**	**CNN4-2**
Conv1D_1	16 filters ks=10, stride=5	16 filters ks=10, stride=5	16 filters ks=10, stride=5
MaxPool1D_1	ks=2, stride=2	ks=2, stride=2	ks=2, stride=2
-	BatchNorm	BatchNorm	BatchNorm
Conv1D_2	-	16 filters ks=10, stride=5	16 filters ks=10, stride=5
MaxPool1D_2	-	ks=2, stride=2	ks=2, stride=2
-	-	BatchNorm	BatchNorm
Dense_1	128 neurons	128 neurons	256 neurons
Dense_2	128 neurons	128 neurons	256 neurons
Softmax	9 neurons	9 neurons	9 neurons
Parameters (ASCAD Random Keys)	302 713	47 305	124 489
Parameters (ASCAD Fixed Key)	159 353	32 969	95 817
Parameters (CHES CTF 2018)	466 553	63 689	157 257

Next, we provide experimental results demonstrating that the procedure described in Section 4 depends on several aspects such as the number of profiling traces, number of parameters in the baseline (original) network, and sparsity level in the pruning process. By identifying the optimal sparsity level in pruning, we can drastically improve the performance of re-initialized sub-networks. Moreover, in some scenarios, we show that even when a large baseline model cannot recover the key, the pruned and re-initialized sub-network succeeds, especially when the number of profiling traces is reduced.

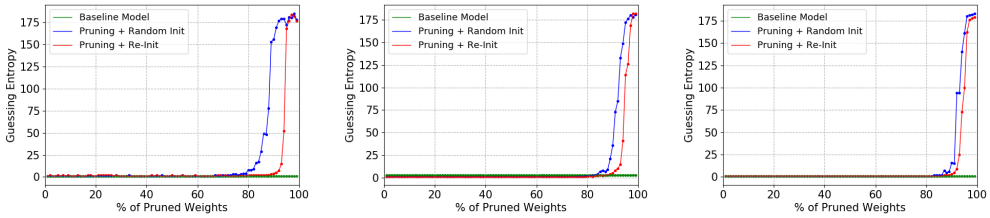
Interpreting Plots:

This section's results are given in terms of guessing entropy for different baseline models, datasets, and sparsity levels. The sparsity level is provided in the x-axis, where we apply pruning to the trained baseline neural network from 1% up to 99%. In each plot, there is a green line that represents the resulting guessing entropy for the baseline model *without pruning*. Thus, the green line is shown together with the plots to indicate what would be the obtained guessing entropy when baseline models are trained for 300 epochs without any pruning. The blue line is the resulting guessing entropy after the trained baseline model is pruned according to the indicated sparsity level (x-axis) and initialized with *random weights* and trained for 50 epochs. Finally, the red line is the resulting guessing entropy from the same previous pruned model and re-initialized with *initial weights* from the baseline model and trained for 50 epochs. For each sparsity level, each experiment is repeated ten times. Therefore, each plot results from training $98 \times 2 \times 10 = 1\,960$ pruned models.

We briefly discuss the limits that the pruning and the Lottery Ticket Hypothesis offer regarding the results and their explainability:
✓Pruning allows making smaller neural networks that perform on the level or even better than larger neural networks.
✓The Lottery Ticket Hypothesis assumes that there will be smaller, good performing sub-networks, so-called winning tickets.
✓Winning tickets in profiling SCA allow reaching small sub-networks with good attack performance, as measured with GE and SR.
✗Pruning and LTH do not provide explainability, and as such, it can be difficult to explain (in the sense of understanding what the neural network learns) what exactly happens in a neural network after pruning.

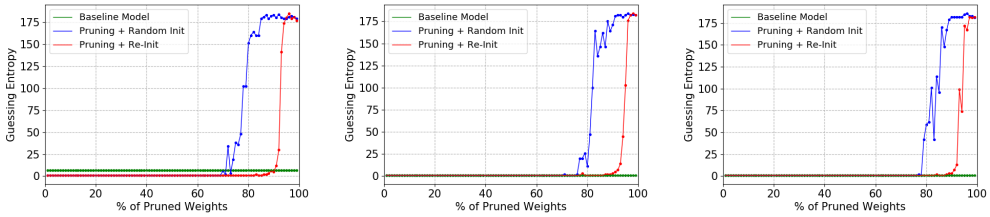
5.2 ASCAD with a Fixed Key

Figures 2, 3, and 4 provide results for the ASCAD fixed key dataset when MLP4, MLP6, and MLP8 are used as baseline models, respectively, for different number of profiling traces. Looking at these figures, we can immediately conclude that pruning and re-initializing the neural network according to the LTH procedure provides similar results regardless of the size of MLP. Next, the number of profiling traces used in each model does not affect these pruned models' performance. If the pruned models are initialized with random weights, the model's performance is directly related to model size and the number of profiling traces. Adding more profiling traces gradually improves the behavior of the model that is randomly initialized, approaching the behavior of the model that is re-initialized. The baseline model performs better than the pruned model that uses random initialization if the percentage of pruned weights is larger than 50% and the number of profiling traces is sufficient to build a strong model. For the pruned model that is re-initialized, the baseline model performs better only if we prune more than 90% of weights. Interestingly, we can observe a marginally better performance of the pruned and re-initialized model for MLP6



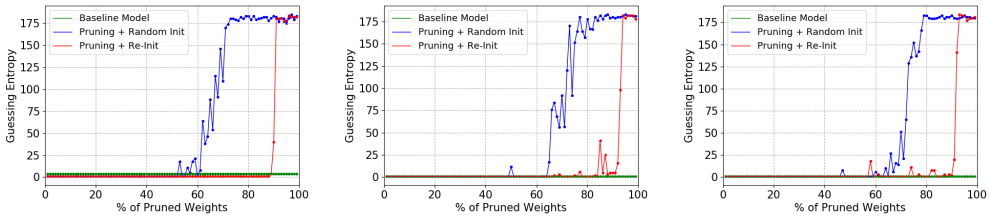
(a) 30 000 profiling traces. **(b)** 40 000 profiling traces. **(c)** 50 000 profiling traces.

Figure 2: ASCAD Fixed Key, MLP4



(a) 30 000 profiling traces. **(b)** 40 000 profiling traces. **(c)** 50 000 profiling traces.

Figure 3: ASCAD Fixed Key, MLP6

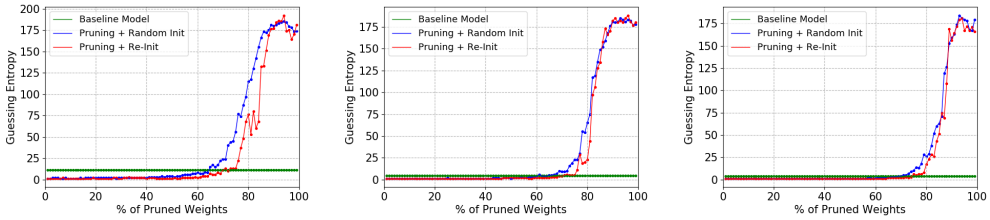


(a) 30 000 profiling traces. **(b)** 40 000 profiling traces. **(c)** 50 000 profiling traces.

Figure 4: ASCAD Fixed Key, MLP8

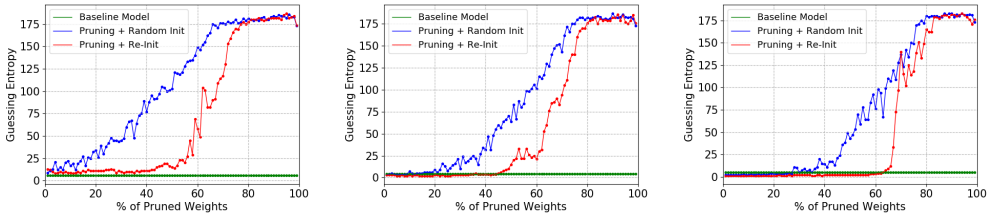
if the number of profiling traces is low (30 000). This happens as the large (baseline) model has too much capacity, and it starts to overfit.

Next, we give results for three different CNN architectures. Figures 5 and 6 indicate that for 30 000 training traces, as the dataset is small, the baseline model generally performs well but shows signs of overfitting. Then, pruning up to 60% of weights improves the performance regardless of the weight initialization procedure. Next, increasing the number of traces shows improved behavior for the baseline model as now we require larger architecture to reach the full capacity. Still, carefully selected sub-networks are sufficient to break the target, even when pruning 80% of weights. Going to a more complex architecture (CNN4), we see that the baseline model performs better than the CNN3 case as it has a stronger feature selection. At the same time, pruning enables similar performance where the larger the training set, the smaller the differences between weight initialization procedures (re-initialization or random). In Figure 7, we consider the most complex CNN architecture. Interestingly, for 40 000 and 50 000 traces, we observe an even better performance of pruned networks, which means that the baseline network overfits and using more profiling traces enables stronger attack performance for sub-networks.



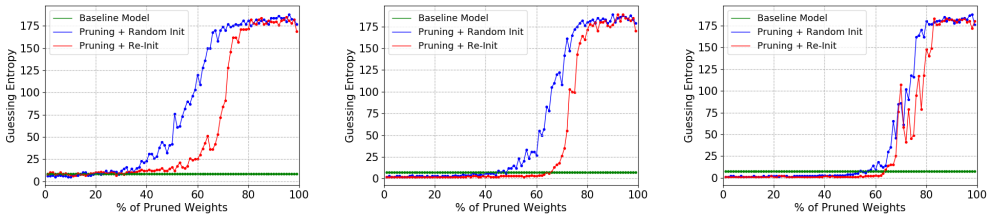
(a) 30 000 profiling traces. **(b)** 40 000 profiling traces. **(c)** 50 000 profiling traces.

Figure 5: ASCAD Fixed Key, CNN3



(a) 30 000 profiling traces. **(b)** 40 000 profiling traces. **(c)** 50 000 profiling traces.

Figure 6: ASCAD Fixed Key, CNN4



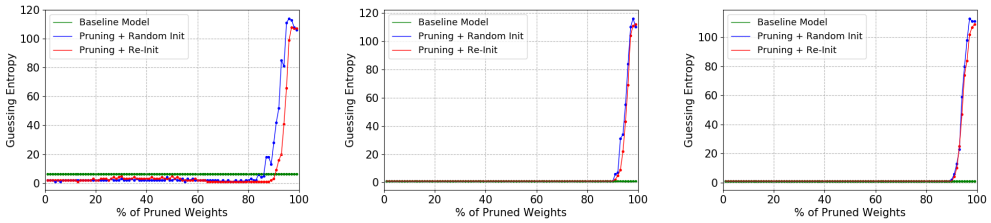
(a) 30 000 profiling traces. **(b)** 40 000 profiling traces. **(c)** 50 000 profiling traces.

Figure 7: ASCAD Fixed Key, CNN4-2

5.3 ASCAD with Random Keys

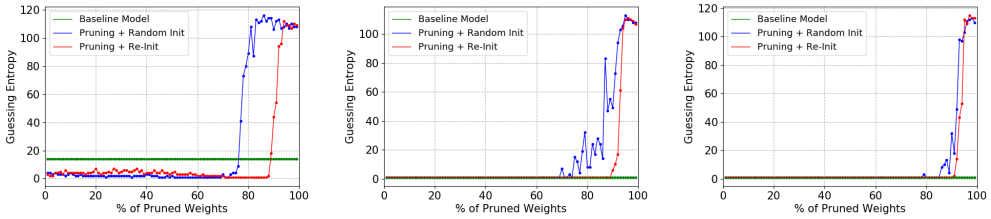
In this section, we provide results for the ASCAD random keys dataset, as introduced in Section 2.4. Again, we apply our procedure for a different number of profiling traces (60 000, 100 000, and 200 000) on the six different baseline models (MLP4, MLP6, MLP8, CNN3, CNN4, and CNN4-2).

Figure 8 shows results for different number of profiling traces on the MLP4 baseline model. This MLP model has four dense layers, and it can be considered a small model, which is efficient for the ASCAD dataset for a large number of profiling traces (above 100 000), as indicated by the baseline model guessing entropy results. However, if the number of profiling traces is reduced (60 000), the guessing entropy result for the baseline model trained for 300 epochs shows worse results due to overfitting. On the other hand, applying the lottery ticket hypothesis on this MLP4 baseline model shows good results even when the number of profiling traces is reduced. Comparing the blue and red lines, we can also verify that re-initializing the pruned model to the initial baseline model weights shows superior results for higher sparsity levels (% of pruned weights).



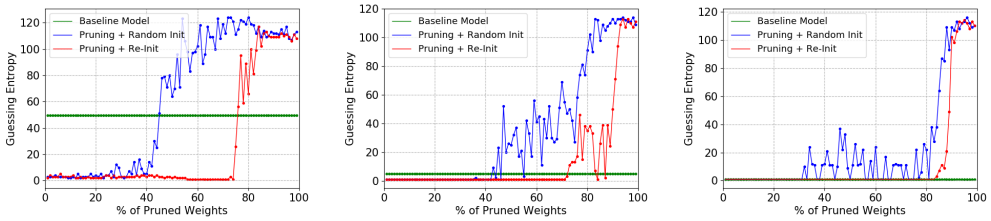
(a) 60 000 profiling traces. **(b)** 100 000 profiling traces. **(c)** 200 000 profiling traces.

Figure 8: ASCAD Random Keys, MLP4



(a) 60 000 profiling traces. **(b)** 100 000 profiling traces. **(c)** 200 000 profiling traces.

Figure 9: ASCAD Random Keys, MLP6

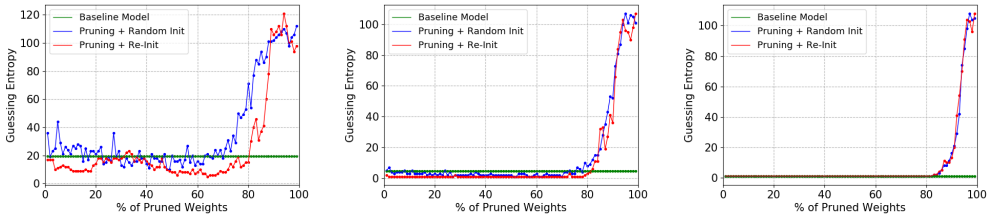


(a) 60 000 profiling traces. **(b)** 100 000 profiling traces. **(c)** 200 000 profiling traces.

Figure 10: ASCAD Random Keys, MLP8

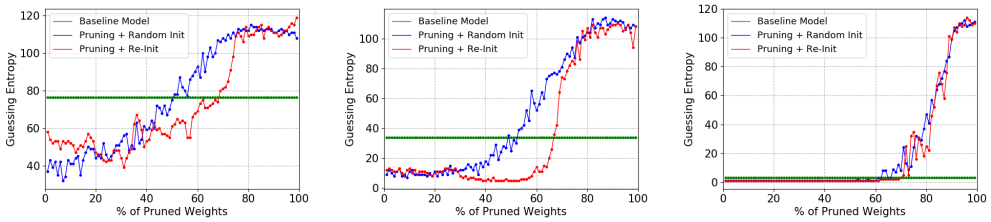
The observations are confirmed in Figures 9 and 10 for machine learning models with more capacity (MLP6 and MLP8). Indeed, profiling sets that are too small cause overfitting for the baseline model, which can be easily resolved following the pruning method. Notice that the random initialization always works worse than re-initialization, and it also gives more irregular behavior due to the randomness in the process. This indeed confirms that the lottery ticket hypothesis is valid in the profiling SCA context. As shown in Figure 9, pruning approximately 90% of the weights from the baseline model provides successful guessing entropy results when weights are set to the initial baseline weights.

By comparing Figures 9 and 10, we can observe that larger baseline models tend to provide less successful results when the lottery ticket hypothesis procedure is applied. Larger baseline models overfit training data more easily, and, as a consequence, the pruning process is applied to a model that might overfit. The solution for this problem is to consider early stopping for the baseline model training. This way, pruning would be applied to the baseline model weights when they reach the best training epoch. To confirm our hypothesis, we can consider Figure 10c. The baseline model (MLP8) is trained on 200 000 profiling traces for 300 epochs and does not overfit, as seen in the baseline model's guessing



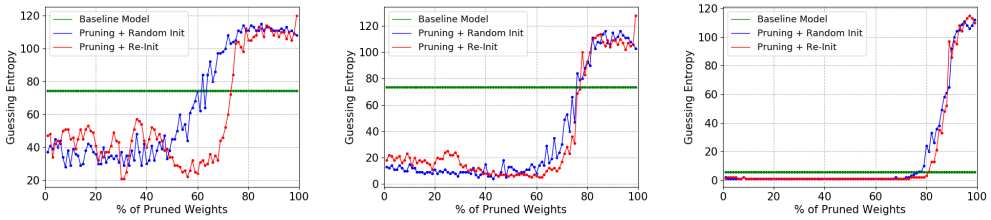
(a) 60 000 profiling traces. **(b)** 100 000 profiling traces. **(c)** 200 000 profiling traces.

Figure 11: ASCAD Random Keys, CNN3



(a) 60 000 profiling traces. **(b)** 100 000 profiling traces. **(c)** 200 000 profiling traces.

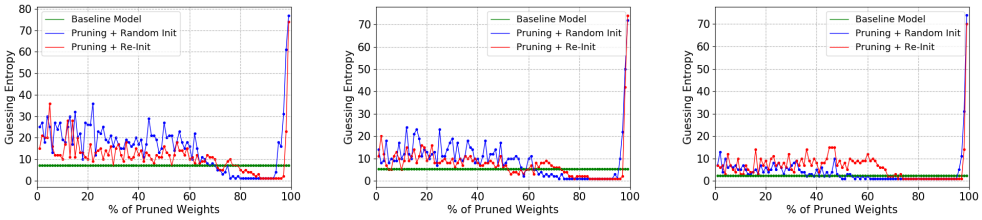
Figure 12: ASCAD Random Keys, CNN4



(a) 60 000 profiling traces. **(b)** 100 000 profiling traces. **(c)** 200 000 profiling traces.

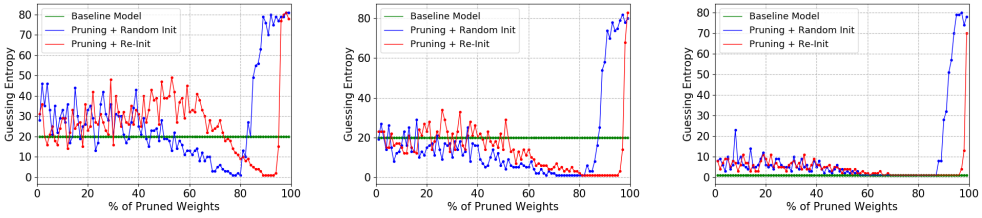
Figure 13: ASCAD Random Keys, CNN4-2

entropy. In this case, the pruned model performance (when weights are re-initialized to the initial baseline model weights) is as good as for smaller baseline models trained on the same number of profiling traces (see, e.g., Figure 9c). The CNN architectures selected for this analysis show better GE results for the baseline model when more profiling traces are used. However, when less profiling traces are used, as is the case of results provided in Figures 11b, 12b, and 13b, the baseline guessing entropy is not reaching 1 (only for CNN3 it reaches minimum GE of 4 when trained on 200 000 traces). Adding more profiling traces helps, but the number of profiling traces should align with the model complexity. The evaluated CNN models worked well for the ASCAD fixed key dataset, as shown in the last section. However, these models (especially CNN4 and CNN4-2) appear to be less appropriate for the ASCAD random keys dataset. In such cases, pruning plays an important role to (partially) overcome this. After pruning, it is possible to reach very low GE values (under 5) for a specific percentage of pruned weights. In particular, results show pruning plus weight re-initialization is better than pruning plus random initialization. For all cases, notice that we can prune up to around 50% of weights and still reach good performance even though we use (relatively) simple CNN architectures.



(a) 20 000 profiling traces. **(b)** 30 000 profiling traces. **(c)** 40 000 profiling traces.

Figure 14: CHES CTF, MLP4



(a) 20 000 profiling traces. **(b)** 30 000 profiling traces. **(c)** 40 000 profiling traces.

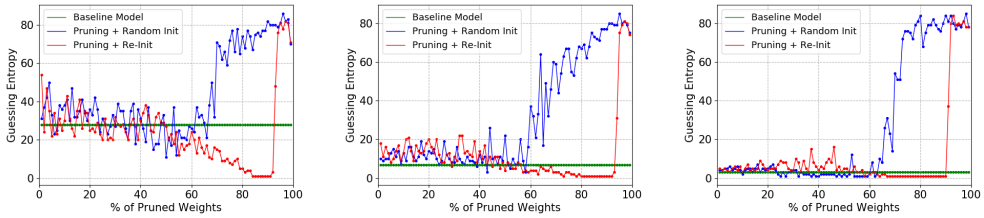
Figure 15: CHES CTF, MLP6

5.4 CHES CTF 2018

For the CHES CTF 2018 dataset, we repeated the experiments on the same neural network architectures defined in Tables 1 and 2. In this case, we observed much better results for MLPs compared to the CNN performances. Thus, for this dataset and MLPs, we confirmed the practical advantage of considering the lottery ticket hypothesis procedure in profiling SCA.

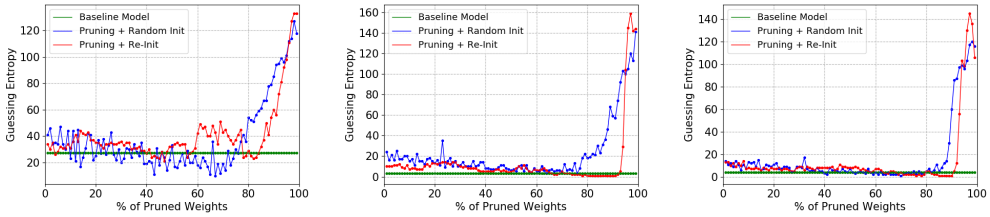
Figure 14 shows the final guessing entropy for different sparsity levels on three different number of profiling traces: 20 000, 30 000, and 40 000. As indicated by the green line in Figures 14a, 14b, and 14c, the baseline guessing entropy cannot reach GE of 1 for MLP4 trained on 300 epochs. Adding more profiling traces helps, but still GE stays above 1. When the network is pruned, we can immediately see how GE improves, especially for sparsity levels around 80% to 95%. In particular, the re-initialization of pruned weights with the initial baseline weights shows better results compared to pruning with the random initialization. Figures 15 and 16 confirm our observations as we see that more profiling traces is required for good attack performance for the baseline model, especially as the architecture becomes more complex. On the other hand, we can prune up to 95% of weights if we re-initialize the pruned model and still reach superior attack performance.

Results for CNNs on CHES CTF 2018 dataset are acceptable (i.e., converged to GE close to 1) for the CNN3 architecture only, as shown in Figure 17. There, we see the benefit of adding more profiling traces as the baseline model overfits. Still, some sub-networks are providing better attack performance. For CNN4 and CNN4-2 (Figures 18 and 19), the baseline model provides poor performances when trained on 300 epochs. We postulate this happens as the baseline model has a significantly larger capacity than needed, so it either overfits or underfits, becoming similar to random guessing. In other words, CNN4 and CNN4-2 on smaller profiling sets (lower than 30 000 traces) show no generalization for the baseline model, indicating that these two models are not compatible with the target dataset. Still, even with those models, we can observe how the lottery ticket hypothesis reduces the



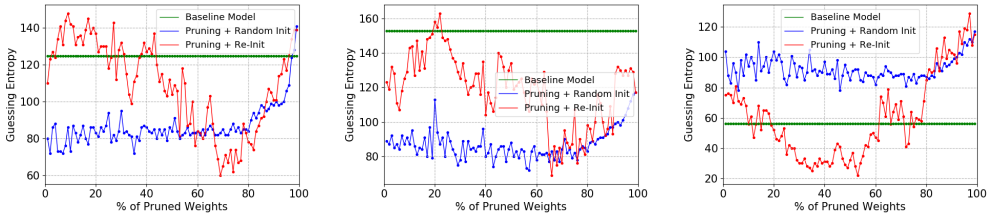
(a) 20 000 profiling traces. **(b)** 30 000 profiling traces. **(c)** 40 000 profiling traces.

Figure 16: CHES CTF, MLP8



(a) 20 000 profiling traces. **(b)** 30 000 profiling traces. **(c)** 40 000 profiling traces.

Figure 17: CHES CTF, CNN3



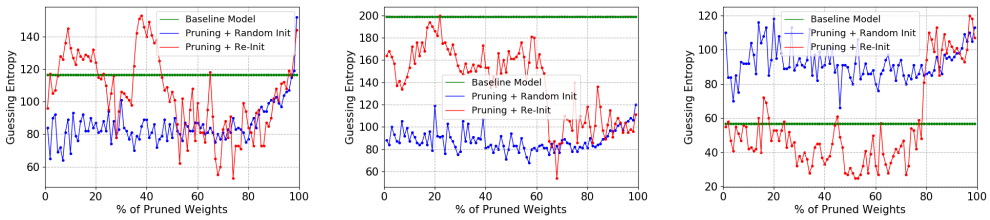
(a) 20 000 profiling traces. **(b)** 30 000 profiling traces. **(c)** 40 000 profiling traces.

Figure 18: CHES CTF, CNN4

guessing entropy for specific sparsity level ranges. Observing Figures 18 and 19, we can immediately verify than for sparsity levels around 70%, the pruned model re-initialized with the initial baseline weights can reach significantly lower guessing entropy (GE < 70) after training for 50 epochs. Increasing the number of attack traces (we consider only 2 000 attack traces) could lead to successful key recovery, which is particularly interesting if we consider that a baseline model provided performance close to random guessing. When the number of profiling traces is increased to 40 000 traces (Figures 18c and 19c), the baseline model shows slightly better results and pruning plus re-initialization still improves the attack performance. In this case, we can verify that pruning plus random initialization might not be a good procedure, as guessing entropy results are inferior to the baseline model results.

5.5 Summary of Results

In this section, we provide a summary of the results observed in the previous sections. In Table 3, we give the percentages of the pruned weights that result in the successful key



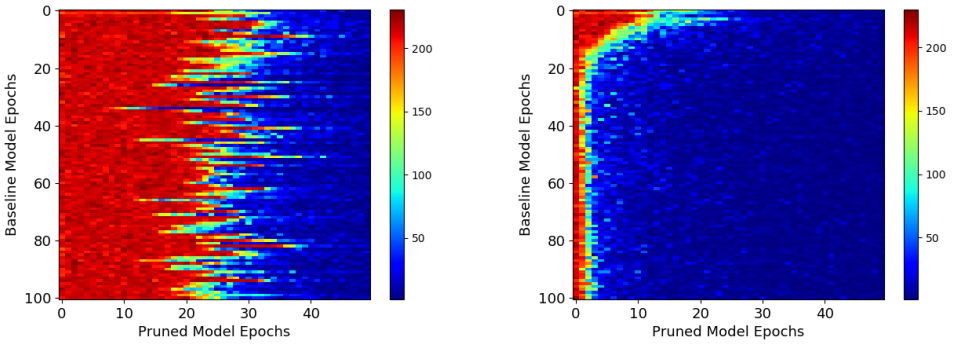
(a) 20 000 profiling traces. **(b)** 30 000 profiling traces. **(c)** 40 000 profiling traces.

Figure 19: CHES CTF, CNN4-2

recovery. Notice that only when the baseline model is not successful, it happens that the pruned model is not successful. Even in those cases, pruned models can show much better performance. In general, we can prune most of the weights in a network and still obtain good attack performance. The more complex architectures commonly used somewhat lower pruning rates as the relations that such networks find tend to be more complex. Using more traces also usually increases the maximally allowed pruning percentage.

Table 3: % of pruned weights that leads to successful byte key recovery with the LTH procedure.

Dataset / Traces	**30 000**	**40 000**	**50 000**
ASCAD Fixed Key - MLP4	1% - 88%	1% - 88%	1% - 87%
ASCAD Fixed Key - MLP6	1% - 86%	1% - 86%	1% - 87%
ASCAD Fixed Key - MLP8	1% - 88%	1% - 85%	1% - 87%
ASCAD Fixed Key - CNN3	1% - 60%	1% - 68%	1% - 73%
ASCAD Fixed Key - CNN4	-	1% - 44%	1% - 63%
ASCAD Fixed Key - CNN4-2	-	1% - 60%	1% - 63%
Dataset / Traces	**60 000**	**100 000**	**200 000**
ASCAD Random Keys - MLP4	64% - 88%	1% - 90%	1% - 90%
ASCAD Random Keys - MLP6	71% - 87%	1% - 89%	1% - 90%
ASCAD Random Keys - MLP8	56% - 75%	1% - 71%	1% - 83%
ASCAD Random Keys - CNN3	-	1% - 78%	1% - 80%
ASCAD Random Keys - CNN4	-	-	1% - 71%
ASCAD Random Keys - CNN4-2	-	-	1% - 73%
Dataset / Traces	**20 000**	**30 000**	**40 000**
CHES CTF 2018 - MLP4	88% - 96%	84% - 96%	74% - 97%
CHES CTF 2018 - MLP6	89% - 93%	81% - 95%	66% - 96%
CHES CTF 2018 - MLP8	84% - 92%	80% - 91%	68% - 90%
CHES CTF 2018 - CNN3	-	1% - 78%	76% - 88%
CHES CTF 2018 - CNN4	-	-	-
CHES CTF 2018 - CNN4-2	-	-	-



(a) Pruning with random initialization. **(b)** Pruning with re-initialization (LTH).

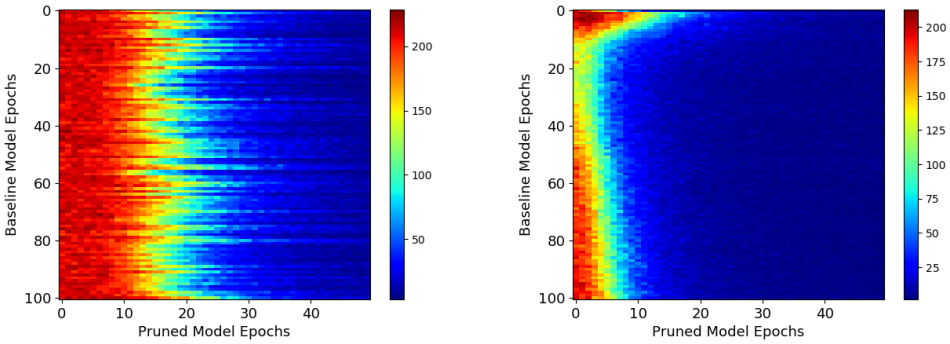
Figure 20: Guessing entropy results for the ASCAD fixed key dataset (30 000 profiling traces) with the MLP4 baseline model. The sparsity level for pruning is set to **80%** and the attack set has 2 000 traces.

5.6 Analyzing the Number of Epochs for the Baseline and Pruned Models

Results provided in previous sections demonstrate that an over-parameterized baseline model can be drastically reduced in size and still provide optimal (or even superior) results if following the lottery ticket hypothesis procedure. We showed for three different datasets and six neural network architectures that there is a range of sparsity levels that we can apply to obtain successful key recovery results. However, in those examples, we fixed the number of baseline training epochs to 300 and the number of pruned model epochs to 50 in all scenarios. This section provides experiments demonstrating that the number of baseline trained epochs does not need to be very large to provide good results. More precisely, this means that there are already sub-networks that are the winning tickets after a short training. Moreover, we also show that the pruned and re-initialized model also requires a very small number of epochs to reach low guessing entropy. This section considers several scenarios, and we note that the rest of the scenarios are aligned.

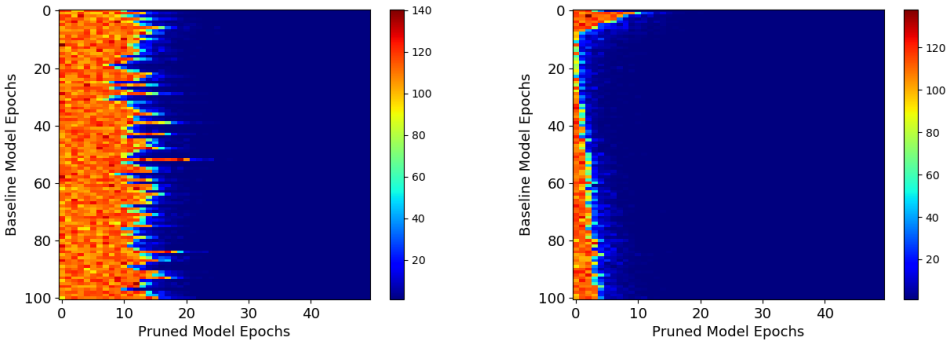
As illustrated in Figure 20a, for the ASCAD fixed key dataset with the MLP4 model, the pruned model when weights are randomly initialized requires at least 35 epochs to reach successful key recovery, *regardless of the number of trained epochs for the baseline model.* This analysis emphasizes that training the baseline model before pruning is irrelevant concerning profiling side-channel analysis performance *if the weights are randomly initialized after pruning.* On the other hand, as shown in Figure 20b, a successful key recovery attack can be implemented by following the LTH procedure with 80% of sparsity and by training the baseline model for not more than 20 epochs, and by re-initializing and training the pruned model for no more than five epochs. More importantly, when the baseline model is not even trained (zero epochs), the pruned and re-initialized model requires no more than 20 epochs to reach the successful key recovery. This does not directly mean that the weights initialization was correctly done. What we can conclude is that MLP4 is over-parameterized and pruning it to 80% (by removing the smallest initial weights) and training the pruned model is successful for the considered dataset.

In Figure 21, we depict results for the CNN3 model and ASCAD fixed key. The results confirm the observations from the MLP4 model, where we see that with the random initialization, the network requires around 20 epochs to break the target, regardless of the training length for the baseline model. The only difference from the MLP4 case is that the results are somewhat better due to a better-performing architecture. CNN3



(a) Pruning with random initialization. **(b)** Pruning with re-initialization (LTH).

Figure 21: Guessing entropy results for the ASCAD fixed key dataset (30 000 profiling traces) with the CNN3 baseline model. The sparsity level for pruning is set to **60%** and the attack set has 2 000 traces.



(a) Pruning with random initialization. **(b)** Pruning with re-initialization (LTH).

Figure 22: Guessing entropy results for the ASCAD Random keys dataset (100 000 profiling traces) with the MLP4 baseline model. The sparsity level for pruning is set to **80%** and the attack set has 2 000 traces.

requires somewhat longer training after re-initialization, but in general, around ten epochs is enough. There is an interesting spot around 30 epochs for the baseline model that performs the best after pruning. This indicates that the model overfits significantly and influences the behavior of winning tickets also. Finally, we depict that the results for MLP4 model and ASCAD random keys in Figure 22. The behavior is similar to previous cases where we see that after a short training of the baseline model (20 epochs), we can train the pruned and re-initialized model for five epochs only and break the target. If we use random initialization for the same performance level, we need to train approximately three times longer.

5.7 Success Rate Results

Up to now, we considered guessing entropy as a metric of success. As the results show that the pruned (and re-initialized) models can reach better attack performance than the baseline model, it is clear that the probabilities for each trace x_i in the matrix P change to indicate more fit key candidates. Now, we briefly concentrate on the question of whether

the LTH procedure changes the whole probability matrix (so, even less likely guesses), or most of the changes happen for the most likely key. Consequently, we give results for the success rate (SR) metric in Table 4 and 5 for MLP and CNN, respectively. Note that these tables represent specific pruning rates, but the results for other rates are aligned with the GE results. In green, we depict all cases where the pruned network (LTH) performs better than the baseline network (BS). In Table 4, observe how using more training traces increases the chances that the LTH results will be better. This is especially clear for 200 000 training traces from the ASCAD random keys dataset, where for all nine settings, LTH shows better results. Even more important, we can see that the improvements due to LTH can be very significant (e.g., CNN3, where we see improvement from 80.6% to 99.3%). At the same time, for simpler networks, we observe fewer improvements due to LTH. Table 5 presents CNN results and confirms our observations. More training traces and more complex networks increase the benefit of using LTH. In general, we see that pruning can significantly improve the SR results. For instance, for ASCAD random keys and MLP6, we increase SR from 23.9% for the baseline architecture to 92.9% for the architecture with 80% of pruned weights. Thus, we reach significantly better attack performance for a significantly smaller neural network. We also note the ASCAD fixed key dataset and CNN3 for 30 000 training traces where we observe that the baseline model reached 27.4%, which happened as all the measurements were classified as belonging to the HW 4. Interestingly, we see that pruning can help to avoid the imbalanced dataset problem [PHJ+18]. Finally, we observe that for all tested cases but one (CHES CTF 2018 and MLP), there is a number of training traces and pruning rate that improves the results over the baseline case, which re-iterates the potential of pruning in SCA.

Table 4: Success rates (in %) after the processing of 2 000 traces in the attack phase and for 80% of weight pruning in the LTH procedure.

Dataset / Traces	**30 000**	**40 000**	**50 000**
	BS | LTH	**BS | LTH**	**BS | LTH**
ASCAD Fixed Key - MLP4	85.3 | 75.9	66.5 | 80.3	86.4 | 89.0
ASCAD Fixed Key - MLP6	41.4 | 88.1	89.5 | 92.5	85.4 | 90.9
ASCAD Fixed Key - MLP8	48.9 | 92.7	83.8 | 95.0	86.7 | 95.7
Dataset / Traces	**60 000**	**100 000**	**200 000**
	BS | LTH	**BS | LTH**	**BS | LTH**
ASCAD Random Keys - MLP4	45.5 | 91.9	88.7 | 99.6	99.4 | 100.0
ASCAD Random Keys - MLP6	23.9 | 92.2	97.2 | 100.0	97.7 | 100.0
ASCAD Random Keys - MLP8	3.9 | 11.9	47.9 | 69.5	96.7 | 100.0
Dataset / Traces	**20 000**	**30 000**	**40 000**
	BS | LTH	**BS | LTH**	**BS | LTH**
CHES CTF 2018 - MLP4	32.8 | 45.2	42.9 | 72.1	61.4 | 86.7
CHES CTF 2018 - MLP6	11.3 | 27.8	12.6 | 68.4	95.8 | 93.3
CHES CTF 2018 - MLP8	6.3 | 44.0	34.3 | 75.5	57.6 | 97.6

5.8 Attacking the Full Key with One Pruned Model

The previous section demonstrated that pruning and re-initializing the pruned model with initial baseline weights is very efficient when focusing on one AES key byte. The only drawback in the procedure presented above is identifying the optimal sparsity level percentage to obtain a stable and successful (*winning ticket*) model. This section demonstrates that such a procedure is only needed for one single key byte when attacking the full AES key.

Table 5: Success rates (in %) after the processing of 2 000 traces in the attack phase and for 60% of weight pruning in the LTH procedure.

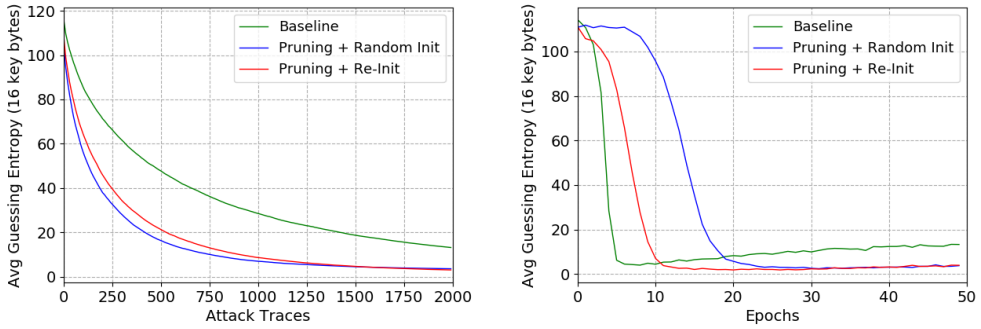
Dataset / Traces	**30 000**	**40 000**	**50 000**
	BS | LTH	**BS | LTH**	**BS | LTH**
ASCAD Fixed Key - CNN3	27.4 | 72.2	47.1 | 72.0	52.0 | 80.7
ASCAD Fixed Key - CNN4	44.7 | 7.65	57.2 | 34.15	48.11 | 59.105
ASCAD Fixed Key - CNN4-2	35.4 | 14.4	39.0 | 61.0	36.11 | 64.11
Dataset / Traces	**60 000**	**100 000**	**200 000**
	BS | LTH	**BS | LTH**	**BS | LTH**
ASCAD Random Keys - CNN3	15.7 | 36.3	48.0 | 92.7	80.6 | 99.3
ASCAD Random Keys - CNN4	5.73 | 8.8	25.47 | 28.97	63.05 | 82.45
ASCAD Random Keys - CNN4-2	4.4 | 16.9	1.6 | 44.9	43.7 | 94.3
Dataset / Traces	**20 000**	**30 000**	**40 000**
	BS | LTH	**BS | LTH**	**BS | LTH**
CHES CTF 2018 - CNN3	6.7 | 2.5	53.56 | 38.22	50.0 | 37.22
CHES CTF 2018 - CNN4	0.05 | 0.325	0.088 | 0.059	1.433 | 2.0
CHES CTF 2018 - CNN4-2	0.05 | 0.0	0.0 | 0.056	1.625 | 2.75

Once a pruned model is created, and the correct amount of pruned weights is defined, this pruned model can be directly applied to the remaining key bytes.

In this section, we only consider CHES CTF 2018 dataset. After identifying that for MLP4 and MLP6 (we consider those two models as they reached the best results as discussed in the previous section), a sparsity level of 80% shows good results when applying the LTH procedure, we train the baseline models for 20 epochs on the first key byte (key byte 0) and re-use the pruned model with 80% of sparsity on the remaining 15 key bytes. Results for MLP4 model are shown in Figure 23.

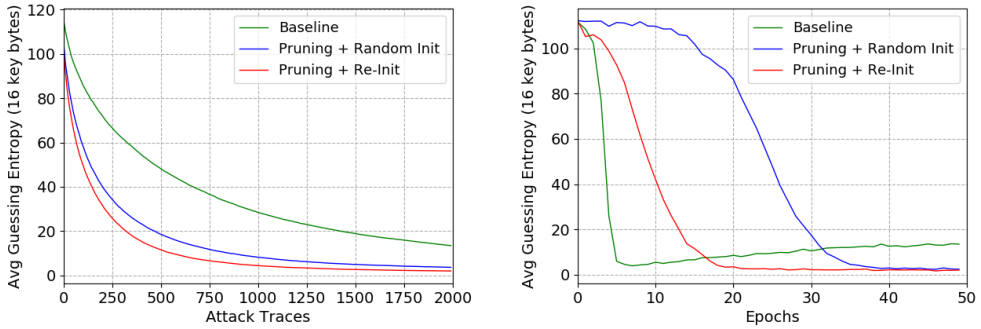
As shown in Figure 23a, the average guessing entropy for the 16 AES key bytes is lower after the processing of 50 epochs. Analyzing the guessing entropy during training, as illustrated in Figure 23b indicates that the baseline model converges faster than pruned models. However, because the baseline model is over-parameterized, it starts to show overfitting after approximately 20 epochs, which results in a guessing entropy increase for more trained epochs. Using the LTH procedure with the same baseline model for all key bytes, we achieve a stable guessing entropy even processing 50 epochs. This immediately tells us that LTH brings an advantage compared to retraining the same baseline model for different key bytes, and no expensive early stopping mechanism (such as calculating guessing entropy for each epoch) needs to be adopted during training. Finally, we consider the setup where we prune 90% of weights (Figure 24. We can see even better results considering the number of attack traces to reach as low as possible GE. More precisely, we observe that the pruning and re-initialization procedure works the best. Similar to the previous case, we observe that the baseline model has the best performance in the beginning epochs, but it soon starts to overfit. On the other hand, pruned models do not overfit even considering the full training process, and they reach better average guessing entropy.

Figures 25 and 26 show results for CHES CTF 2018 dataset and CNN3. In these particular cases, we can verify that using the same pruned model according to LTH procedure on all 16 key bytes over-performs the baseline model trained separately for each key byte.



(a) GE *vs.* attack traces. **(b)** GE *vs.* epochs.

Figure 23: Average guessing entropy for the full AES key for the CHES CTF 2018 dataset for MLP4. Baseline model is trained for 20 epochs on the first key byte. Pruning with **80%** of sparsity level is applied and the same pruned model (following the LTH procedure) is trained for 50 epochs on all key bytes



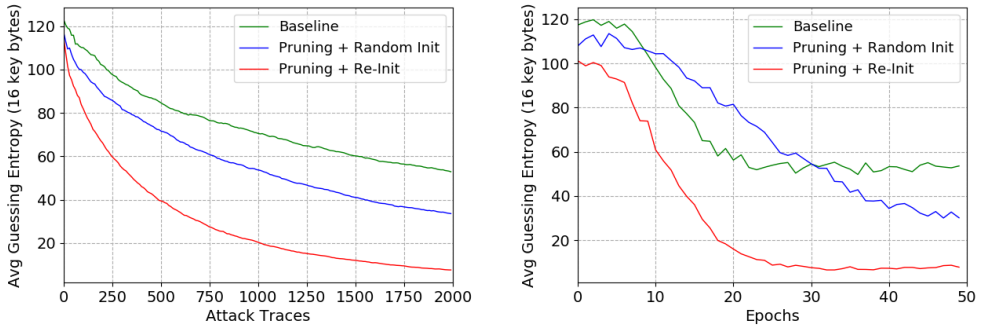
(a) GE *vs.* attack traces **(b)** GE *vs.* epochs.

Figure 24: Average guessing entropy for the full AES key for the CHES CTF 2018 dataset for MLP4. Baseline model is trained for 20 epochs on the first key byte. Pruning with **90%** of sparsity level is applied and the same pruned model (following the LTH procedure) is trained for 50 epochs on all key bytes

5.9 General Observations

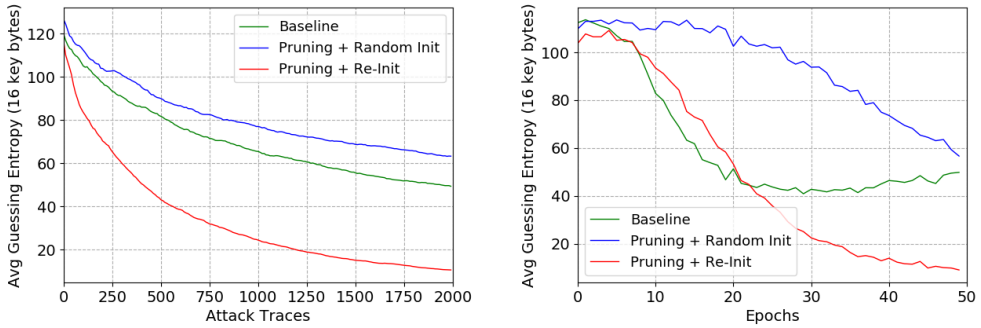
Based on the conducted experiments, we can recognize several general observations:

- If the baseline model behaves like random guessing for a limited set of attack traces, pruning might still improve performance.
- If the baseline works well and does not overfit, then pruning maintains the performance but requires smaller networks.
- If there are not enough profiling traces for the model capacity, we will overfit, and pruning can help avoid that.
- More profiling traces improves pruning results, but it also reduces differences between weight initialization techniques.
- Pruning and weight re-initialization work the best.
- Pruning can improve the attack results as indicated by various SCA performance metrics.



(a) GE *vs.* attack traces **(b)** GE *vs.* epochs.

Figure 25: Average guessing entropy for the full AES key for the CHES CTF 2018 dataset for CNN3. Baseline model is trained for 20 epochs on the first key byte. Pruning with **70%** of sparsity level is applied and the same pruned model (following the LTH procedure) is trained for 50 epochs on all key bytes



(a) GE *vs.* attack traces **(b)** GE *vs.* epochs.

Figure 26: Average guessing entropy for the full AES key for the CHES CTF 2018 dataset for CNN3. Baseline model is trained for 20 epochs on the first key byte. Pruning with **80%** of sparsity level is applied and the same pruned model (following the LTH procedure) is trained for 50 epochs on all key bytes

- Pruning represents a strong option when considering a neural network model trained for one key byte to be applied for other key bytes.

6 Conclusions and Future Work

This paper discussed how pruning could improve the attack performance for deep learning-based side-channel analysis. We considered the recently proposed Lottery Ticket Hypothesis that assumes there are small sub-networks in the original network that perform on the same level as the original network. To the best of our knowledge, both of those concepts were never before investigated in profiling SCA. Our experimental investigation confirms this hypothesis for profiling SCA, which allows us to prune up to 90% of weights and still reach good attack performance. Thus, we manage to reach the same attack performance for significantly smaller networks (easier to tune and faster to train). What is more, we show

how pruning helps in cases when a large network overfits or has issues due to imbalanced data. Indeed, in those cases, pruning, besides resulting in smaller architectures, enables improved attack performance.

As future work, we plan to consider more sophisticated pruning techniques and the ID leakage model. Finally, as discussed, pruning allows smaller neural networks and good performance but does not provide insights into neural networks' explainability. It could be interesting to consider various feature visualization techniques to evaluate the important features before and after the pruning.

References

[APSQ06] C. Archambeau, E. Peeters, F. X. Standaert, and J. J. Quisquater. Template attacks in principal subspaces. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 1–14, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BOFG20] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning?, 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.

[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.

[CK14] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, pages 253–270, Cham, 2014. Springer International Publishing.

[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.

[FC18] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018.

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

[HPGM16] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Side-channel analysis of lightweight ciphers: Does lightweight equal easy? In Gerhard P. Hancke and Konstantinos Markantonakis, editors, *Radio Frequency Identification and IoT Security - 12th International Workshop, RFIDSec 2016, Hong Kong, China, November 30 - December 2, 2016, Revised Selected Papers*, volume 10155 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2016.

[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.

[Jan89] Steven A. Janowsky. Pruning versus clipping in neural networks. *Phys. Rev. A*, 39:6600–6603, Jun 1989.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, 1996.

[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.

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 20–33. Springer, 2015.

[MHM14] Zdenek Martinasek, Jan Hajny, and Lukas Malina. Optimization of power analysis using neural network. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, pages 94–107, Cham, 2014. Springer International Publishing.

[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.

[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.

[PHJ+17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 4095–4102, 2017.

[PHJ+18] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. 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.

[PHJB19] S. Picek, A. Heuser, A. Jovic, and L. Batina. A systematic evaluation of profiling through focused feature selection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2802–2815, 2019.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[RWPP21] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2021/071, 2021. https://eprint.iacr.org/2021/071.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 30–46, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[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.

[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.

[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.

[YZLC12] Shuguo Yang, Yongbin Zhou, Jiye Liu, and Danyang Chen. Back propagation neural network based leakage characterization for practical security analysis of cryptographic implementations. In Howon Kim, editor, *Information Security and Cryptology - ICISC 2011*, pages 169–185, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[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.