Profiled Power Analysis Attacks Using Convolutional Neural Networks with Domain Knowledge

Benjamin Hettwer[1(✉)], Stefan Gehrer[1], and Tim Güneysu[2]

[1] Robert Bosch GmbH, Corporate Sector Research, Stuttgart, Germany
{benjamin.hettwer,stefan.gehrer}@de.bosch.com

[2] Horst Görtz Institute for IT-Security, Ruhr University Bochum, Bochum, Germany
tim.gueneysu@rub.de

Abstract. Evaluation of cryptographic implementations against profiled side-channel attacks plays a fundamental role in security testing nowadays. Recently, deep neural networks and especially Convolutional Neural Networks have been introduced as a new tool for that purpose. Although having several practical advantages over common Gaussian templates such as intrinsic feature extraction, the deep-learning-based profiling techniques proposed in literature still require a suitable leakage model for the implementation under test. Since this is a crucial task, we are introducing domain knowledge to exploit the full power of approximating very complex functions with neural networks. By doing so, we are able to attack the secret key directly without any assumption about the leakage behavior. Our experiments confirmed that our method is much more efficient than state-of-the-art profiling approaches when targeting an unprotected hardware and a protected software implementation of the AES.

Keywords: Side-channel attacks · Deep learning
Convolutional Neural Networks

1 Introduction

Power-based Side-Channel Attacks (SCAs) are a well-known and powerful class of threats for security enabled devices, for example in context of the Internet of Things. They exploit information leakages gained from the power consumption or electromagnetic emanations of a device to extract secret information such as cryptographic keys, even though the employed algorithms are mathematically sound. This is caused by the correlation of power consumption and processed data. Since the advent of power-based SCAs by Kocher et al. in 1999 [14], numerous papers have been published on this topic. Most of them fit into one of the following categories:

Non-profiled SCAs techniques aim to recover the secret key by performing statistical calculations on power measurements of the device under attack regarding a hypothesis of the device's leakage. Typical examples are Differential Power Analysis [15], Correlation Power Analysis [5], and Mutual Information Analysis [9].

Profiled SCAs assume a stronger adversary who is in possession of a profiling device. It is an open copy of the attacked device which the adversary can manipulate to characterize the leakages very precisely in a first step. Once this has been done, the built model can be used to attack the actual target device in the key extraction phase. Template Attacks (TAs) [7], Stochastic attacks [26] and machine-learning-based attacks [3,12,16] are common approaches in this area.

In the same manner, researchers and industry developed methods to counteract SCAs. *Masking*, for instance, aims for randomizing intermediate values that are internally processed by the cryptographic device in order to break the connection between the secret (respectively some intermediate value that depends on the secret) and its power footprint [19]. The concept of *Hiding* countermeasures are different from masking in a sense that their goal is to change the power characteristics directly. This can be achieved, for example, by making every operation consume the same amount of energy. However, it has been shown that protected implementations can be broken as well, whereby particularly profiled SCAs are a reasonable choice [10,23].

There is a recent line of work that deals with the application of *Deep Learning (DL)* techniques for profiled side-channel analysis. A common factor that motivates the usage of DL models in general is that they intrinsically incorporate a feature extraction mechanism. That is, unlike most standard Machine Learning (ML) classifiers, DL models can learn from the raw input data set as they are able to identify the most informative data themselves without human engineering. Within the SCA community, Maghrebi et al. [18] showed in a series of experiments that DL can outperform TAs and standard ML techniques like support vector machines when targeting hard- and software implementations of AES. One year later, Cagli et al. [6] investigated *Convolutional Neural Networks (CNNs)* combined with data augmentation to defeat cryptographic implementations which are protected with different jitter-based countermeasures. Again, better results were reported for the DL network compared to TAs with manual trace realignment. Summarizing the insights of the two studies, it becomes evident that DL techniques and in particular CNNs gives two major advantages that make them interesting for profiled SCAs:

– They are able to automatically extract the areas in the side-channel traces which contains the most information. When using standard SCA techniques, the selection of the so-called Points of Interests (POIs) is often done manually as preprocessing step ahead of the actual attack. This is not only tedious, but also error prone as proper POI selection has shown to have a significant impact on the attack efficiency [35].

– CNNs are invariant to small input modifications such as noise (also artificially generated). Furthermore, they integrate time samples from the complete traces efficiently (meaning they require fewer parameters which needs to be optimized during training) for their decision. This property enables them to perform a higher-order SCA and defeat masking countermeasures.

All studies on deep-learning-based SCAs assumed that the attacker has some implicit knowledge about the leakage behavior of the attacked implementation. However, the choice of an adequate leakage model (i.e., an approximation of the physical signal that is generated by the device when computing some sensitive intermediate value) is usually crucial for the success of SCAs [8] and heavily depends on how much information about the target architecture is available to the adversary. Since this is may be difficult to determine upfront, we present a black-box approach for evaluating cryptographic implementations without a leakage model by using CNNs with Domain Knowledge (DK) neurons.

1.1 Contributions

The contributions of this paper are twofold:

1. We introduce a novel CNN architecture for profiled SCAs which allows to encode domain specific information. By doing so, it is possible to feed the plaintext or ciphertext as an additional source of information into the network (apart from the power measurements). The CNN with DK is dedicated to autonomously learn the leakage of the device with regard to the secret key.
2. We perform practical experiments with an unprotected hardware and a protected software implementation of AES. The results confirm that our method reduces the search space for breaking the secret key in the attack phase by at least three orders of magnitude for the hardware implementation, and more than ten orders in case of the protected software implementation.

The rest of this paper is structured as follows: In Sect. 2, background on profiled SCAs, Neural Networks (NNs) and DL is provided. Section 3 introduces CNNs and our architectural extension with domain neurons. In Sect. 4, the results of our experiments are presented and discussed. The last Section summarizes the paper and gives insights on possible future work.

2 Preliminaries

This section serves as entry point to profiled SCAs, NNs and DL. We refer the reader to [19] for a more profound introduction into power-based SCAs, and [11] for a comprehensive summary on NN and DL.

2.1 Profiled Side-Channel Analysis

Profiled SCAs are considered as the most powerful type of SCAs and are divided in two phases. In a first step, the adversary takes advantage of a profiling device on which he can fully control input and secret key parameters of the cryptographic algorithm. He uses that to acquire a set of N_P profiling side-channel traces $X \in \mathbb{R}^D$, where D denotes the number of sample points in the measurements. Let $V = \mathrm{g}(t, k)$ be a random variable representing the result of an intermediate operation of the target cipher which depends partly on public information t (plaintext or ciphertext chunk) and secret key $k \in \mathcal{K}$, where $\mathcal{K}$ is the set of possible key values. V is assumed to have an influence on the deterministic part of the side-channel measurements. The ultimate goal of the attacker during the profiling phase is then to estimate the probability:

$$\Pr[X|V = v] \tag{1}$$

for every possible value $v \in V$ from the profiling base $\{X_i, v_i\}_{i=1,\ldots,N_P}$. In TAs for example, the Probability Density Function (PDF) of (1) is assumed to be multivariate Gaussian and can described by the parameter pairs (μ_v, Σ_v), depicting the mean values and covariance matrices for the corresponding values of v [7].

During the attack phase, the adversary generates a new set of N_A attack traces from the actual target device (which is structurally identical to the profiling device) whereby the secret key k is fixed and unknown. In order to retrieve it, estimations for all possible key candidates $k^* \in \mathcal{K}$ are made and combined following the *maximum likelihood* strategy such that:

$$k = \underset{k^* \in \mathcal{K}}{\operatorname{argmax}} \prod_{i=1}^{N_A} \Pr[V = v_i | X_i] \tag{2}$$

where the probabilities on the right are retrieved with the help of the built profile and public information t which is also available for the attack traces. In order to avoid numerical instabilities, it is common to process the logarithms of the likelihoods.

Although the Gaussian model assumption is often fairly realistic in practice [19], arbitrary functions of the side-channel leakage cannot be captured with templates. In settings where the PDF of the leakage is not known upfront, ML-based profiling methods are more promising. Another issue that comes with TAs is the necessity to find a small number of POIs in the high-dimensional side-channel measurements. This is due to size restriction of the covariance matrices Σ_v, which are $(N_S \times N_S)$ large when N_S is the number of POIs. In order to discover the POIs, dimensionality reduction techniques such as (PCA) can be employed. PCA captures the data with the largest variance and thus helps to reduce the amount of noise in the traces. That is why PCA is a heavily used technique in side-channel analysis, not only for TAs, but also in settings where the profiling is done with ML techniques [16,25,33]. However, in general one can say that ML-based attacks are more suitable when it is difficult to restrict the number of POIs effectively [17].

2.2 Neural Networks and Deep Learning

NNs were partly inspired by biological learning systems (e.g. the human brain) and date back at least into the 1960s. They are nowadays the privileged choice for supervised classification tasks. For these, the learning system is fed with training examples from a data set consisting of input data vectors (=features) and associated outcome measurement (=label) and the goal is to find a suitable relationship in order to map new inputs to the correct label. Note that in the context of profiled SCA, the first set is equal to the profiling base N_P and the second one corresponds to the attack set N_A.

NNs are composed of densely interconnected units called neurons, which take a number of real-valued inputs and produce a single real-value output [20]. The simplest type of a NN is the *perceptron*. As illustrated in Fig. 1, it receives a vector of input features $X = (x_1, \ldots, x_D)$ and performs a linear combination with the weight values $w_1, \ldots, w_D$ of its input connections and a bias value w_0. The result is passed through an Activation (ACT) function f, e.g., the Rectified Linear Unit (ReLU) [21] in order to calculate the output value $\tilde{y}$. For learning the perceptron, the weights are adjusted according to the training data set.

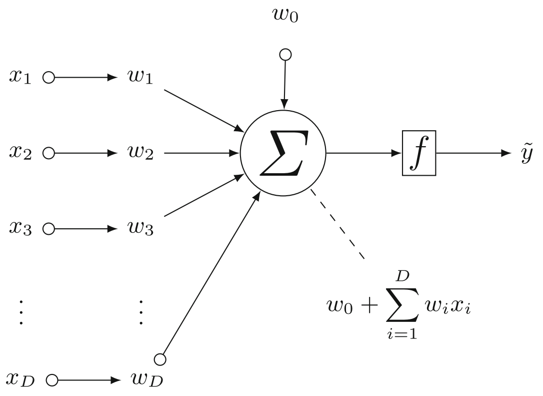

Fig. 1. Perceptron

Single-layer perceptrons are only able to represent functions whose underlying data set is linearly separable such as the Boolean AND function. To overcome this limitation and represent more complex mappings, many perceptrons can be stacked together to form a whole network which are generally referred to as *Multi-Layer Perceptrons (MLPs)*. An MLP consists of three types of units, typically arranged in layers as shown in Fig. 2. The input layer is just a representation of the raw input features. All neurons of the input layer are connected to each neuron of the following hidden layer. The number of hidden layers in an MLP and the number of units per hidden varies, depending on the required model capacity to fit the training data. In general, too many units in the hidden layer may lead to overfitting, while underestimating the number of neurons has an negative effect on the classification performance of the MLP [11]. The units in the output layer, finally, directly correspond to the predictions of the classification problem to solve.

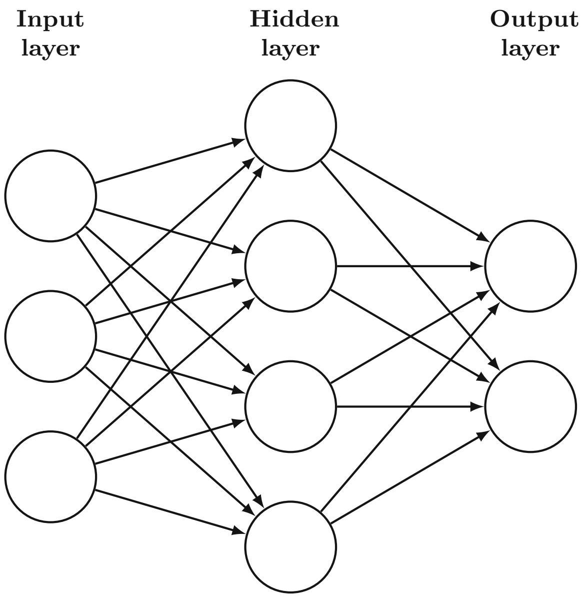

Fig. 2. Example of a simple MLP with 3 input units, 4 hidden units, 2 output units (bias units omitted).

Training the MLP is an iterative, multi-step process by which the weights of the network are optimized to minimize a loss function, which depicts the difference between the expected output label and the prediction result. The learning rate hyperparameter determines how fast the weights of the network are driven towards the optimal solution. In practice, optimizer algorithms such as Stochastic Gradient Descent (SGD) or ADAM are employed for that purpose [11].

In recent years there has been a growing interest in NN models with multiple hidden layers stacked upon each other, which are commonly specified under the term *deep learning*. It is a particular powerful type of ML techniques that are able to represent the learning task as nested hierarchy of concepts, where more abstract concept representations are built from simpler ones. The usage of deep NNs is motivated by the fact that they have and outperformed classical ML approaches in solving central problems in artificial intelligence such as speech recognition and image classification. These tasks usually deal with high-dimensional data which makes it exponentially more difficult to learn a classifier that generalizes well on unseen examples, a challenge that is also known as the curse of dimensionality [11]. Since this applies in exactly the same manner for the SCA domain as discussed before, deep NNs and especially CNNs seem like a promising choice as tool for profiled SCAs.

3 Convolutional Neural Networks

In this section, we first describe the primary building blocks of CNNs until we present our architectural extension with DK neurons.

3.1 Core Constructions

CNNs tackle the problem of large input data dimensions by including task-specific mechanisms into their architecture that allow to reduce the number of parameters of the model, while keeping or even increasing the accuracy of the network [22]. CNNs are primarily used in the field of pattern recognition within images, however they can also be used to process 1-D time-series data (as it is the case for side-channel traces). Additional to the Fully-Connected (FC) layers used in classical MLPs, CNNs include two other types of layers, namely Convolutional (CONV) layers and Pooling (POOL) layers:

CONV layers determine the output of neurons which are connected to small spatial regions of the input by calculating the scalar product with a set of so-called kernels or filters as illustrated in Fig. 3. The movement policy of the filters can be modified by the strides parameter. The weight parameters of the kernels are learned to activate when they detect a specific feature or pattern at a certain position in the input. In order to perceive enough information, different filters are used yielding several outputs which increases the depth of the network. CONV layers are to some extent, shift, scale, and distortion invariant. This property has shown to be very useful against de-synchronized side-channel traces [6].

POOL layers perform downsampling of their given input in order to reduce the number of parameters and the computational complexity of the network, by considering the max (=max-pooling) or average (=average-pooling) of a certain spatial extent as the output. They are important for getting low-dimensional abstract feature representations and compressing the information that is extracted in the CONV layers.

Apart from the CONV and POOL layers which are specific for CNNs, there are two additional techniques that can be found in common architectures of CNNs. These are dropout and batch normalization:

Dropout is a regularization technique that helps the network to increase generalization and reduce the phenomena of overfitting [28]. The key idea of dropout is to randomly drop units (along with their connections) from the NN during training. The probability to drop a unit can be controlled by the probability coefficient $\mathrm{P}_{Drop} \in [0, 1)$. Because of that, dropout can be seen as an ensemble method that combines a exponential number of different "thinned" NN architectures efficiently during training. At test time, a single network with downscaled weights is used for predictions.

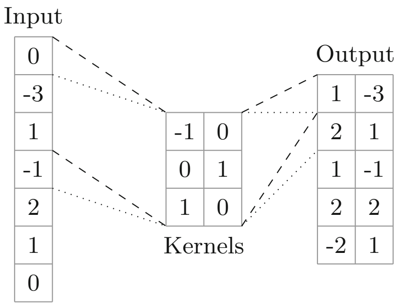

Fig. 3. Example of a 1-D convolution operation with 2 kernels of length 3 and stride of 1. The output is formed by applying the kernel to each part of the input (as with a sliding window).

Batch Normalization was introduced by Ioffe et al. [13] to establish a stable distribution of activation values throughout the whole layered structure of a network. A stable distribution makes the network more robust to parameter tuning since the input of one layer depends on the output of the previous layer. Therefore, normalization is incorporated into the network architecture by applying it to each mini-batch of training examples. This eventually allows the usage of higher learning rates.

3.2 Principal Architecture

Following the Input (IN) layer, CNN architectures typically consist of repetitive blocks of CONV and POOL layers. The basic concepts of sparse, local connectivity, weight sharing and subsampling enable the network to extract more abstract representations of given inputs, until spatial output dimensions are small enough to be connected to subsequent FC layers. Additionally, the use of non-linear ACT functions such as ReLU or sigmoid right after each CONV and FC layer enables the network to learn more complex functions. In a classification setting (as for example the one we describe in Sect. 2.1), the neurons of the last layer in the network output probabilities over discrete classes. These are calculated by means of the *Softmax (SOFT)* function.

To sum up, the architecture of a typical CNN consists of two major parts. A feature extractor and a feature combinator. The feature extractor consists of alternating CONV and POOL layers. It yields low dimensional representations of the input (in our case a side-channel trace), giving crucial information to the subsequent layers for solving the classification task. FC layers act as feature combinators and connect information to the desired output. A current CNN can therefore characterized by the following construction:

$$\mathrm{IN} \circ [\mathrm{CONV} \circ \mathrm{ACT} \circ \mathrm{POOL}]^{n_1} \circ [\mathrm{FC} \circ \mathrm{ACT}]^{n_2} \circ \mathrm{FC} \circ \mathrm{SOFT}$$

where n_1 and n_2 denote the number of feature extractor blocks, respectively the number of FC layers used.

3.3 CNNs with Domain Knowledge Neurons

In our approach, we study the effect of additional DK neurons in the CNN architecture for profiled SCAs. Their addition is motivated by the fact that merging domain specific information with extracted features of the CONV layers enables the network to go into different statistics at decision level [32]. In that sense, we propose a multimodal CNN with late information fusion strategy where additional public data is fed to the network in order to increase the efficiency of the attack. Since we have targeted the first byte of the AES key in our experiments as described in Sect. 4, we decided to use the corresponding plaintext byte as input for the DK neurons. However, it is also conceivable to exploit other related data that is available to the attacker (e.g. the ciphertext or information about the internal structure of the attacked implementation). Introducing a chunk of the plaintext into the network as second input brings two major advantages that motivates our approach:

- We do not have to stick to a certain leakage model. Instead of assuming that the attacked implementation leaks information regarding a certain operation for which we do the profiling (for example the output of the AES S-Box, respectively the hamming weight of the S-Box), we directly use the secret key k as a label. By doing so, we give the network the ability to autonomously learn the most meaningful representation of the leakage which is needed to classify the used secret key.
- The second advantage is a direct consequence of our generic leakage model. In the attack phase, we do not make a key guess on all possible candidates and combine the estimations on it via maximum-likelihood as seen in Sect. 2.1. Instead, the network gives us a direct key estimation in form of the probabilities:
$$\Pr[k|X, t] \tag{3}$$
for every attack trace X and associated plaintext t. This leads to a faster convergence of the key rank as we will see later in the experiments section.

Our developed CNN architecture is illustrated in Fig. 4. A detailed description is given in Table 2 in the Appendix. In summary, the feature extractor part of the model consists of three CONV layers and two POOL layers. All CONV layers use the same kernel size of eight, but the number of filters is increased from eight, to 16, up to 32. Dimensionality reduction of the features is reached by max-pooling across two data points after the first two CONV layers. After flattening the spatial depth of the feature extractors into a single dimension, it is concatenated with the input of the DK neurons. Since we merge one byte of the plaintext one-hot encoded into network, the DK layer contains 256 neurons (one for each possible value). One-hot encoding represent the plaintext byte as vector of 256 binary variables where only the correct value is set to one. The information from the feature extractor part and the DK neurons is combined by a following FC layer. The Output (OUT) layer consists of 256 neurons as we make a classification for one key byte. In order to avoid overfitting, four dropout layers are included into the network architecture with a consistent dropout rate $P_{Drop} = 0.2$. Furthermore, batch normalization is employed after all CONV and FC layers. Throughout the network, ReLU is used as an activation function.

We stress that our CNN design is not the product of some architectural optimization technique. It was rather obtained by following best practices for developing deep NN architectures [27], and examination of related work [6,18].

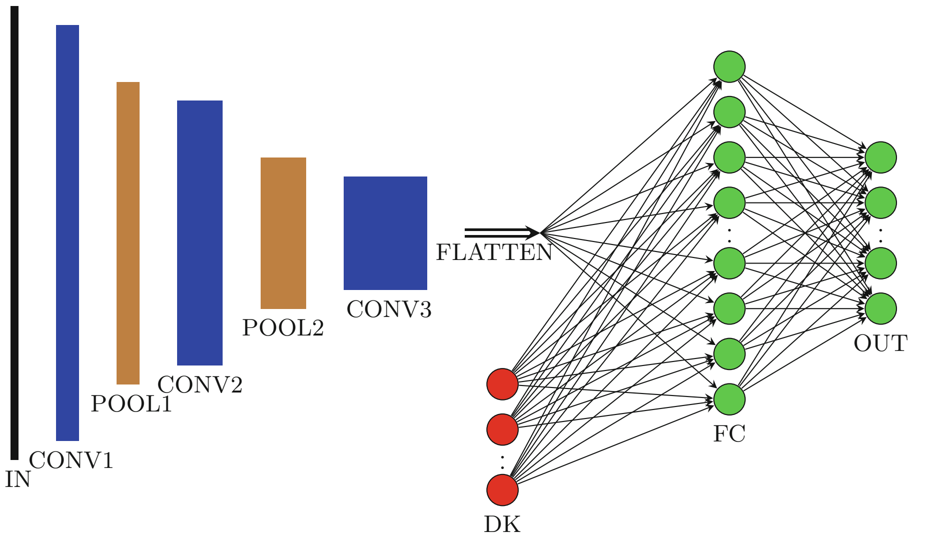

Fig. 4. Simplified visualization of CNN with domain input neurons.

4 Experiments

In the following section, we present our experimental results. After explaining the general attack setup, we compare our CNN with DK approach against four different profiling attacks from literature regarding attack efficiency when targeting an unprotected hardware and a protected software implementation of AES.

4.1 Baseline

For our experiments, we have implemented three deep NNs which were proposed in literature as baseline for our CNN with DK neurons. An overview of the evaluated models and associated target operations compared to our approach is given in Table 1. The numbers in the first column represent the number of layers with trainable weights. We chose these networks as reference, since the proposing authors applied them to break the same or very similar targets (unprotected hardware and protected software implementations of AES). Additionally, we performed a classical TA for both attacked data sets. In all experiments we aim to recover the first byte of the AES key. However, we stress that if one is able to retrieve one byte of the key successfully, the remaining bytes can be attacked likewise.

Table 1. Overview of implemented attacks

Type	Profiling target (Label)	Source
2-layer MLP	$V = \text{S-box}(t[0], k[0])$	[18]
3-layer CNN	$V = \text{S-box}(t[0], k[0])$	[18]
5-layer CNN	$V = \text{S-box}(t[0], k[0])$	[24]
TA	$V = \text{S-box}(t[0], k[0])$	[7]
5-layer CNN w/ DK	$k[0]$	This paper

Not all baseline models are described in the same level of detail in the according papers. For example, the activation functions for the MLP in [18] are not given. Therefore, we performed a so-called *grid search* for estimating the missing hyperparameters that are needed to rebuild and train the networks. It works as follows: First, an interval or set of possible values has to be selected for each parameter that should be optimized. Grid search is then just a simple strategy that tries all possible parameter combinations over the predefined ranges. We list the optimized parameters and associated search intervals for each of the evaluated models in Table 3 in the Appendix due space restrictions. The applied methodology, however, has been the same for all attacks and is described in the following section.

4.2 Methodology

Data Sets. For the conducted experiments, we have considered data sets of $N_P = 200\,000$ profiling traces with random plaintext and keys. The number of attack traces N_A with random plaintext but fixed secret key k varies for the attacks. We have used four sets each having 10 000 attack traces for the unprotected hardware implementation, and two sets each containing 10 000 attack traces for the protected software implementation. All attack sets were acquired with a different key in order to prevent any bias in the results due to overfitting to a certain key value.

Evaluation Metric. A single, well-know metric from the SCA domain has been used to evaluate the performance of the attacks: The Key Guessing Entropy (KGE) or key rank function. It is a technique which quantifies the difficulty to retrieve the correct value of the key regarding the required number of attack traces [29]. In principle, the KGE is calculated by summing up the log-likelihoods obtained in Eq. (2) over all key guesses $k^* \in \mathcal{K}$ (respectively the log-likelihoods of (3)) and do a ranking of the result. This ranking is updated after each attack trace. The KGE has the advantage of taking the full information on the probability distributions that are given in (2) or (3) into account, whereas the standard accuracy metric from the DL domain only considers the label with the highest confidence.

Attack Scenario. In order to have a fair comparison, we have applied the following strategy for all attacks:

1. We have done a grid search hyperparameter optimization for all models according to the values in Table 3, meaning we trained each model for all possible parameter combinations with the full profiling set N_P and validated its performance with 2000 attack traces from N_A. The model variants that yield the lowest KGE were considered for further analysis.
2. Next, we performed 20 (10 for the software implementation) independent attacks using the models obtained in the first step and calculated the mean KGE, whereas each attack was conducted with an independent set of 2000 traces from N_A.

The experiments last around three weeks on a single Nvidia GTX 1080 Ti graphics card. All implemented attacks are based on the Keras [1] and scikit-learn [2] frameworks.

4.3 Results for Unprotected Hardware Implementation

Our first series of experiments have been based on the public data set of the *DPA Contest v2* [30]. These side-channel traces were acquired from an unprotected AES design running on an FPGA platform. The used AES module performs one round per clock cycle. Each trace contains 3253 sample points and covers a complete encryption operation.

As a preprocessing step, we transformed all traces to have zero mean and unit variance (sometimes referred to as data standardization). We also investigated the effect of normalizing the traces into a range of [0, 1] or having no preprocessing at all, but got the best results with standardization.

We have not reduced the dimension of the traces, except for TA. TAs requires the attacker to determine a small number of sample points which contain the most discriminative information. Otherwise they can become computationally intractable as laid out in Sect. 2.1. We employed a PCA for that purpose with the number of components to keep as hyperparameter. The exact parameter configurations for the networks can be found in the Appendix.

Figure 5 shows the mean key ranks according to the number of traces for each implemented attack. From that, we can make the following observations:

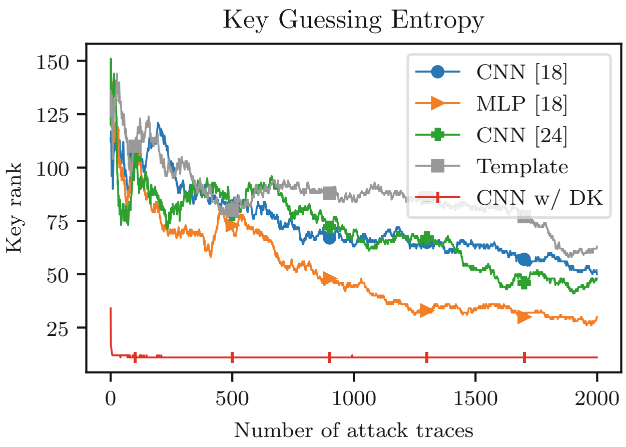

Fig. 5. Mean ranks when targeting the first key byte of an unprotected AES hardware implementation.

- Our CNN with domain neurons outperforms all other approaches, meaning it has the lowest mean KGE after 2000 attack traces (8 vs. 30 when comparing it with the MLP-based attack).
- None of the attacks reaches a stable key rank of zero. We indeed examined that a larger number of traces is necessary to recover the key with a success rate of 100% (approximately 5000 with the MLP). This is not completely in line with the good results obtained in [18] and could be a direct consequence of our hyperparameter optimization process and the assumptions we had to make when reimplementing the networks. Additionally, targeting an S-box that is not followed by a register may not be the optimal choice in a hardware setting since the leakage of combinatorial logic is typically lower than register leakage.
- Even though our developed CNN is not able to converge to a key rank of zero (also not with more than 2000 attack traces), it stabilizes under the top ten with less than ten attempts. The CNN with DK converges so much faster due to higher probabilities for the top ranked key estimations. For example, the top five probabilities obtained after the SOFT layer account for approximately 95% of the complete probability distribution, an effect that is not visible for the baseline models with such an intensity. This makes our attack especially interesting for settings where the number of attack traces is restricted to a few of tens or even less.

4.4 Results for Protected Software Implementation

The second platform we have targeted is a software-based AES implementation equipped with two SCA countermeasures:

- A first-order secure masking scheme called *Rotating Sbox Masking (RSM)*, and
- Shuffling.

In RSM, the mask values are fixed to carefully chosen values, but rotated for every execution. It is therefore considered a lightweight masking scheme. The employed shuffling algorithm in the design randomly changes the order of execution of the S-boxes. The implementation originates from DPA Contest v4.2 [4].

Since the traces which were provided within the DPA Contest v4.2 were generated with a single fixed key and we are required to have random keys for the profiling, we self-acquired the data sets N_P and N_A on a ChipWhisperer-Lite board for the second series of experiments. The board was running with a clock frequency of 7.37 MHz. Each trace is composed of 10 000 sample points representing approximately the first one and a half rounds of an encryption operation. As an example, we have plotted three measurements in Fig. 6.

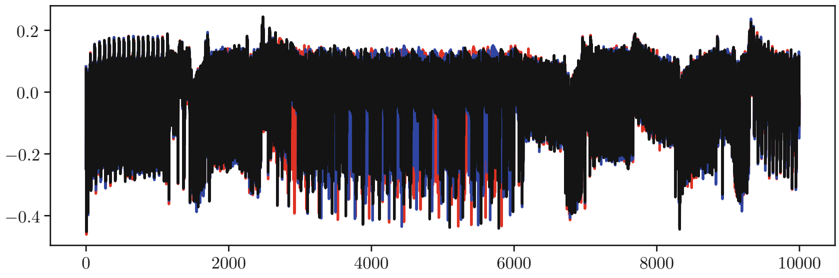

Fig. 6. Three example traces of the protected software AES implementation. The shuffling of the S-boxes is clearly visible in the range between the time samples 2500 and 6500.

We have applied the same data standardization preprocessing as for the hardware target also to the traces of the software implementation. Additionally, a separate hyperparameter optimization for the software data set has been conducted. The results of the attacks are illustrated in Fig. 7. One can notice that:

- The CNN with DK performs very well on the software implementation. Indeed, it takes roughly 20 traces to get to key rank zero for the first time and stabilizes after roughly 600 attack traces. This demonstrates that our developed method is also able to defeat cryptographic implementations which are secured with several countermeasures.

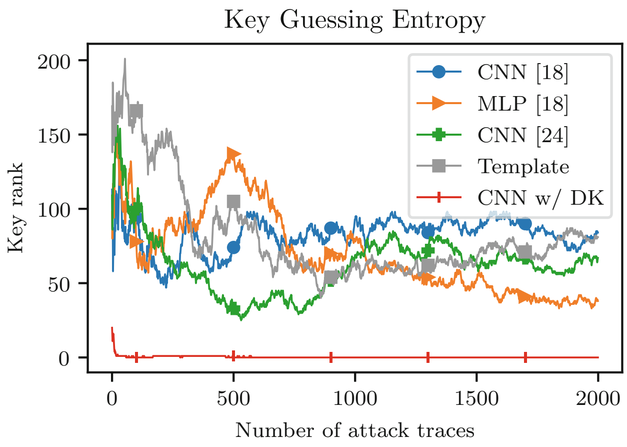

Fig. 7. Mean ranks when targeting the first key byte of an protected AES software implementation.

– Compared to the results for the unprotected hardware implementation, all approaches except ours perform worse for the software implementation. This indicates that the employed masking and shuffling countermeasures effectively decrease the leakage of the targeted S-box. We have also tested the effect of using a whole attack data set with fixed key (10 000 traces) but were not able to reach a constant KGE of zero with the TA and the networks from related work.

Examining the Effect of Domain Knowledge Neurons. In order to assess the effect of DK on the attack success, we have trained our developed CNN architecture from scratch under the exact same conditions but without the additional input of the plaintext. Afterwards, we have computed the mean KGE for the CNN without domain neurons in the same manner as we have done for the other implemented attacks. The results are shown in Fig. 8.

From the plots, it can be concluded that the information provided by the domain neurons in fact improve the performance of the network. Both CNNs (with and without DK) are able to reach a key rank below five after less than 20 traces, which indicates that our generic architecture by itself leads to a significant performance boost. However, only the network which is equipped with the domain input converges to zero. This demonstrates our assumption that additional knowledge, that is present to the attacker anyway, can be used more efficiently as it was done in state-of-the-art approaches. Maghrebi et al., e.g., used the plaintext only to generate the labels for training/profiling and was therefore not given to the networks in the attack phase to classify unseen traces [18].

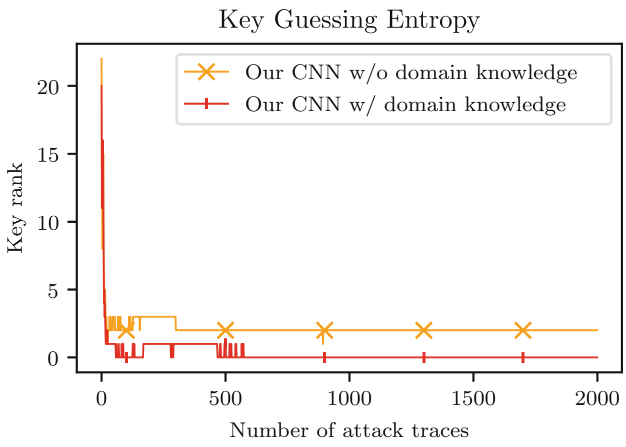

Fig. 8. Mean KGE when targeting a protected AES software implementation with, and without DK.

Hybrid learning systems (as our approach can be considered) have shown remarkably result on several real-world problems [31,34]. Our developed strategy adapts the idea to the SCA domain. The experiments presented in this section clearly illustrate that combining different types of information (e.g. side-channel traces and the plaintext) into one DL classifier can boost the performance of profiled SCAs up to several orders of magnitude (compared to state-of-the art attack methods). Furthermore, we stress that our approach may also be beneficial to evaluate other kinds of cryptographic implementations apart from AES as we make not us of any internal algorithmic structures or implementation details.

5 Conclusion

In this paper we have introduced CNNs with DK neurons as a tool for profiled SCAs. The addition of domain neurons supplies the network with extra information such as the plaintext. We showed that this feature gives a great practical advantage compared to state-of-the-art profiling attacks [18,24], which require to manually choose a certain operation of the attacked implementation for which the profiling is done. Instead, we have demonstrated by experiments with two different data sets that our proposed CNN with DK effectively manages to autonomously capture the function with the highest leakage for breaking the secret key directly. Our method can thus be seen as a novel and generic tool to assess the side-channel resistance of cryptographic implementations in a real black-box manner (i.e. assuming an attacker with no knowledge about internal implementation structures).

Future work might explore other kinds of DK than the plaintext. For instance, one could try to attack the AES subkey in the last round and feed the corresponding ciphertext into the network. An alternative path of future work could be to study the effect of domain neurons in combination with other deep NN architectures (e.g. Recurrent Neural Networks).

Acknowledgment. The authors would like to thank the reviewers for their comments. This work is supported in parts by the German Federal Ministry of Education and Research (BMBF) under grant agreement number 16KIS0606K (SecRec).

A Network Parameters

Table 2. Network configuration of CNN with domain neurons.

Layer type	Hyperparameters
Trace input	-
Convolution 1D	Filters = 8, filter length = 8
Max-pooling	Pool length = 2
Dropout	$P_{Drop} = 0.2$
Convolution 1D	Filters = 16, filter length = 8
Batch normalization	-
Max-pooling	Pool length = 2
Dropout	$P_{Drop} = 0.2$
Convolution 1D	Filters = 32, filter length = 8
Batch normalization	-
Dropout	$P_{Drop} = 0.2$
Flatten	-
Domain input	Neurons = 256
Concatenate	-
Fully-connected	Neurons = 400
Batch normalization	-
Dropout	$P_{Drop} = 0.2$
Output	Neurons = 256

Table 3. Results of grid search hyperparameter optimization for all implemented attacks. Chosen values for the hardware attack are marked in bold letters, chosen values for the software attack are marked by underlining.

Type	Hyperparameter
2-layer MLP	Batch size: [**50**, 100]
	Epochs: [100, **200**]
	Optimizers: [SGD, **RMSprop**, Adam, Nadam]
	Activation: [ReLU, **sigmoid**, tanh]
	Learn rate: [0.001, 0.0001, **0.00001**]
3-layer CNN	Batch size: [**50**, 100]
	Epochs: [**100**, 200]
	Optimizers: [SGD, **RMSprop**, Adam, Nadam]
	Learn rate: [0.001, **0.0001**, 0.00001]
	P_{Drop}: [**0.2**, 0.3, 0.4, 0.5]
5-layer CNN	Batch size: [**50**, 100]
	Epochs: [**100**, 200]
	Optimizers: [SGD, RMSprop, Adam, **Nadam**]
	Learn rate: [**0.001**, 0.0001, 0.00001]
TA	PCA components: [1, ..., **5**, 6, ..., 100]
5-layer CNN w/DK	Batch size: [**50**, 100]
	Epochs: [**100**, 200]
	Optimizers: [SGD, **RMSprop**, Adam, Nadam]
	Activation: [**ReLU**, sigmoid, tanh]
	Learn rate: [**0.001**, 0.0001, 0.00001]
	P_{Drop}: [0.2, **0.3**, 0.4, 0.5]

References

1. Keras Documentation. https://keras.io/
2. Scikit-learn: machine learning in Python. http://scikit-learn.org/stable/
3. Bartkewitz, T., Lemke-Rust, K.: Efficient template attacks based on probabilistic multi-class support vector machines. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 263–276. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37288-9_18
4. Bhasin, S., Bruneau, N., Danger, J.-L., Guilley, S., Najm, Z.: Analysis and improvements of the DPA contest v4 implementation. In: Chakraborty, R.S., Matyas, V., Schaumont, P. (eds.) SPACE 2014. LNCS, vol. 8804, pp. 201–218. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12060-7_14
5. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2
6. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: Fischer, W., Homma, N. (eds.)

CHES 2017. LNCS, vol. 10529, pp. 45–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_3

7. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
8. Doget, J., Prouff, E., Rivain, M., Standaert, F.X.: Univariate side channel attacks and leakage modeling. J. Cryptogr. Eng. **1**(2), 123 (2011). https://doi.org/10.1007/s13389-011-0010-2
9. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_27
10. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust, HOST, pp. 106–111, May 2015. https://doi.org/10.1109/HST.2015.7140247
11. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
12. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. J. Cryptogr. Eng. **1**(4), 293 (2011). https://doi.org/10.1007/s13389-011-0023-x
13. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167 (2015). http://arxiv.org/abs/1502.03167
14. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
15. Kocher, P., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptogr. Eng. **1**(1), 5–27 (2011). https://doi.org/10.1007/s13389-011-0006-y
16. Lerman, L., Bontempi, G., Markowitch, O.: Side channel attack: an approach based on machine learning. In: Second International Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE 2011 (2011)
17. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.-X.: Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In: Mangard, S., Poschmann, A.Y. (eds.) COSADE 2014. LNCS, vol. 9064, pp. 20–33. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21476-4_2
18. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) SPACE 2016. LNCS, vol. 10076, pp. 3–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49445-6_1
19. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks. Revealing the Secrets of Smart Cards, 1st edn. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-38162-6
20. Mitchell, T.M.: Machine Learning, 1st edn. McGraw-Hill Inc., New York (1997)
21. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML 2010, pp. 807–814. Omnipress, USA (2010). http://dl.acm.org/citation.cfm?id=3104322.3104425
22. O'Shea, K., Nash, R.: An introduction to convolutional neural networks. CoRR abs/1511.08458 (2015)

23. Oswald, E., Mangard, S.: Template attacks on masking—resistance is futile. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 243–256. Springer, Heidelberg (2006). https://doi.org/10.1007/11967668_16
24. Picek, S., Samiotis, I.P., Heuser, A., Kim, J., Bhasin, S., Legay, A.: On the performance of deep learning for side-channel analysis. Cryptology ePrint Archive, Report 2018/004 (2018). https://eprint.iacr.org/2018/004
25. Saravanan, P., Kalpana, P., Preethisri, V., Sneha, V.: Power analysis attack using neural networks with wavelet transform as pre-processor. In: 18th International Symposium on VLSI Design and Test, pp. 1–6, July 2014. https://doi.org/10.1109/ISVDAT.2014.6881059
26. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_3
27. Smith, L.N., Topin, N.: Deep convolutional neural network design patterns. CoRR abs/1611.00847 (2016). http://arxiv.org/abs/1611.00847
28. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
29. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_26
30. TELECOM ParisTech SEN research group: DPA Contest v2. http://www.dpacontest.org/v2/
31. Towell, G.G., Shavlik, J.W.: Knowledge-based artificial neural networks. Artif. Intell. **70**(1–2), 119–165 (1994)
32. Wang, D., Mao, K., Ng, G.W.: Convolutional neural networks and multimodal fusion for text aided image classification. In: 2017 20th International Conference on Information Fusion, Fusion, pp. 1–7, July 2017. https://doi.org/10.23919/ICIF.2017.8009768
33. Whitnall, C., Oswald, E.: Robust profiling for DPA-style attacks. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 3–21. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_1
34. Xie, G.S., Zhang, X.Y., Yan, S., Liu, C.L.: Hybrid CNN and dictionary-based models for scene recognition and domain adaptation. ArXiv e-prints, January 2016
35. Zheng, Y., Zhou, Y., Yu, Z., Hu, C., Zhang, H.: How to compare selections of points of interest for side-channel distinguishers in practice? In: Hui, L.C.K., Qing, S.H., Shi, E., Yiu, S.M. (eds.) ICICS 2014. LNCS, vol. 8958, pp. 200–214. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21966-0_15