Breaking Cryptographic Implementations Using Deep Learning Techniques

Houssem Maghrebi, Thibault Portigliatti*, Emmanuel Prouff

SAFRAN Identity and Security,
18, Chaussée Jules César, 95520 Osny, France.
`firstname.lastname@safrangroup.com`

Abstract. Template attack is the most common and powerful profiled side channel attack. It relies on a realistic assumption regarding the noise of the device under attack: the probability density function of the data is a multivariate Gaussian distribution. To relax this assumption, a recent line of research has investigated new profiling approaches mainly by applying machine learning techniques. The obtained results are commensurate, and in some particular cases better, compared to template attack. In this work, we propose to continue this recent line of research by applying more sophisticated profiling techniques based on deep learning. Our experimental results confirm the overwhelming advantages of the resulting new attacks when targeting both unprotected and protected cryptographic implementations.

Keywords: deep learning, machine learning, side channel attacks, template attack, unprotected AES implementation, masked AES implementation.

1 Introduction

Side Channel Attacks. Side Channel attacks (SCA) are nowadays well known and most designers of secure embedded systems are aware of them. They exploit information leaking from the physical implementations of cryptographic algorithms. Since, this leakage (*e.g.* the power consumption or the electromagnetic emanations) depends on the internally used secret key, the adversary may perform an efficient key-recovery attack to reveal these sensitive data. Since the first public reporting of these threats [30], a lot of effort has been devoted towards the research on side channel attacks and the development of corresponding countermeasures.

Amongst side channel attacks, two classes may be distinguished.

– The so-called *profiling SCA* are the most powerful kind of SCA and consist of two steps. First, the adversary procures a copy of the *target device* and uses it to characterize the dependency between the manipulated data and the device behavior. Secondly, he performs a key-recovery attack on the

* Work done when the author was at SAFRAN Identity and Security.

target device. The set of profiled attacks includes Template attacks [10] and Stochastic cryptanalyses (*aka* Linear Regression Analyses) [16,47,48].
- The set of so-called *non-profiling SCA* corresponds to a much weaker adversary who has only access to the physical leakage captured on the target device. To recover the secret key in use, he performs some statistical analyses to detect dependency between the leakage measurements and this sensitive variable. The set of non-profiled attacks includes Differential Power Analysis (DPA) [30], Correlation Power Analysis (CPA) [9] and Mutual Information Analysis (MIA) [20].

Side Channel Countermeasures. A deep look at the state-of-the-art shows that several countermeasures have been published to deal with side channel attacks. Amongst SCA countermeasures, two classes may be distinguished [36]:

- The set of so-called *masking countermeasures*: the core principle of masking is to ensure that every sensitive variable is randomly split into at least two shares so that the knowledge of a strict sub-part of the shares does not give information on the shared variable itself. Masking can be characterized by the number of random masks used per sensitive variable. So, it is possible to give a general definition for a d^{th}-order masking scheme: every sensitive variable Z is randomly split into $d+1$ shares $M_0, \cdots, M_d$ in such a way that the relation $M_0 \perp \cdots \perp M_d = Z$ is satisfied for a group operation $\perp$ (*e.g.* the XOR operation used in the *Boolean masking*, denoted as $\oplus$) and no tuple of strictly less than $d+1$ shares depends on Z. In the literature, several provably secure higher-order masking schemes have been proposed (see for instance [13], [19] and [44].).
- The set of so-called *hiding countermeasures*: the core idea is to render in making the activity of the physical implementation constant by either adding complementary logic to the existing logic [11] (in a hardware setting) or by using a *specific encoding* of the sensitive data [27,50] (in a software setting).

Machine Learning based Attacks. A recent line of works has investigated new profiling attacks based on Machine Learning (ML) techniques to defeat both unprotected [5,23,28,32,34] and protected cryptographic implementations [21,33]. These contributions focus mainly on two techniques: the Support Vector Machine (SVM) [14,57] and the Random Forest (RF) [45]. Practical results on several data-sets have demonstrated the ability of these attacks to perform successful key recoveries. Besides, authors in [23] have shown that the SVM-based attack outperforms the template attack when applied on highly noisy traces.

Mainly, ML-based attacks exploit the same discriminating criteria (*i.e.* the dependence between the sensitive data and some statistical moments of the leakage) as a template attack. Two major differences between these attacks exist. They are listed hereafter.

- The template attack approximates the data distribution by a multivariate Gaussian distribution (*aka Gaussian leakage assumption*) [10] whose parameters (*i.e.* the mean vector and the covariance matrix) are estimated during

the profiling phase. This implies that the statistical moments of the leakage distribution whose order is greater than 2 are not exploited which can make the attack sub-optimal and even ineffective in some contexts.
– The ML-based attacks make no assumption on the data distribution and build classifications directly from the raw data-set.

Despite the fact that Gaussian leakage is a fairly realistic assumption in side channel context [35,43], applying distribution-agnostic statistical techniques would appear to be a more rational approach.

Our Contribution. Over the past few years, there has been a resurgence of interest in using Deep Learning (DL) techniques which have been applied in several signal processing areas where they have produced interesting results [1,15]. Deep learning is a parallel branch of machine learning which relies on sets of algorithms that attempt to model high-level abstractions in data by using model architectures with multiple processing layers, composed of a sequence of scalar products and non-linear transformations called *activation functions* [51]. Several recent results have demonstrated that DL techniques have convincingly outperformed other existing machine learning approaches in image and automatic speech recognition.

In this work, we propose to apply DL techniques in side channel context. Actually, we highlight the ability of DL to build an accurate profiling leading to an efficient and successful side channel key recovery attack. Our experiments show that our proposed DL-based attacks are more efficient than the ML-based and template attacks when targeting either unprotected or masked cryptographic implementations.

Paper Outline. The paper is organized as follows. In Secs. 2 and 3, we provide an overview on machine learning and deep learning techniques. Then, in Sec. 4 we describe how to use deep learning techniques to perform a successful key recovery. This is followed in Sec. 5 by some practical attack experiments applied on unprotected and masked AES implementations. Finally, Sec. 6 draws general conclusions and opens some perspectives for future work.

2 Overview on Machine Learning Techniques

Machine learning techniques have been developed and used in order to build efficient pattern recognition and features extraction algorithms. Mainly, ML techniques are divided into three categories depending on the learning approach: *unsupervised*, *semi-supervised* and *supervised*. In this paper, we focus on supervised and unsupervised learning techniques.

– *Unsupervised learning* is mainly used when profiling information (*i.e.* training data-set) is not available. So, the purpose is to ensure an efficient data

partitioning without any prior profiling or data modeling. Two classic examples of unsupervised learning techniques are *clustering* (*e.g.* K-means [17]) and *dimensionality reduction* (*e.g.* Principal Component Analysis (PCA)). These techniques have been widely used in side channel contexts to perform either successful key recovery [24,52] or some pre-processing of the physical leakage [4].
– *Supervised learning* refers to techniques that involve a training data-set[1] (*aka* labeled data-set) to build a model. Once the learning has been performed, a supervised learning algorithm is executed which returns, for a new incoming input, an output that is the most accurate one according to the previously learned model. Typical supervised learning techniques include neural networks [8], random forest [45] and support vector machines [14,57].

In the following sections we provide a survey of some supervised learning techniques and their applications in side channel analysis. All of them take as input a training data-set composed of vectors $X^{(i)} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and their corresponding labels $y_i \in \mathbb{R}$ (*e.g.* scores or values of the manipulated sensitive data). After the learning step, their goal is to associate a new vector X with the correct label y.

2.1 Perceptron

The perceptron is the simplest neural network model [8]. It is a linear classifier that uses a learning algorithm to tune its weights in order to minimize a so-called *loss function*[2] as described in Fig. 1. We detail hereafter how perceptron works to perform classification:

– first, an input vector $X = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is presented as an entry to the perceptron.
– then, components of X are summed over the weights $w_i \in \mathbb{R}$ of the perceptron connections (*i.e.* $w_0 + \sum_{i=1}^{n} w_i x_i$, with w_0 being a bias[3]).
– finally, the output of the perceptron is computed by passing the previously computed sum to an *activation function*[4] denoted f.

During the training phase, the perceptron weights, initialized at zeros or small random values, are learned and adjusted according to the profiling data-set ($X^{(i)}$, y_i). By *e.g.* applying a *gardient descent* algorithm, the goal is to find/learn the optimal connecting weights moving the perceptron outputs as close as possible[5]

[1] The training data-set is composed of pairs of some known (input, output).

[2] The loss (*aka* cost, error) function quantifies in a supervised learning problem the compatibility between a prediction and the ground truth label (output). The loss function is typically defined as the negative log-likelihood or the mean squared error.

[3] Introducing a value that is independent of the input shifts the boundary away from the origin.

[4] In the case of the perceptron, the activation function is commonly a Heaviside function. In more complex models (*e.g.* the multilayer perceptron that we will describe in the next section), this function can be chosen to be a sigmoid function (*tanh*).

[5] *e.g.* for the Euclidean distance.

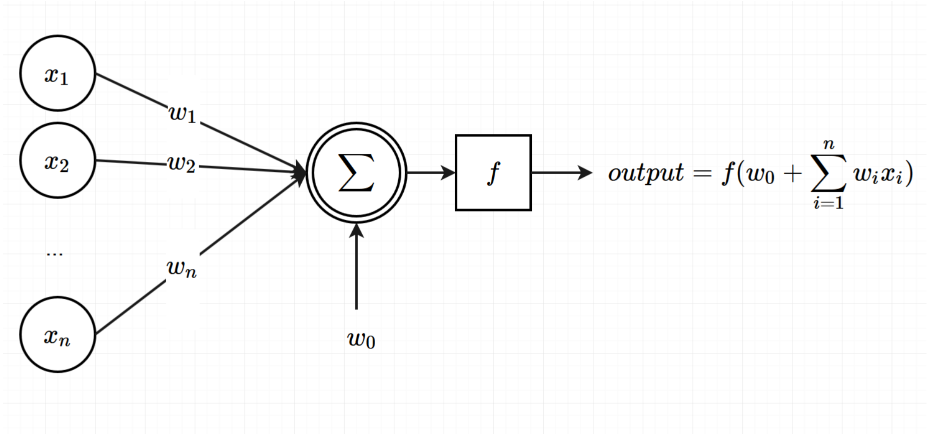

Fig. 1. Representation of a perceptron.

to the correct labels/scores (*e.g.* to minimize the sum of squared differences between the labels y_i and the corresponding perceptron's output).

2.2 Multilayer Perceptron

A Multilayer Perceptron (MLP) is nothing more than a specific way to combine perceptrons[6] in order to build a classifier for more complex data-sets [8]. As shown in Fig. 2, the information is propagated from the left to the right and each units (perceptrons) of a layer is connected to every unit of the previous layer in this model. This is called a *fully connected network*. Each neuron belongs to a layer and the number of layers is a parameter which has to be carefully chosen by the user.

An MLP is made of three different types of layers:

- Input Layer: in the traditional model, this layer is only an intermediate between the input data and the rest of the network. Thus the output of the neurons belonging to this layer is simply the input vector itself.
- Hidden layer: this layer aims at introducing some non-linearity in the model so that the MLP will be able to fit a non-linear separable data-set. Indeed, if the data that have to be learned are linearly separable, there is no need for any hidden layer. Depending on the non-linearity and the complexity of the data model that has to be fit, the number of neurons on the hidden layer or even the number of these layers can be increased. However, one hidden layer is sufficient for a large number of natural problems.
  Regarding the number of neurons on the hidden layers, it has been demonstrated that using a huge number of neurons can lead to *over-fitting* if the

[6] Perceptrons are also called "units", "nodes" or neurons in this model.

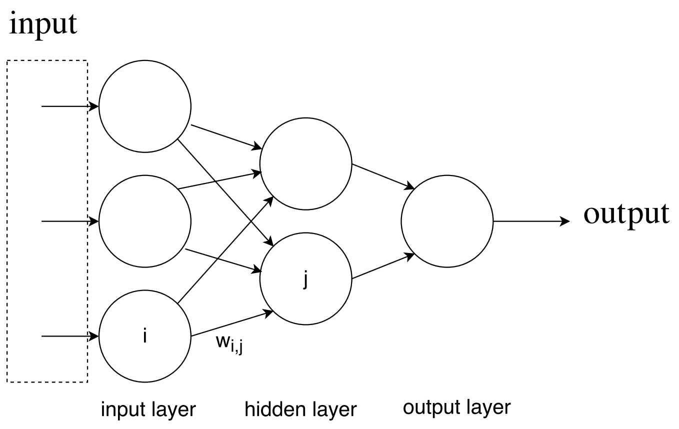

Fig. 2. Example of MLP, where each node is a perceptron as described in Sec. 2.1.

model that has to be learned is close to a linear one [8]. It means that the algorithm is able to correctly learn weights leading to a perfect fit with the training data-set while these weights are not representative of the whole data. On the other hand, the opposite may happen: for a complex data-set, using too few neurons on the hidden layers may lead the gradient minimization approach to fail in returning an accurate solution.

– Output layer: this is the last layer of the network. The output of the nodes on this layer are directly mapped to classes that the user intends to predict.

Training a multilayer perceptron requires, for each layer, the learning of the weighting parameters minimizing the loss function. To do so, the so-called *back-propagation* [8] can be applied. It consists in computing the derivative of the loss function with respect to the weights, one layer after another, and then in modifying the corresponding weights by using the following formula:

$$\Delta w_{ij} = -\frac{\partial E}{\partial w_{i,j}} \ ,$$

where E is the loss function and $w_{i,j}$ denotes the weight of the connection between two neurons of indices $(i,\, j)$.

In several recent works, MLP has been applied to perform successful side channel key recovery. For instance, in [21], authors have presented a neural network based side channel attack to break the masked AES implementation of the DPA contest V4 [55]. In fact, the authors of [21] assume that the adversary has access to the mask values during the profiling phase. Under this assumption, the proposed attack consists first in identifying the mask by applying a neural network mask recovery. Then, a second neural network based attack is performed to recover the secret key with a single trace. While the results of this work are quite interesting, the considered assumption is not always met in real world circumstances.

2.3 Decision Trees and Random Forest

A *decision tree* is a tool involving binary rules to classify data [45]. It is made of a root, several nodes and leaves. Each leaf is associated to a label corresponding to the target value to be recovered. Each node that is not a leaf can lead to two nodes (or leaves). First, the input is presented to the root. It is then forwarded to one of the possible branch starting from this node. The process is repeated until a leaf is reached. An illustration of this process for a 2-bit XOR operation is depicted in Fig. 3.

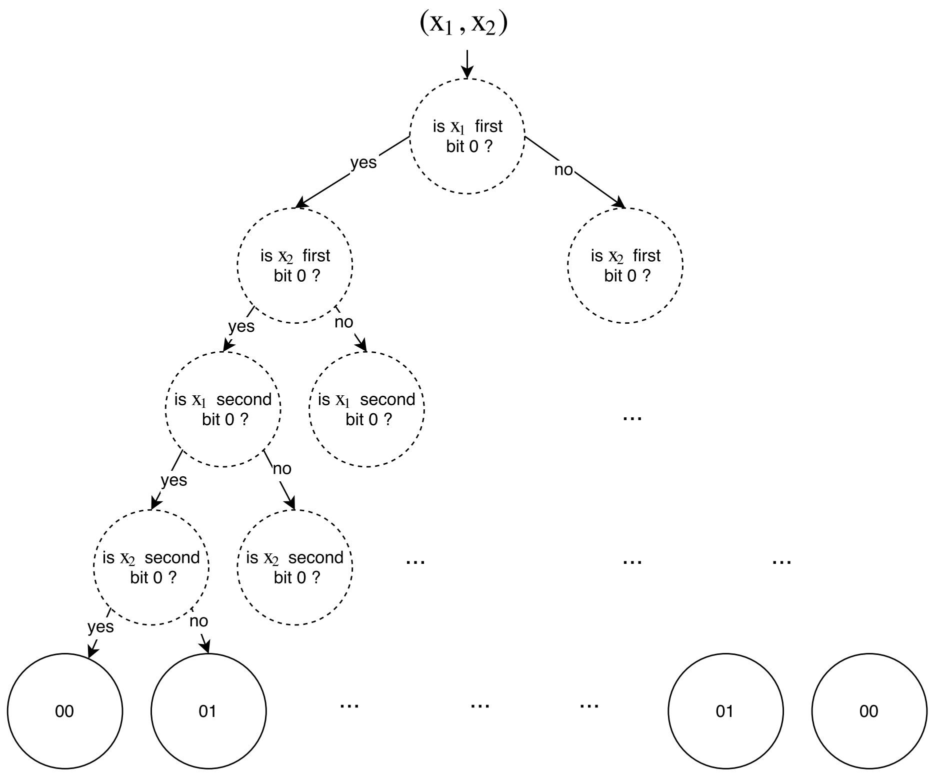

Fig. 3. Partial graphical representation of a decision tree performing the XOR operation between 2 bits variables x_1 and x_2. The leaves correspond to the XOR result.

A random forest is composed of many decision trees, each one working with a different subset of the training data-set [45]. On the top of all of the trees, the global output is computed through a majority vote among these classification trees outputs. RFs have been successfully applied in SCA context to defeat cryptographic implementations [33,34]. In this paper, we will try to compare RF-based attack with deep learning ones in terms of key recovery effectiveness.

2.4 Support Vector Machine

A support vector machine [14,57] is a linear classifier that not only intends to find an hyper-plane to separate data classes but also intends to find the optimal one maximizing the margin between these classes as described in Fig. 4. To deal with non-linearly separable data-sets, it is possible for instance to use a *kernel function* for instance that maps these data into a feature space of higher dimensions in which the classes become linearly separable [49].

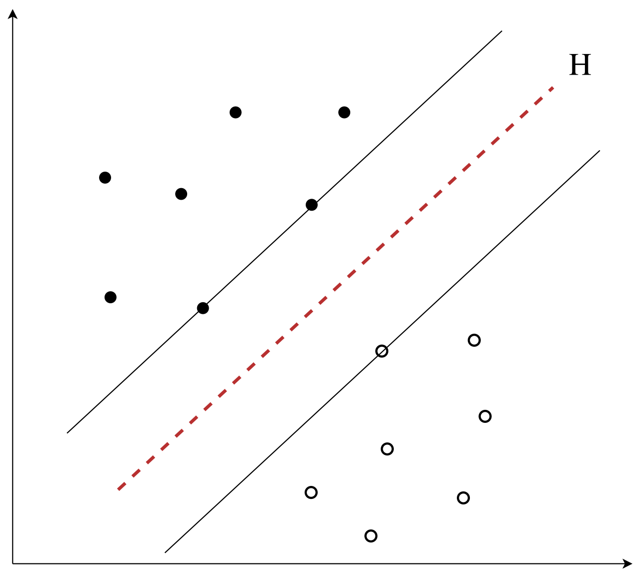

Fig. 4. Binary hyper-plane Classification.

In the side channel literature, several works have investigated the use of SVM towards performing successful attacks to break either unprotected [5,23,28,32,34] or protected cryptographic implementations [33]. Actually, authors in [23] have demonstrated that when the Signal-to-Noise Ratio (SNR)[7] of the targeted data-set is very low, the SVM-based attack outperforms the template attack.

[7] The SNR is defined as the ratio of signal power to the noise power.

3 Overview on Deep Learning Techniques

For several reasons (mainly the *vanishing gradient problem* [25] and the lack of computational power), it was not possible to train many-layered neural networks until a few years ago. Recent discoveries, taking full advantage of GPU for computations and using the *rectified linear unit function* ($f : x \mapsto max(0, x)$) as an activation function instead of the classical sigmoid ($g : x \mapsto \frac{1}{1+e^{-x}}$), made it possible to stack many layers allowing networks to learn more and more abstract representation of the training data-set [29]. This is known as deep learning techniques [1]. One major difference between deep learning and usual machine learning is that the latter ones are classifiers usually working from human-engineered features while the former ones learn the features directly from the raw data before making any classification [6]. In the following sections, some of the most widely used learning techniques are detailed.

3.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a specific kind of neural network built by stacking the following layers [31,40]:

– A convolutional layer: on this layer, during the forward computation phase, the input data are convoluted with some filters. The output of the convolution is commonly called a *feature map*. It shows where the features detected by the filter can be found on the input data. In Fig. 5, we provide an example of a convolutional layer where the input vector X is represented as a matrix (*i.e.* $X = (x_{i,j}) \in \mathbb{R}^{t \times t}$ where t is smallest square integer greater than the size n of X viewed as a vector) and padded with zeros around the border[8]. The output values can be expressed as $y_{i,j} = \sum_{a=1}^{m} \sum_{b=1}^{m} w_{a,b} x_{i+a,j+b}$, where $w_{a,b}$ denotes the weights of the filter viewed as an m-by-m matrix. During the backward computation, the filter weights are learned[9] by trying to minimize the overall loss.
– A Max Pooling layer: this is a sub-sampling layer. The feature map is divided into regions and the output of this layer is the concatenation of the maximum values of all these regions. Such layers can help reducing computation complexity and enhance the robustness of the model with respect to a translation of the input.
– A SoftMax layer: it is added on the top of the previous stacked layers. It converts scores from the previous layer to a probability distribution over the classes.

Learning the filters enables to extract high level features from the data. This step may therefore be used as a dimensionality reduction or a Points Of

[8] The goal is to control the size of the output.
[9] As for the MLP weights estimations, the filter parameters are learned using the back-propagation algorithm.

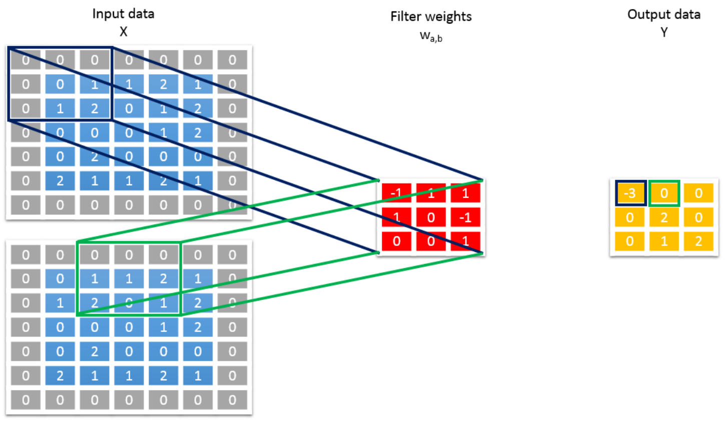

Fig. 5. An example of a convolutional layer where $n = 25$, $t = 5$ and $m = 3$.

Interest (POI) selection technique (*e.g.* a PCA). Based on this remark, it would be interesting to assess the efficiency of the CNN internal features extraction function in selecting the most informative points to perform a successful key recovery attack.

3.2 Stacked Auto-Encoders

Stacked auto-encoders are artificial neural networks with many layers trained by following a very specific procedure [37]. This procedure consists in training each layer independently, using the output of the previous layer as input for the current one. Each layer is composed of an encoder and a decoder, both being a dense layer (*i.e.* fully connected layer)[10]. The role of the encoder is to generate higher level features from the inputs. Whereas, the decoder role is to reconstruct the inputs from the intermediate features learned by the encoder[11] as described in Fig. 6. A very uninteresting network would learn the identity function. To avoid such a behavior, a thumb rule could be that each layer has to be smaller than the previous one[12]. This way the network will be forced to learn a compressed representation of the input. Once the training is done, the decoder is removed, the newly generated encoder is stacked with the previously trained ones and the procedure can be repeated using the output of the newly trained layer.

[10] This is also known as a restricted Boltzmann machine [46].

[11] We refer the interested reader to another type of auto-encoder deep learning technique called *Denoising auto-encoder* [56,58]. This specific kind of auto-encoder aims at removing the noise when fed with a noisy input.

[12] This is not mandatory; some empirical results have shown that it might be better to sometimes have more neurons on the first hidden layer than on the output as a ”pre-learning” step.

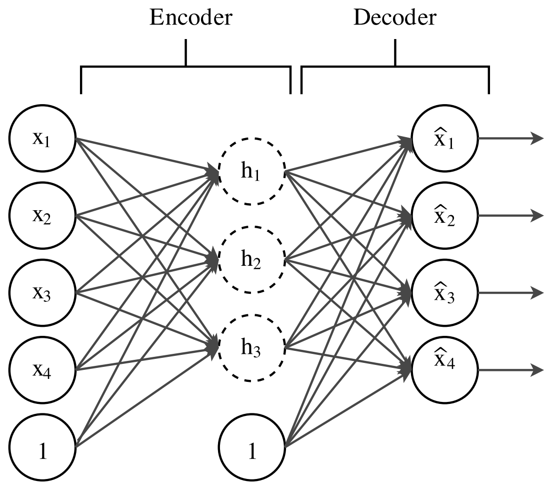

Fig. 6. *Learning an auto-encoder layer.* First, the input $X = (x_0, x_1, x_2, x_3, x_4) \in \mathbb{R}^5$ is encoded. Then, the obtained result $H = (h_0, h_1, h_2, h_3, h_4) \in \mathbb{R}^5$ is decoded using the second layer of the diagram to reconstruct the input $\widehat{X} = (\widehat{x}_0, \widehat{x}_1, \widehat{x}_2, \widehat{x}_3, \widehat{x}_4) \in \mathbb{R}^5$. The difference $(X - \widehat{X})$ is then computed and fed to the back-propagation algorithm to estimate the optimal weights minimizing the loss function.

On the top of the stacked auto-encoder layers, a SoftMax classifier is usually added to predict the class of the input using the high level extracted features of the last layer. Each of these layers (including the SoftMax layer) is trained sequentially. But once the last layer is trained, a global training using the well-known Back-propagation algorithm is performed. This technique is known as *fine tuning* [37].

Like CNN, auto-encoders are features extractors. Their role is to build high level features that are easier to use in a profiling task. This task is particularly meaningful in SCA where the features selection method is critical.

3.3 Recurrent Neural Networks

The Recurrent Neural Networks (RNN) [22] are dedicated to data for which the same information is spread over several time samples. Thus, instead of assuming that the components of the input vectors are mutually independent, each neuron will infer its output from both the current input and output of previous units. The RNN technique could be applied in the context of SCA since the leakage is spread actually over several time samples.

In Fig. 7, we explain how this time-dependency is used by the RNN during the profiling phase. Let n be the number of sample in our trace. For any i in $[1, n]$, the i^{th} output s_i rewrites $s_i = f(U \cdot x_i + W \cdot s_{i-1})$, where (U, W) are the connecting weights that the RNN have to learn and f denotes the activation function. To

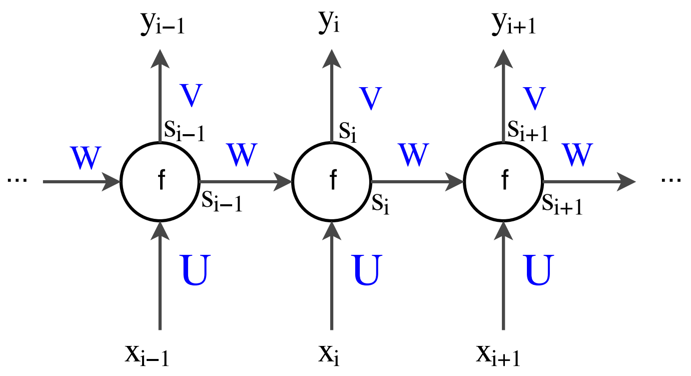

Fig. 7. An unrolled recurrent layer.

get the i^{th} output y_j, a SoftMax layer is added such that $y_j = \text{SoftMax}(V \cdot s_i)$ where V is a connecting weight. Unlike traditional deep learning techniques which use different weights at each layer, a RNN shares the same parameters (U, V, W) across all layers[13]. To adjust the network weights of the i^{th} unit, two different back-propagation phases are processed: the classical one (to learn U) and a temporal one (to learn W which depends on $(i-1)^{\text{th}}$ output).

3.4 Long and Short Term Memory Units

The Long and Short Term Memory (LSTM) is based on the RNN [26]. It has been originally introduced to solve problems that had been reported when using RNN, mainly the vanishing or the exploding gradients [7]. It enables the network to deal with long time lags between relevant time-series of the processed dataset. To do so, a *cell state* (*aka* memory cell) is added inside each unit. It contains some statistical information (*e.g.* mean, variance) computed over a previously processed time-series of the data. This cell can either be written on or erased depending on the relevance of the stored information. The decision of writing on the cell or of clearing it is taken by a small neural network [26].

In side channel context, this feature is quite interesting when dealing with higher-order attacks where the adversary have to combine several delayed time samples in order to defeat masked implementations for instance.

In the rest of this paper, we will focus on LSTM rather than RNN for the reasons outlined above.

[13] The purpose is to reduce the number of parameters to be learned.

4 Towards New Profiling Methods

Several profiling approaches have been introduced in the literature. A common profiling side channel attack is the template attack proposed in [10] which is based on the Gaussian assumption[14]. It is known as the most powerful type of profiling in a SCA context when (1) the Gaussian assumption is verified and (2) the size of the leakage observations is small (typically smaller than 10.000).

When the Gaussian assumption is relaxed, several profiling based side channel attacks have been suggested including techniques based on machine learning. Actually, machine learning models make no assumption on the probability density function of the data. For example, random forest model builds a set of decision trees that classifies the data-set based on a voting system [34] and SVM-based attack discriminates data-set using hyper-plane clustering [23]. Indeed, one of the main drawbacks of the template attacks is their high data complexity [12] as opposed to the ML-based attacks which are generally useful when dealing with very high-dimensional data [34].

In the following section, we describe the commonly used template attack before introducing our new profiling approaches based on deep learning techniques.

4.1 Template Attack

Template attacks have been introduced in 2002 by Chari *et al.* [10]. Since then, many works have been published proposing either some efficiency improvements (*e.g.* using Principal Component Analysis) [4,5,12] or to extend it to break protected implementations [41]. The seminal template attack consists first in using a set of profiling traces[15] and the corresponding intermediate results in order to estimate the probability density function (pdf) $f_z(L|Z = z)$ where Z and L are random variables respectively denoting the target intermediate result and the corresponding leakage during its processing by the device, and where z ranges over all the definition set of Z. Usually L is multivariate, say defined over $\mathbb{R}^d$ for some integer d (*e.g.* $d = 1.000$). Under the Gaussian assumption, this pdf is estimated by a multivariate normal law:

$$f_z(L|Z = z) \simeq \frac{1}{(2\pi)^d det(\Sigma_z)} exp\left(-\frac{1}{2}(L-\mu_z)^T \Sigma_z (L-\mu_z)\right) \quad ,$$

where Σ_z denotes the $(d \times d)$-matrix of covariances of $(L|Z = z)$ and where the d-dimensional vector μ_z denotes its mean[16].

Next, during the attack phase, the adversary uses a new set of traces $(l_i)_{1\leq i\leq n}$ for which the corresponding values z_i are unknown. From a key hypothesis k, he deduces predictions $\hat{z}_i$ on these values and computes the maximum likelihood

[14] which is that the the distribution of the leakage when the algorithm inputs are fixed is well estimated by a Gaussian Law.

[15] This set of traces is typically acquired on an open copy of the targeted device.

[16] The couple (μ_z, Σ_z) represents the template of the value z.

approach $\prod_{j=1}^{n} f_{\hat{z_j}}(l_j|Z=\hat{z_j})$. To minimize approximation errors, it is often more convenient in practice to process the log-likelihood.

4.2 Deep Learning in Side Channel Analysis Context

Like other machine learning techniques (*e.g.* SVM and RF), a deep learning technique builds a profiling model for each possible value z_i of the targeted sensitive variable Z during the training phase and, during the attack phase these models are involved to output the most likely key (*i.e.* label) k^* used during the acquisition of the attack traces set $(l_i)_{1\leq i\leq n}$.

In side channel attack context, an adversary is rather interested in the computation of the probability of each possible value $\hat{z_i}$ deduced from a key hypothesis. Therefore, to recover the good key, the adversary computes the maximum or the log-maximum likelihood approach like for template attack ($\prod_{j=1}^{n} P(l_j|Z=\hat{z_j})$).

Indeed, our deep learning techniques only differs from the machine learning one in the method used to profile data. However, the attack phase remains the same for both kinds of attack.

5 Experimental Results

In the following section, we compare for different implementation sets the effectiveness and the efficiency of our proposed DL-based attacks with those of ML-based and template-based attacks. Mainly, we have targeted a hardware and a software implementation of an unprotected AES and a first-order masked AES implementation.

5.1 Experimental Setup

We detail hereafter our experimental setup.

Attacker Profile. Since we are dealing with profiled attacks, we assume an attacker who has full control of a training device during the profiling phase and is able to measure the power consumption during the execution of a cryptographic algorithm. Then during the attack phase, the adversary aims at recovering the unknown secret key, processed by the same device, by collecting a new set of power consumption traces. To guarantee a fair and realistic attack comparison, we stress the fact that the training and the attack data-sets must be different.

Targeted Operation. Regarding the targeted operation, we consider one or several AES SBox outputs during the first round: $Z = SBox[X \oplus k^*]$ where X and k^* respectively denote the plaintext and the secret key. We motivate our choice towards targeting this non-linear operation by the facts that it is a common target in side channel analysis and that it has a high level of confusion.

Training and Attack Phase Setup. For fair attack comparison, we have considered fixed size data-sets for the profiling and the attack: 1.000 power traces per sensitive value (*i.e.* $Z = z$) for the training phase and 20.000 power traces with a fixed key k^* for the attack phase.

Evaluation Metric. For the different targeted implementations, we have considered a fixed attack setup. In fact, each attack is conducted on 10 independent sets of 2.000 traces each (since we have a set of 20.000 power traces for the attack phase). Then, we have computed the averaged rank of the correct key among all key hypotheses (*aka* the *guessing entropy metric* [53]).

5.2 Unprotected AES Implementations

DPA Contest V2. Our first experiments were carried out on the DPA contest V2 data-set [54]. It is an FPGA-based unprotected AES implementation. Each trace contains 3.253 samples measuring the power consumption of an AES execution.

To break this hardware implementation, we have conducted 4 different DL-based attacks (AE, CNN, LSTM and MLP)[17]. For the MLP-based attack, we have considered two versions: for the first one, we have pre-processed traces by applying a PCA in order to extract the 16 most informative components (since we will target the 16 SBox outputs). For the second MLP-based attack, no dimensionality technique was applied. Our purpose here is to check if the commonly used PCA technique could enhance the efficiency of deep learning based attacks.

For the sake of completeness, we have performed the seminal template attack and the RF-based attack[18]. The evolution of the correct key rank according to the number of traces for each attack when targeting the first AES SBox is described in Fig. 8. Besides, the averaged guessing entropy over the 16 AES SBox is shown in Fig. 9.

From Fig. 9, the following observations may be emphasized:

– the CNN and the AE-based attack slightly outperform template attack. For instance, for the CNN-based attack 200 traces are roughly needed in average to recover the key with a success rate of 100%. For the template attack, an adversary needs roughly 400 traces. This could be explained by the fact that CNN applies a nice features extraction technique based on filters allowing dealing with the most informative samples form the processed traces.
– Prepossessing with PCA does not enhance the efficiency of MLP-based attack. In fact, the PCA is probably removing some data components which are informative for linear clustering representation, but negatively impact the accuracy of the non-linear model profiling of the MLP network.

[17] The parameters for each attack are detailed in Appendix. A.

[18] In our attack experiments, we didn't reported the results of the SVM-based attack since it achieves a comparable results as those obtained for the RF-based attack. The same observations were highlighted in [33].

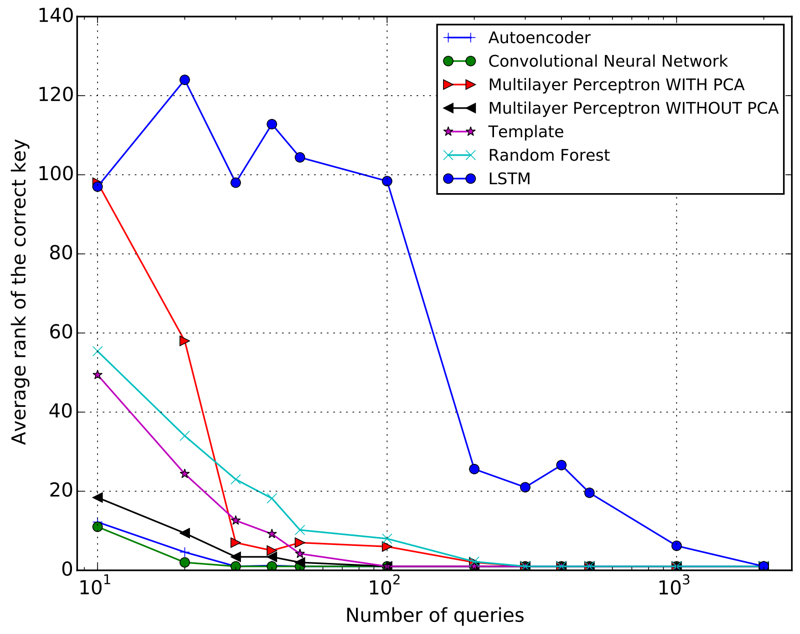

Fig. 8. Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis in log scale base 10) for each attack when targeting the first AES SBox.

– The LSTM performs worse compared to the other types of deep learning techniques. This could be due to the fact that the leakage of this hardware implementation is not time-dependent (*i.e.* the leakage is spread over few time samples).

Software Unprotected AES Implementation. For our second experiments, we have considered an unprotected AES implementation on the ChipWhisperer-Capture Rev2 board [39]. This board is a very compact side channel attack platform. It enables users to quickly and easily test their implementation against side channel attacks.

For the sake of comparison, we have performed the same attacks as these conducted on the DPA contest V2 implementation. In Fig. 10 and Fig. 11, we reported respectively the guessing entropy when targeting the first AES SBox and the averaged guessing entropy over the first four SBoxes for each attack and for an increasing attack traces set.

From Fig. 11, the following observations could be emphasized:

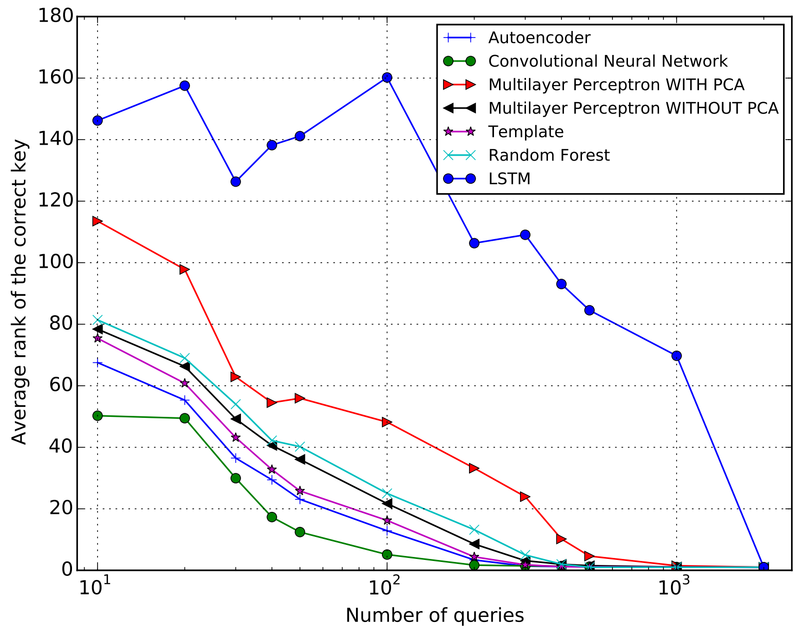

Fig. 9. Averaged guessing entropy over the 16 AES SBoxes (y-axis) according to an increasing number of traces (x-axis in log scale base 10).

– Our proposed deep learning based attacks outperform both template and RF-based attack. For instance, for the AE-based attack 20 traces are roughly needed in average to recover the first four bytes of AES key with a success rate of 100%. For the template attack and RF-based attack, an adversary needs respectively 100 and 80 traces.
– The performed attacks requires less than 100 traces to recover the first four bytes of the key. A natural explanation of this result could be that the SNR is very high on the ChipWhisperer side channel platform.
– The LSTM performs well compared to the results obtained on the DPA contest V2 data-set. This could be due to the facts that the leakage of a software implementation is very time-dependent and that the samples are less noisy.

5.3 First-Order masked AES Implementation

Our last experiments were carried out on a first-order masked AES implementation on the ChipWhisperer-Capture Rev2 board. The 16 SBoxes outputs are

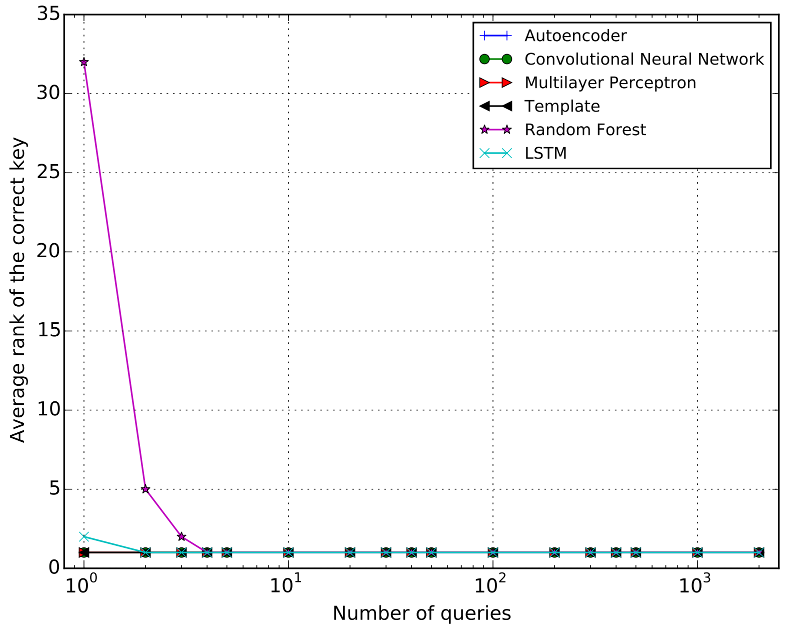

Fig. 10. Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis in log scale base 10) for each attack when targeting the first AES SBox.

masked with the same mask. Our attacks were performed using the same leakage model as that used for the previously evaluated unprotected implementations (*i.e.* the training data were profiled with respect to the SBox output $S[X \oplus k]$). Unlike the recently published ML-based attacks to break masked implementations [21,33], we stress the fact that no prior profiling of the mask values was made during the training phase. The attack results when targeting the first SBox are shown in Fig. 12.

From Fig. 12, one can conclude that our deep learning based attacks perform well against masked implementation. In fact, 500 and 1000 traces are respectively needed for AE and CNN/MLP-based attacks to recover the key. Actually, the deep learning techniques apply some activation functions as described in Sec. 2.1. Those functions (*e.g.* a sigmoid) implicitly perform product combinations of the data samples which has as an effect the removal of the mask dependency[19] exactly like a second-order side channel attack [42].

[19] The product combining function maps the leakages of the masked data $(Z \oplus M)$ and the mask (M) into a univariate sample depending on the sensitive data Z.

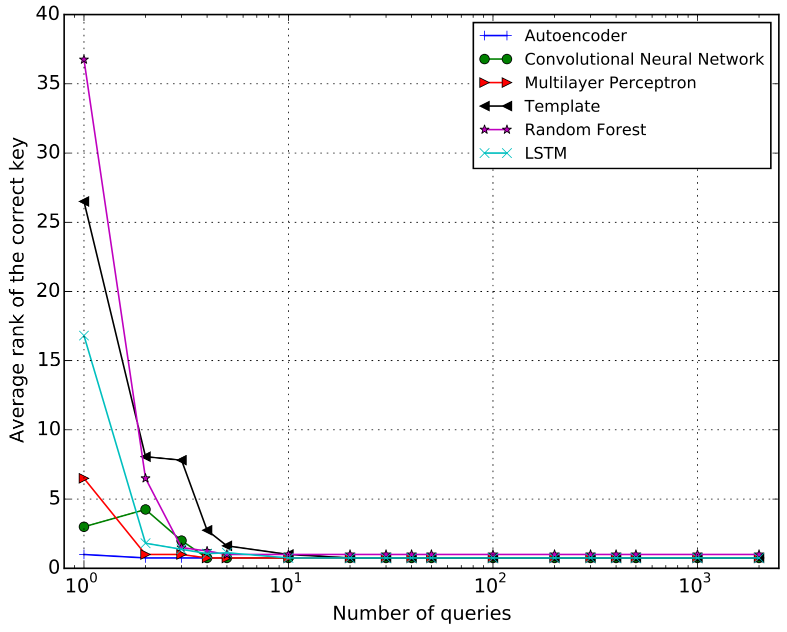

Fig. 11. Averaged guessing entropy over the first four AES SBoxes (y-axis) according to an increasing number of traces (x-axis in log scale base 10).

For template attack and RF-based attack more traces are needed to reach a success rate of 100%.

6 Conclusion and Perspectives

In this paper, to the best of our knowledge, we study for the first time the application of deep learning techniques in the context of side channel attacks. The deep learning techniques are based on some nice features suitable to perform successful key recovery. Mainly, they use different methods of features extraction (CNN and AE) and exploit time dependency of samples (RNN, LSTM). In order to evaluate the efficiency of our proposed attacks, we have compared them to the most commonly used template attack and machine learning attacks. The comparison between these attacks was conducted on three different data-sets by evaluating the number of traces required during the attack phase to achieve a unity guessing entropy with a fixed size of profiling data-set. Our practical results have shown the overwhelming advantage of our proposal in breaking both unprotected and protected AES implementations. Indeed, for the different

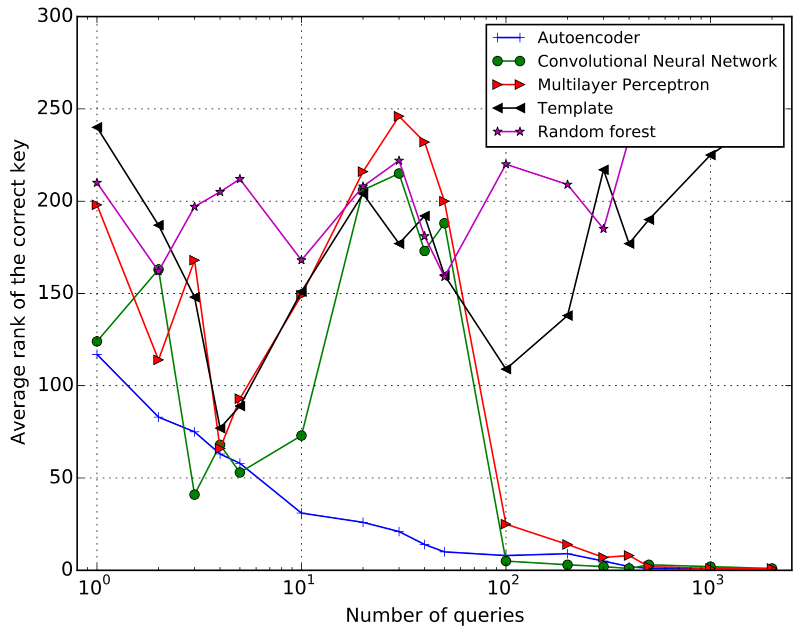

Fig. 12. Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis in log scale base 10) for each attack when targeting the first AES SBox.

targeted implementations, our attacks outperform the state-of-the-art profiling side channel attacks.

A future work may consist in targeting other types of protection (*e.g.* shuffling, combined masking and shuffling) with our proposed DL-based attacks. Moreover, our work opens avenues for further research of new deep learning techniques in order to better adapt them to challenge cryptographic implementations.

References

1. Deep learning website. `http://deeplearning.net/tutorial/`.
2. Keras library. `https://keras.io/`.
3. Scikit-learn library. `http://scikit-learn.org/stable/`.
4. C. Archambeau, É. Peeters, F.-X. Standaert, and J.-J. Quisquater. Template Attacks in Principal Subspaces. In *CHES*, volume 4249 of *LNCS*, pages 1–14. Springer, October 10-13 2006. Yokohama, Japan.

5. T. Bartkewitz and K. Lemke-Rust. *Efficient Template Attacks Based on Probabilistic Multi-class Support Vector Machines*, pages 263–276. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
6. Y. Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, Jan. 2009.
7. Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, Mar. 1994.
8. C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., New York, NY, USA, 1995.
9. É. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, August 11–13 2004. Cambridge, MA, USA.
10. S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, August 2002. San Francisco Bay (Redwood City), USA.
11. Z. Chen and Y. Zhou. Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage. In *CHES*, volume 4249 of *LNCS*, pages 242–254. Springer, October 10-13 2006. Yokohama, Japan, `http://dx.doi.org/10.1007/11894063_20`.
12. O. Choudary and M. G. Kuhn. Efficient Template Attacks. Cryptology ePrint Archive, Report 2013/770, 2013. `http://eprint.iacr.org/2013/770`.
13. J.-S. Coron. Higher Order Masking of Look-Up Tables. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2014.
14. C. Cortes and V. Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, Sept. 1995.
15. L. Deng and D. Yu. Deep learning: Methods and applications. *Found. Trends Signal Process.*, 7(3&#8211;4):197–387, June 2014.
16. J. Doget, E. Prouff, M. Rivain, and F.-X. Standaert. Univariate side channel attacks and leakage modeling. *J. Cryptographic Engineering*, 1(2):123–144, 2011.
17. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2Nd Edition).* Wiley-Interscience, 2000.
18. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* SpringerVerlag, 2003.
19. L. Genelle, E. Prouff, and M. Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.
20. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In *CHES, 10th International Workshop*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, August 10-13 2008. Washington, D.C., USA.
21. R. Gilmore, N. Hanley, and M. O'Neill. Neural network based attack on a masked implementation of aes. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 106–111, May 2015.
22. M. Hermans and B. Schrauwen. Training and analysing deep recurrent neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 190–198. Curran Associates, Inc., 2013.

23. A. Heuser and M. Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In W. Schindler and S. A. Huss, editors, *COSADE*, volume 7275 of *LNCS*, pages 249–264. Springer, 2012.
24. J. Heyszl, A. Ibing, S. Mangard, F. D. Santis, and G. Sigl. Clustering Algorithms for Non-Profiled Single-Execution Attacks on Exponentiations. *IACR Cryptology ePrint Archive*, 2013:438, 2013.
25. S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, Apr. 1998.
26. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.
27. P. Hoogvorst, J.-L. Danger, and G. Duc. Software Implementation of Dual-Rail Representation. In *COSADE*, February 24-25 2011. Darmstadt, Germany.
28. G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011.
29. K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *ICCV*, pages 2146–2153. IEEE, 2009.
30. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO*, volume 1666 of *LNCS*, pages pp 388–397. Springer, 1999.
31. Y. LeCun and Y. Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.
32. L. Lerman, G. Bontempi, and O. Markowitch. Power analysis attack: an approach based on machine learning. *International Journal of Applied Cryptography*, 3(2):97–115, 2014.
33. L. Lerman, S. F. Medeiros, G. Bontempi, and O. Markowitch. A machine learning approach against a masked AES. In A. Francillon and P. Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2013.
34. L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F. Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In S. Mangard and A. Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 20–33. Springer, 2015.
35. V. Lomné, E. Prouff, M. Rivain, T. Roche, and A. Thillard. *How to Estimate the Success Rate of Higher-Order Side-ChannelAttacks*, pages 35–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.
36. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006. ISBN 0-387-30857-1, `http://www.dpabook.org/`.
37. J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, ICANN'11, pages 52–59, Berlin, Heidelberg, 2011. Springer-Verlag.
38. M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

39. C. O'Flynn and Z. D. Chen. Chipwhisperer: An open-source platform for hardware embedded security research. Cryptology ePrint Archive, Report 2014/204, 2014. `http://eprint.iacr.org/2014/204`.
40. K. O'Shea and R. Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.
41. E. Oswald and S. Mangard. Template Attacks on Masking — Resistance Is Futile. In M. Abe, editor, *CT-RSA*, volume 4377 of *Lecture Notes in Computer Science*, pages 243–256. Springer, 2007.
42. E. Prouff, M. Rivain, and R. Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
43. M. Rivain. On the Exact Success Rate of Side Channel Analysis in the Gaussian Model. In *Selected Areas in Cryptography*, volume 5381 of *LNCS*, pages 165–183. Springer, August 14-15 2008. Sackville, New Brunswick, Canada.
44. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
45. L. Rokach and O. Maimon. *Data Mining with Decision Trees: Theroy and Applications.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2008.
46. R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.
47. W. Schindler. Advanced stochastic methods in side channel analysis on block ciphers in the presence of masking. *Journal of Mathematical Cryptology*, 2(3):291–310, October 2008. ISSN (Online) 1862-2984, ISSN (Print) 1862-2976, DOI: 10.1515/JMC.2008.013.
48. W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In LNCS, editor, *CHES*, volume 3659 of *LNCS*, pages 30–46. Springer, Sept 2005. Edinburgh, Scotland, UK.
49. B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001.
50. V. Servant, N. Debande, H. Maghrebi, and J. Bringer. *Study of a Novel Software Constant Weight Implementation*, pages 35–48. Springer International Publishing, Cham, 2015.
51. T. C. Silva and L. Zhao. *Machine Learning in Complex Networks.* Springer, 2016.
52. Y. Souissi, M. Nassar, S. Guilley, J.-L. Danger, and F. Flament. First Principal Components Analysis: A New Side Channel Distinguisher. In K. H. Rhee and D. Nyang, editors, *ICISC*, volume 6829 of *Lecture Notes in Computer Science*, pages 407–419. Springer, 2010.
53. F.-X. Standaert, T. Malkin, and M. Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *EUROCRYPT*, volume 5479 of *LNCS*, pages 443–461. Springer, April 26-30 2009. Cologne, Germany.
54. TELECOM ParisTech SEN research group. DPA Contest (2$^{\text{nd}}$ edition), 2009–2010. `http://www.DPAcontest.org/v2/`.
55. TELECOM ParisTech SEN research group. DPA Contest (4$^{\text{th}}$ edition), 2013–2014. `http://www.DPAcontest.org/v4/`.
56. P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

57. J. Weston and C. Watkins. Multi-class support vector machines, 1998.
58. J. Xie, L. Xu, and E. Chen. Image denoising and inpainting with deep neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 341–349. Curran Associates, Inc., 2012.

A Attack Settings

Our proposed deep learning attacks are based on Keras library [2]. We provide hereafter the architecture and the used parameters for our deep learning networks.

- Multilayer Perceptron:
  - Dense input layer: the number of neurons = the number of samples in the processed trace
  - Dense hidden layer: 20 neurons
  - Dense output layer: 256 neurons
- Stacked Auto-Encoder:
  - Dense input layer: the number of neurons = the number of samples in the processed trace
  - Dense hidden layer: 100 neurons
  - Dense hidden layer: 50 neurons
  - Dense hidden layer: 20 neurons
  - Dense output layer: 256 neurons
- Convolutionnal Neural Network:
  - Convolution layer
    - Number of filters: 8
    - Filters length: 16
    - Activation function: Rectified Linear Unit
  - Dropout
  - Max pooling layer with a pooling size: 2
  - Convolution layer
    - Number of filters: 8
    - Filters length: 8
    - Activation function: $tanh(x)$
  - Dropout
  - Dense output layer: 256 neurons
- Long and Short Term Memory:
  - LSTM layer: 26 units
  - LSTM layer: 26 units
  - Dense output layer: 256 neurons
- Random Forest: For this machine learning based attack, we have used the *scikit-learn* python library [3].
  - Number of trees: 300

In several published works [23,28], authors have noticed the influence of the parameters chosen for SVM and RF networks on the attack results. When dealing with deep learning techniques we have observed the same effect. To find the optimal parameters setup for our practical attacks, a deeply analyzed method is detailed in the following section.

A.1 How to Choose the Optimal Parameters?

When dealing with *artificial neural networks*, several meta-parameters have to be tuned (*e.g.* number of layers, number of neurons on each layer, activation function, ...). One common technique to find the optimal parameters is to use *evolutionary algorithms* [18] and more precisely the so-called *genetic algorithm* [38].

At the beginning of the algorithm, a *population* (a set of *individuals* with different *genes*) is randomly initialized. In our case, an individual is a list of the parameters we want to estimate (*e.g.* number of layers, number of neurons on each layer, activation function, ...) and the genes are the corresponding values. Then, the performance of each individual is evaluated using what is called a *fitness function*. In our context, the fitness function is the guessing entropy outputted by the attack. Said, differently, for each set of parameters we perform the attack and we note the guessing entropy obtained. Only the individuals that achieve good guessing entropy scores are kept. Their genes are mutated and mixed to generate a better population. This process is repeated until a satisfying fitness is achieved (*i.e.* a guessing entropy equals one).