# Scoring the predictions: a way to improve profiling side-channel attacks

Damien Robissout[1,2], Lilian Bossuet[2] and Amaury Habrard[2+,3]

[1] IMDEA Software Institute, Madrid, Spain, name.surname@imdea.org
[2] Université Jean Monnet Saint-Etienne, CNRS, Institut d'Optique Graduate School, Inria[+], Laboratoire Hubert Curien UMR 5516, F-42023, Saint-Etienne, France
name.surname@univ-st-etienne.fr
[3] Institut Universitaire de France (IUF), France

**Abstract.** Side-channel analysis is an important part of the security evaluations of hardware components and more specifically of those that include cryptographic algorithms. Profiling attacks are among the most powerful attacks as they assume the attacker has access to a clone device of the one under attack. Using the clone device allows the attacker to make a profile of physical leakages linked to the execution of algorithms. This work focuses on the characteristics of this profile and the information that can be extracted from its application to the attack traces. More specifically, looking at unsuccessful attacks, it shows that by carefully ordering the attack traces used and limiting their number, better results can be achieved with the same profile. Using this method allows us to consider the classical attack method, i.e. where the traces are randomly ordered, as the worst case scenario. The best case scenario is when the attacker is able to successfully order and select the best attack traces. A method for identifying efficient ordering when using deep learning models as profiles is also provided. A new loss function "Scoring loss" is dedicated to training machine learning models that give a score to the attack prediction and the score can be used to order the predictions.

## 1 Introduction

Side-channel analysis is one of the most threatening attacks possible against secure hardware components. They are based on a careful study of a physical value, such as power consumption or electromagnetic emanations, linked to computations underway in the device. When the computations concern secure algorithms, the resulting leakages can yield important information concerning secret variables. If analyzed correctly, the leaks can impact the level of security of the device and even enable recovery of the secret key.

One of the most powerful side-channel attacks is the template attack [CRR03]. This type of attack is based on the assumption that the adversary has access to a perfect copy of the device under attack. Using this copy, the attacker can characterize the power consumption leakages of specific operations and use statistical analysis to create templates. By applying the templates to the power consumption of the target device, the attacker can then guess the secret key used for the computations based on the probabilities obtained. Even though the assumptions needed to perform these kinds of attacks are strong (access to a similar device, the chosen plaintext, leaking intermediate value, etc), they are taken into consideration in security evaluations of hardware components. However, one of the only downside is that some preprocessing of the power traces is also required. Preprocessing typically includes alignment of the traces, i.e. synchronization of the power consumption of the device across several executions of the encryption algorithm, and detection of *points of interest* (PoI) that correspond to points where information concerning the intermediate values of the algorithm leaks.

A new kind of attack, very similar to the template attack, has appeared in the last few years based on machine learning algorithms, for example deep neural networks (DNN). These networks are trained to do what the templates are used for, i.e. assign probabilities to each possible intermediate value based on its likelihood, with the highest value corresponding to the most likely guess. In addition, once the learning is complete, they automatically find the PoIs in the traces and, due to the way they recombine the input data in their intermediate layers, they are somewhat resilient to desynchronization. This new way of performing profiling attacks has attracted considerable attention as in most scenarios, it performs better than classical template attacks. However, to maximize its performance, it relies on finding the best architecture, i.e. the best combination of hyperparameters, for the neural network. Tuning the hyperparameters [ZBHV20], i.e. the exact architecture of the networks, is what usually takes the most time and can lead to widely different results between good and bad tuning. Another important aspect to consider when training a neural network is the overfitting phenomenon [RBHG21]. Overfitting is learning to predict values using the full power consumption trace instead of only the PoIs, which has a direct impact on the performances of the attack as the learned features are not representative of the attack traces. All in all, training an efficient neural network is not an easy task and often results in underperforming networks that are able to rank the correct key relatively high but unable to recover it completely.

**Contributions** In this work, the focus is on particular cases where the networks are underperforming and our objective is to explore a new way to perform the profiling attacks. The new method is based on the use of a fixed network architecture. In other words, the network used to obtain the predictions is not retrained or modified to obtain a more accurate model. In addition, the attacks performed in the experimental sections use a fixed number of traces to simulate a case where the adversary has limited access to attack traces. Using the results of the network, it is possible to identify its level of confidence in different traces and hence to deduce an order to use for the attack traces; this means that when accumulating the probabilities contained in the network's prediction of the attack traces, a specific order is used for the predictions. In turn, this makes it possible to reach a new rank that is lower than the final rank obtained without ordering, thereby improving the attacks. The classical way of performing profiling attacks is shown to correspond to a worst case scenario for the attacker and that by properly ordering the predictions, the rank of the good key can be lowered and in some cases the key can be recovered.

**Article organization** The following section of this article provides an overview of the key concepts of side-channel analysis. Section 3 formalizes the new ordering problem introduced and explains the changes to the attack method. Section 4 details a first approach to predicting the order of the attack predictions based on the level of confidence of the network that can be considered a first step toward solving the problem. Section 4 also contains an experimental section with the details of the tests performed to validate the new approach. Section 5 explains how machine learning can be used to solve the problem of ordering the predictions based on the learning to rank approach. Section 5 also focuses on how to adapt the Ranking Loss, that is, a loss function adapted from the learning to rank problem to train DNN to perform side-channel attacks more efficiently, into the Scoring Loss, a loss function that can be used to train neural networks to solve the ordering problem. Finally, Section 6 contains the experimental results of the approach and the application of this approach to the problem of ordering the predictions.

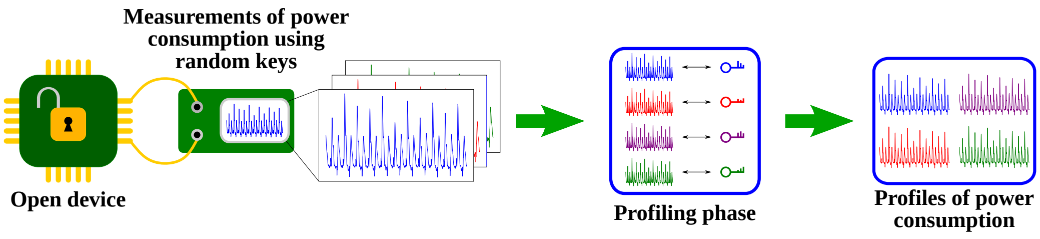

Measurements of power consumption using random keys
Open device
Profiling phase
Profiles of power consumption

Figure 1: The steps of the profiling phase.

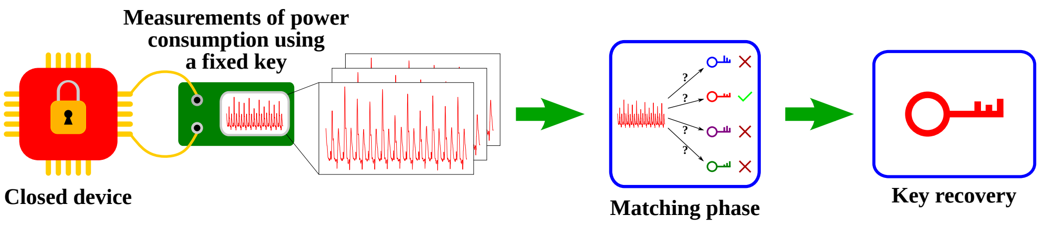


Figure 2: The steps of the matching phase.

## 2 Preliminaries and notations

### 2.1 Profiling side-channel analysis

The goal of profiling side-channel analysis is to retrieve a secret value or parts of a secret value used in the computation of an encryption algorithm that can lead to a complete or partial recovery of a secret key by the attacker. However, profiling side-channel analysis requires making strong assumptions about the power of the attacker. It assumes the attacker has access to a physical measurement from the device used to perform the encryption and to a copy of the device under attack for which the attacker has total control over the values manipulated. It also assumes information on the values of intermediate variables is contained in the measurements and that this information is related to the secret or part of the secret.

The physical measurements used in this article are assumed to be power consumption. They are stored in temporal vectors called traces, denoted $\mathbf{t}$, and grouped in a set $\mathbf{T} \in \mathbb{R}^{N \times D}$ where $N$ is the number of traces in $\mathbf{T}$ and $D$ is the dimension of the traces. The targeted intermediate variable is $Z = f(P, K)$. Here $f$ represents parts of the computation of a cryptographic primitive, $P$ $(\in \mathcal{P})$ a public variable (e.g. a plaintext or ciphertext) and $K$ $(\in \mathcal{K})$ a part of the key (e.g. a byte). The goal of the attacker is to retrieve the value of $\mathbf{k}^*$, the secret key used by the cryptographic algorithm. To do so, the attacker uses the *divide and conquer* approach to find fractions of the key (e.g. a bit or a byte) separately and then to combine them to obtain the full key. This article focuses on the use of profiling attacks to recover the different parts of the secret key. These attacks can be broken down into two separate phases: the profiling phase, described in Figure 1, and the matching phase, described in Figure 2.

During the profiling phase, the adversary has access to a test device, often called open device, which is a copy of the device targeted by the attack and over which the attacker has total control as well as knowledge of the values used by the algorithm. This means the attacker can determine where the leakages of the sensitive variable $Z$ $(\in \mathcal{Z})$ can be found in the traces. Using this knowledge, the adversary builds a model $F : \mathbb{R}^D \to \mathbb{R}^{|\mathcal{Z}|}$ to estimate the probability $Pr[Z = z \mid \mathbf{T}]$, i.e. the probability that, given the set of traces $\mathbf{T}$, the value of the sensitive variable is $z$.

Once the model is created, the attacker can use it during the matching phase to estimate the values of the intermediate variable in the device under attack. To do so, the attacker needs to acquire traces from the device under attack, also known as closed device, for which he only controls the plaintext, $P$ ($\in \mathcal{P}$) to be encrypted by the algorithm, plus knowledge of the resulting ciphertext. In order to have a uniform distribution of the secret variable $Z$, the plaintext value must also follow a uniform distribution, since the key value is fixed in the closed devise. Using the traces obtained during the encryptions, the attacker computes the probabilities $Pr[Z = z \mid \mathbf{T}]$ and combines them. The value with the highest probability is then considered as the recovered key in the identity leakage model.

To evaluate the performance of the model, all the key candidates are classified in a vector of size $\mid \mathcal{K} \mid$, denoted $\mathbf{g} = (g_1, g_2, ..., g_{|\mathcal{K}|})$, according to their respective probability. $g_1$ is considered to be the most likely candidate and $g_{|\mathcal{K}|}$ the least likely. The actual position of the $b^{th}$ byte of the secret key in $\mathbf{g}$ is denoted $\mathbf{g}(\mathbf{k}^*[b])$ and called *rank*. The *guessing entropy* [SMY09] is defined as the average rank of a byte $b$ of $\mathbf{k}^*$, denoted $\mathbf{k}^*[b]$, among all key hypotheses. The guessing entropy is a common metric in side-channel analysis to evaluate the performance of attacks. A successful attack, using $N_a$ traces, is equivalent to a guessing entropy equal to 1.

## 3 Introduction to the problem

This section presents a new way to perform profiling side-channel attacks using ordering of the traces. Typically, when performing side-channel analysis, the leakage traces are used indiscriminately in the sense that no one trace holds more information about the sensitive variable than any other. In this work, a new approach to such attacks is explored based on the following hypothesis: the amount of information extracted from a trace depends on the model used, a template or a neural network, and it is possible to discriminate correctly predicted traces from the others. This approach raises a certain number of questions that are discussed in this article.

### 3.1 Discussion of the problem

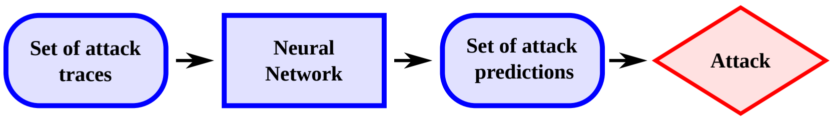
The problem can be reformulated as: is it possible that some pairs (trace, prediction) have a negative impact on the key recovery? If yes, is it possible to detect these pairs and to isolate them to improve the results of the attack?

These questions are first applied to the use of deep neural networks to predict the value of the sensitive variable in profiling attacks. Indeed, training neural networks can be difficult, especially in the presence of countermeasure in the targets, such as desynchronization or masking. Such cases can require considerable hyperparameter tuning and training to obtain an accurate network. This can be seen in [HHO20] and [AGF22], where the authors had to apply new and sometimes complex techniques to improve the performance of models trained on synchronized data against desynchronized traces. This process can be very time consuming and hence problematic, for example in the evaluation of a hardware component by an IT Security Evaluation Facility (ITSEF). Consequently, the trained neural networks are usually not the most accurate and can produce better results when applied to traces from the training set. This means that it is easier for a network to predict traces close to the training traces with more confidence. The question is then: is it possible to detect such traces inside the attack set for which the secret key is not known?

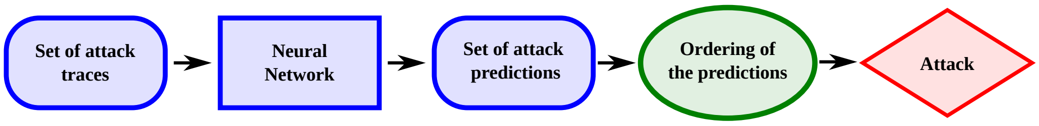
Another important point concerns template attacks that differ from attacks based on deep learning for the computation of the predictions but, like in those attacks, the template attack relies on the use of profiling traces similar to training traces. Therefore, the same type of bias may be present in the template attack and thus there is a need to look for differences in the way the traces are predicted.

### 3.2 Modification of the classical attack scheme

This new approach changes the usual way of performing an attack in profiling side-channel analysis. The base method is described in Figure 3a. Once the model is trained, the attack traces are fed to the model and the model then outputs a set of



(a) Classical attack method



(b) Attack method using ordering

Figure 3: Steps in (a) the classical attack method and (b) in the attack method using ordering.

predictions linked to the traces. The predictions are then randomly selected and used to perform the attack by accumulating the information on the key either through the probabilities or through the scores. Once all the predictions are used, the guess vector containing all the possible key values is computed and the key is recovered or not.

This new way of performing the attacks resembles the classical way of attacking but has an additional key step between computing the predictions and the attack. The additional step consists in computing the order of the traces based on their predictions by the model (Figure 3b). During the attack, the order is used to determine the order with which to use the traces in the attack, i.e. to determine which prediction will be used next in the accumulation of probability. In the end, the same traces are used for both methods of attack only not in the same order so the end result is the same, i.e. the final guess vectors are equal. The only difference is the evolution of the rank of the correct key.

To summarize the introduction of the new method of attack, we now provide a description of the process we used in the experiments to obtain the results shown. First, a fixed number of attack traces were selected from the attack set. Then, two attacks were performed, the classical attack and the attack using the proposed new method. In the first attack, the order in which the predictions are used in the evolution of the rank was randomized before the attack is performed. In the second attack, an order was computed and used to obtain the evolution of the rank. This is the method used to obtain the ranking shown in Figure 4. Once both attacks were complete, the process was repeated using a different subset of attack traces again drawn from the attack set. All the evolutions of the ranks were then averaged to obtain the final results.

The next section formalizes this difference between the methods and gives an example to illustrate this new approach.

## 3.3 Formalization of the problem

To properly formalize the problem, let us recall and define some terms:

- $\mathcal{T}$: the set of attack traces;
- $F_\theta$: a model with parameters $\theta$ that makes predictions on the traces $t \in \mathcal{T}$;
- $\mathcal{P}$: the set of predictions of $F_\theta$ on $\mathcal{T}$, i.e. $\mathcal{P} = F_\theta(\mathcal{T})$;
- $\mathrm{Adv}_{\mathcal{P}}(n)$: an adversary using $n$ predictions of $\mathcal{P}$ to recover the key $k^*$;
- $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}}(n)} = r_n$: the result of the attack of $\mathrm{Adv}_{\mathcal{P}}(n)$ expressed using the rank $r_n$ of the key $k^*$ in the guess vector $\mathbf{g}$ containing the key hypotheses sorted according to the results of the attacks.

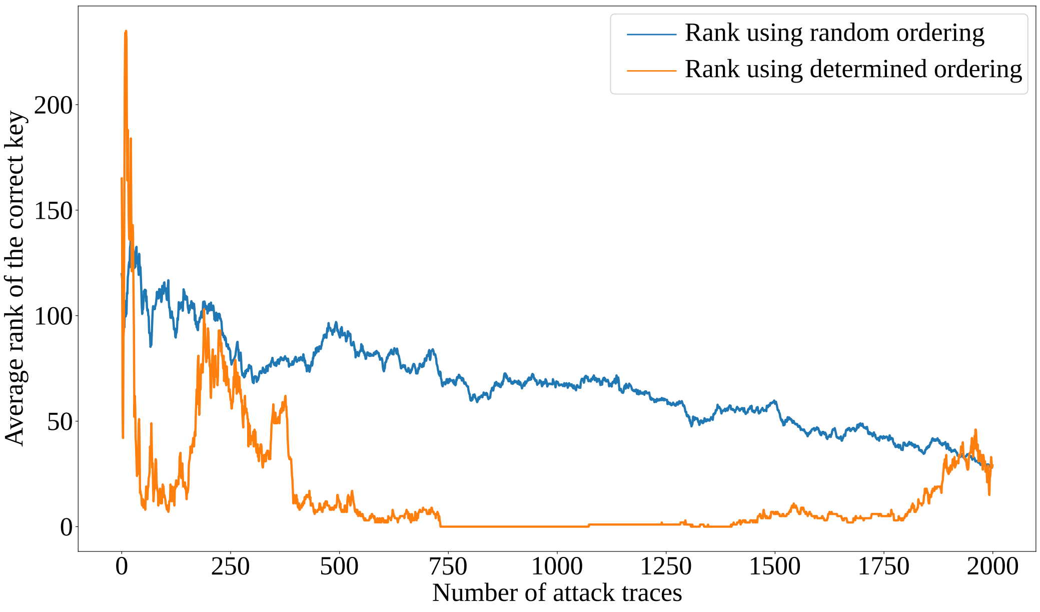


Figure 4: Evolution of the average rank of the correct key using the same predictions for the attack with or without ordering.

From these definitions, if $\text{Exp}_{\text{Adv}_{\mathcal{P}}(n)} = 1$ then the key $k^*$ is successfully recovered using $n$ traces. Conversely, if $\text{Exp}_{\text{Adv}_{\mathcal{P}}(n)} = r_n > 1$ then the key is not directly recovered. In the latter case, one can try to reduce the value of $r_n$ by increasing the number $n$ of traces used in the attack. However, it is not always possible to acquire more traces and sometimes there is a maximum value $n_{\max}$ of the number of traces usable for the attack, which creates a problem when $\text{Exp}_{\text{Adv}_{\mathcal{P}}(n_{\max})} = r_{n_{\max}} > 1$. This is where this work comes into play and poses the following problem.

**Problem** Find an order $o$, where $\mathcal{P}_o$ represents the predictions of the traces ordered according to $o$, such that there exists $n_o \in \mathbb{N}^*$ with $n_o < n_{\max}$ such that:

$$\text{Exp}_{\text{Adv}_{\mathcal{P}_o}(n_o)} = r_{n_o} < r_{n_{\max}}.$$

Figure 4 illustrate such a phenomenon. It shows two different evolutions of rank based on the same set of predictions. One of the evolutions uses a random ordering of its traces and the other uses a determined ordering. Since the set of traces is the same, the final rank is the same but the evolutions of the ranks differ considerably. The rank of the attack using random ordering decreases at a more or less fixed rate before reaching the final rank value whereas the rank of the attack using a determined attack order varies much more, i.e. decreases rapidly to rank 1 and reaches it using 750 traces. The rank of the correct key remains at 1 for a few hundred traces before starting to increase to reach the final rank value $r_{n_{\max}}$. In this example, using the classical attack method, a false conclusion can be reached: *the attack does not succeed using $n_{max}$ traces*. However, given that the $n_{\max}$ predictions are correctly ordered, the attack may succeed when $n_o$ traces are used. In the rest of this article, the predictions are used to establish the order and are referred as "orderings of the predictions" or "ordered predictions".

**Remark 1** The goal of this article is to identify and analyse different ways to establish the order $o$. As can be seen in the example, being able to find the order and the right number of traces to use can reduce the rank of the secret key and even in some cases, can recover it. This possibility calls into question some security evaluations

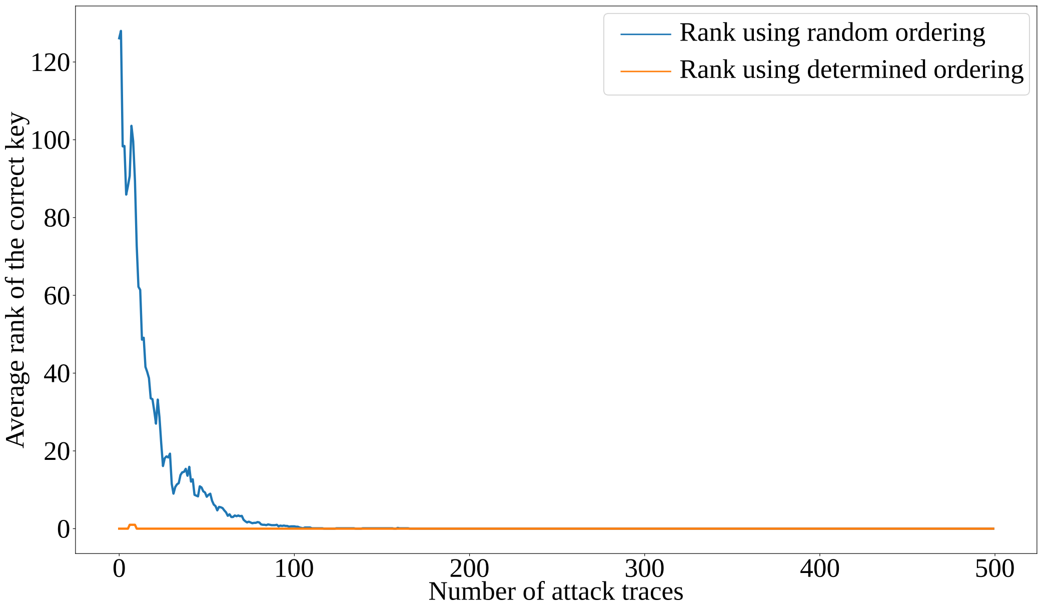


Figure 5: Evolution of the average rank of the correct key using two different types of ordering while the attack is successful.

that certify the security of systems that may be vulnerable to this new attack method. However, some aspects of this new approach need to be clarified, first: *it is not a way to improve the results of successful attacks.* Indeed, if $\text{Exp}_{\text{Adv}_{\mathcal{P}}(n_{\max})} = 1$ and an order $o$ and a number of traces $n_o$ are found such that $\text{Exp}_{\text{Adv}_{\mathcal{P}_o}(n_o)} = 1$, this is not an improvement over the first attack. The key aspect to consider when performing a side-channel attack is the number of traces the attacker needs to acquire in order to retrieve the key. If an attack succeeds using $n_{\max}$ traces, then an attack that is successful using $n_o$ traces is not an improvement if $n_{\max}$ traces are needed to find this subset. This is illustrated in Figure 5 where two attacks, using the same dataset and different orders, one random and one determined, both manage to recover the key in less than 500 traces, and can therefore be considered to be equivalent.

**Remark 2** Following this clarification, the main effect of this new approach can be discussed. By showing the possibility of making the rank converge toward 1 by carefully selecting the traces used through ordering, the different possible cases of a side-channel attack when $\text{Exp}_{\text{Adv}_{\mathcal{P}}(n_{\max})} > 1$ can be redefined. The best case scenario for the attacker becomes the case where it is possible to find an order $o$ and a number of traces $n_o$ such that $\text{Exp}_{\text{Adv}_{\mathcal{P}_o}(n_o)} = 1$. This means that an attack that failed using the classical method is now able to recover the secret key. The worst case scenario now corresponds to the case where such an order cannot be established and the attacker uses the classical method. Indeed, the attacks fall in two categories, one where an order cannot be determined and the final rank is considered the best rank, and one where an order improving this rank can be found and the final rank is no longer the best rank. This new dichotomy means that the new approach described here can only be an improvement. This is the main reason why this method can call security evaluations into question in cases where it is not taken into consideration.

**Remark 3** To complete the point made in the previous remark, let $\mathcal{P}$ be a set of $n_{\max}$ predictions of attack traces and $o$ be any ordering of $\mathcal{P}$, then:

$$\text{Exp}_{\text{Adv}_{\mathcal{P}_o}(n_{\max})} = r_{n_{\max}},$$

where $r_{n_{\max}}$ is the final rank reached when all the predictions in the set have been used. This property comes from the fact that, given the same set of traces, the result of the attack is always the same due to the deterministic nature of the probability accumulation.

**Remark 4** As a general remark, it should be noted that this new attack method does not guarantee the recovery of the correct key as finding the best possible order is very difficult. However, in most scenarios where the attack is unsuccessful, i.e. the minimum rank is still greater than zero, the minimum rank is still lower than the final rank. This means that from then on, it is possible to perform a key enumeration in a smaller search space in order to recover the key byte, for example. The process can be repeated for each byte of the key until the recovery is complete. The application of this new method can thus lead to a global reduction of the search space for each byte of the key.

**Remark 5** This final remark discusses how to find the best number of traces to use when performing ordered attacks. This question is beyond the scope of the present article and is left for future research. However, some leads that are worth exploring are mentioned in Section 6.3.

The following section focuses on the study of attacks that do not succeed even when the whole dataset is used to identify an order that improves the result of the attack. For this purpose, a way to distinguish between two types of predictions, those that help the rank to converge and those that have a negative impact on the rank of the correct key, needs to be established.

## 4 Study of the confidence of the models

As mentioned in the previous section, it is important to be able to distinguish between predictions that help achieve convergence of the rank of the correct key and predictions that have a negative impact on this rank. This distinction is possible due to the method of attack that is based on training models to make predictions on the possible value of the intermediate variable. The training phase is done using profiling traces that are similar to the traces used to attack the device.

However, it is impossible to obtain a training set that perfectly represents the physical leakage of the device, which means that training using this dataset necessarily introduces a bias in the final model. This training creates an imbalance between the model's prediction of the traces from the training set and the traces from the attack set, as the model is much more precise when predicting the profiling traces. This is confirmed by the fact that attacks on the training traces always require fewer traces to succeed than attacks using attack traces. Thus, it is possible to infer that the model extracts more information from the traces that were used to train it than from the unknown traces of the attack set. It is also possible to infer that the traces from the attack set which are the most similar to traces from the training set are more likely to be better predicted than others. This is the origin of the distinction between the well predicted traces and the other traces. However, this distinction is not possible using only the leakage traces or it would need the attacker to be able to perfectly characterize the leakage, which in practice, is very difficult. One possible solution to this problem is to try to use the predictions of the model to make this distinction.

Indeed, the focus is not on the traces themselves but on the way the model predicts them and it is consequently interesting to examine the distribution of the values within each prediction. After which the distinction between the traces is based on the distinction between the well predicted traces and the other traces. This supports the hypothesis that each trace contains the same level of information but that the model's exploitation of the information differs from trace to trace. The difference between well predicted traces and the others is linked to the capacity of the model to extract relevant information.

As such, a way to characterize the distinction at the attack level is needed. As a preliminary phase, we studied the behavior of predictions originating from the validation and test sets for which the key is known. This means the label of each trace is also known and so are the values of the labels inside the predictions. This knowledge gives access to the position of the labels among the other possible classes for each predictions. This is equivalent to observing the rank of attacks using only one trace. In the rest of the article, the term *rank of a prediction* is used to describe the position of its label among all the possible classes ranked from the most probable to the least probable. Using the ranks of the predictions, it is possible to determine what can be called the *perfect order* in which the predictions are ordered according to their rank. However, this method requires knowledge of the key used during encryption in order to know the values of the classes, which is only applicable in the case of the training and test sets, not for the attack set. Therefore, other ways to order the predictions are needed that are not based on knowledge of the intermediate variable. We now examine potential methods to study the network's confidence in its predictions so we can order the predictions in order of decreasing confidence.

## 4.1 Some methods to distinguish the quality of the predictions

The first tests we conducted were based on some special statistical values computed for each prediction in the set considered. The following analysis concentrates on predictions resulting from the use of a *Softmax* activation function at the end of a neural network. This function forces the values of the predictions to be between 0 and 1 and to sum up to 1. The level of confidence of the network is then analyzed from the distribution of the values in the prediction. However, it is important to distinguish between confidence and performance, as a model can be highly confident in its prediction but can still be making a mistake. Nevertheless, the confidence of the network remains useful to distinguish between well-predicted traces and the others.

**The median** The median of a distribution is the value that separates the distribution into two parts each containing the same number of elements. The first part contains all the values that are above the median, and the second part, all the values below the median. In this case, when considering attacking a byte of the key under the identity leakage model, there are 256 values in the predictions and the median separates the 128 highest values from the 128 lowest ones. The median is one way to distinguish the correctly predicted predictions from the others. Indeed, it is possible to link this value to the confidence the network has in its prediction. The more confident the network, the more likely it will attribute higher values to the classes it considers most likely and, as a consequence, the lower the values of the other classes will be. This is due to the use of the Softmax function that normalizes the values of each prediction. Concerning the impact on the median, if the network gives a high value to a small number of classes, the median will tend to be low. On the other hand, if the network is not sure what to predict, it will assign close values to all the classes which lead to a higher median. Based on those facts, it is possible to order the prediction from the one with the lowest median to the one with the highest. The median acts as an indicator of the network's confidence and can be used to decide which predictions will be the best to use during the attack.

**The minimum** The minimum value of each prediction is also an interesting value to explore. Its connection to the network's confidence is less clear but still exists. Indeed, if a network associates a high value with the class it considers most likely, this will drive the values of all the other classes down and consequently, the minimum of the prediction will also be lowered. Depending on the value of the minimum, it will then be possible to order the predictions, from lowest to highest to represent the network's confidence. The maximum can be used in a similar way, but in this case, the predictions must be sorted from highest maximum value to the lowest maximum value. It makes sense that the more confidence a network has in a class, the higher the values it attributes will be. However, in the following tests, even though the maximum produced good results, it was never as good as the minimum and consequently, in the experiments, the focus was on the minimum.

The use of the different distinguishers is illustrated in Figure 6 which shows the processing of three traces by the network and the different orders attributed depending on the value used to evaluate the network's confidence. Two ways of ordering the predictions were tested. In the first, the predictions were ordered in ascending order depending on the value of the distinguisher, as described above. In the second, the predictions were sorted in descending order of the values of the distinguisher. Both ways are described in Figure 6.

**Other indicators** Other values that could serve as indicators of the confidence of the network were studied including variance, entropy, the distance between the most probable classes and the distance between the maximum and the minimum, to name a few of the most important ones. Details are not given on each as they all showed some potential but were not as efficient as the median or the minimum.

The following section presents the empirical results of orderings computed using the median and the minimum.

## 4.2 Experimental results on neural networks

As mentioned in the introduction, the goal of this work is to study the effect of the new attack method in a context where access to new traces and the training and tuning of the models are limited. This means that the model is trained using a fixed number of traces for a fixed number of epochs and the resulting network is used for all the attacks. This context explores in particular a situation where the attacker has limited access to attack traces but still tries to improve the results of the attack.

### 4.2.1 Description of the databases and the neural network used

**Databases** The experimental results presented in this work were mostly obtained using the ASCAD[1] [BPS+19] database to train and test the networks. This database was created by the French cybersecurity agency (ANSSI) with the aim of giving researchers in this field a common starting point to train different architectures and compare their results. The experiments reported in this section were based on the fixed key database, called ASCAD fixed key, for which all the traces were acquired using the same encryption key. This set is composed of 60,000 power consumption traces obtained with a AVR ATMega8515 8-bit microcontroller running an AES algorithm secured against first order attacks, i.e. including a first order masking countermeasure.

The traces were split into two subsets: the first one contains 50,000 traces and was used to train the networks and the second one, with the 10,000 remaining traces, was used as a testing set. The traces are composed of 700 points representing power consumption over time. The points were chosen from the original 100,000 points of the raw traces as they contain leaking information on the third byte of the output of the first masked SubByte operation and on the value of the corresponding mask. The leakage model used was the identity model and the values of the labels $Y$ associated with each trace were obtained as follows:

$$Y(\mathbf{k}^*) = \mathit{SubBytes}(\mathbf{p}[3] \oplus \mathbf{k}^*[3]),$$

where $p$ represents the input plaintext of the algorithm and $k^*$ is the encryption key. Therefore the output of the neural networks is a probability distribution over the 256 possible values of the output of the unmasked SubByte operation.

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

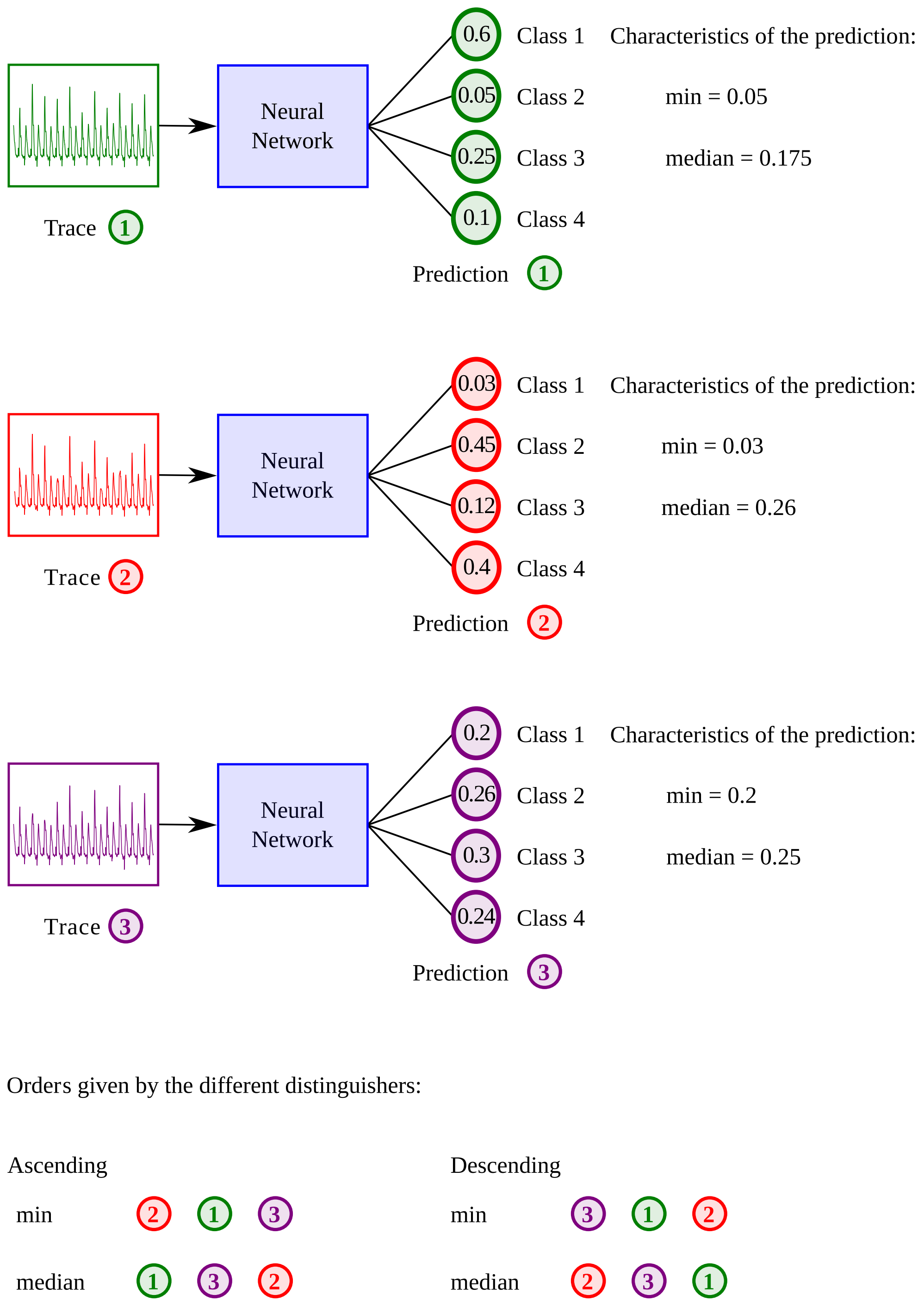


Figure 6: Example of the use of distinguishers (minimum or median) to order predictions of the different classes.

The experiments were also performed on a variant of this dataset, called ASCAD variable key, for which the training traces were acquired using random encryption keys. This dataset is composed of 200, 000 traces for the training and 100, 000 traces for testing and the traces are composed of 1, 400 points.

Furthermore, the databases each have 3 variants depending on the amount of desynchronization in the traces. Desynchronization is a common countermeasure present in the traces that aims to decorrelate the operations of the encryption algorithm from specific time points. This results in power consumption traces in which the same point does not correspond to the same operation and therefore does not contain the same information. The model of desynchronization considered here is a temporal shift. It means that all the points of the traces are shifted by a random amount. The variants are called Desync0, Desync50 and Desync100 where the number represents the maximum shift possible for the traces of those sets.

**Network** The network used for the experiments is the convolutional neural network $\text{CNN}_{\text{best}}$ introduced in the work setting up the ASCAD database [BPS+19]. $\text{CNN}_{\text{best}}$ is the best result obtained from a hyperparameter search. It is composed of 5 convolution layers followed by two fully-connected layers, all using the ReLU activation function. The convolution filters are of size 11 and their quantity doubles after each layer, starting from 64 and reaches a maximum of 512. Average pooling was used after each convolutional layer to reduce the increase in the dimension of the data. Training was done with the Categorical Cross-Entropy loss function and the RMSprop optimizer with a learning rate of $10^{-5}$. The network was trained on the dataset Desync0 for 75 epochs and then evaluated on the test sets of the different variants.

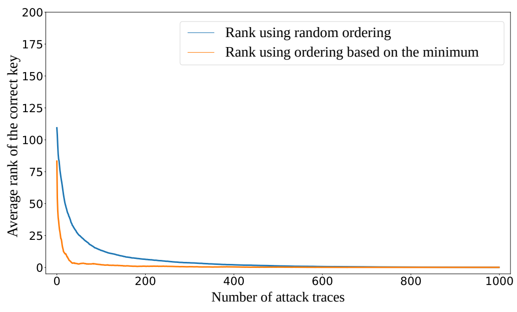
### 4.2.2 Experiments using the ASCAD fixed key

The $\text{CNN}_{\text{best}}$ neural network was used to start studying the behavior of the attacks by targeting the database ASCAD. The curves included in this section represent the evolution of the mean rank of the correct key computed from 100 attacks using random traces selected among the 10, 000 traces that comprise the test set. The goal here is to compare the evolution of the mean rank when the order of the traces is random or when the order is based on one of the indicators discussed in the previous section.

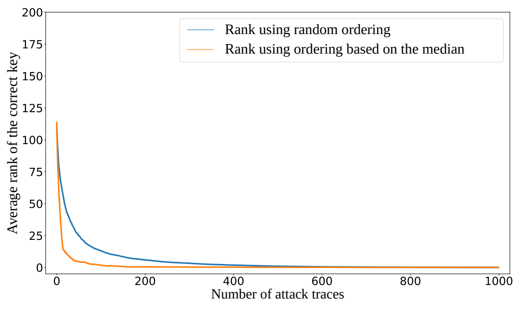
**Desync0** Figure 7 represents the mean rank of attacks using the minimum and the median as indicators and an **ascending** order. The attacks were performed on synchronized traces from Desync0, which explains why they all succeeded using less than 1, 000 traces. As explained previously, in this case, using ordering when the rank already converges to 1 does not lead to improvement. This is the worst case mentioned in Section 3.3 but it had no impact on the result since the attack was successful. However, the effect of the order is still noticeable as the curves representing the uses of the minimum and the median converge faster than the curves representing random ordering.

**Desync50** The following tests were conducted using a more difficult dataset for the network, the variant Desync50. The network, which was trained on synchronized traces, was applied to desynchronized traces the network had never seen before, thus making the attacks more difficult for the network.

The tests on Desync50 were done using 5, 000 traces for the attacks, in order to observe how the behavior changed when more traces were used. Figure 8 shows the evolution of the average rank of attacks using random ordering or **descending** ordering based on the median and the minimum. As previously, the use of the median produced slightly better results than the minimum with a rank of 1 reached after a few thousands traces while the final rank was 3. This suggests that the correct key can be retrieved directly using this attack method, unlike using the classical method, as long as one is capable of choosing the correct number of traces to use.

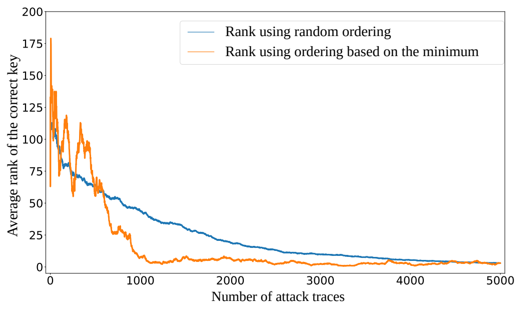


(a) Minimum

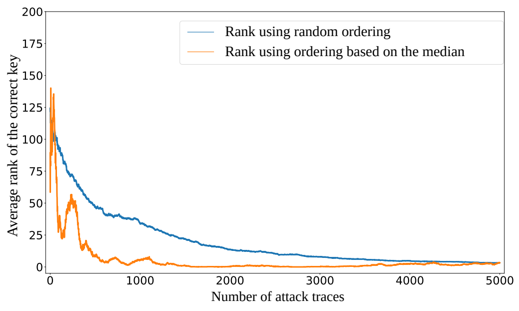


(b) Median

Figure 7: Evolution of the average rank of the correct key for attacks using random ordering and **ascending** ordering based on a distinguisher (minimum and median) of the predictions of the network $\mathrm{CNN}_{\mathrm{best}}$ on the database ASCAD fixed key and the Desync0 dataset.



(a) Minimum



(b) Median

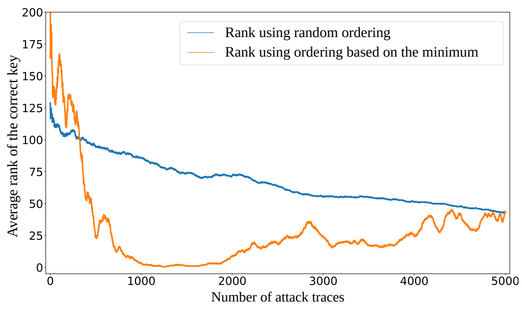
Figure 8: Evolution of the average rank of the correct key for attacks using random ordering and **descending** ordering based on a distinguisher (minimum and median) of the predictions of the network $\mathrm{CNN}_{\mathrm{best}}$ on the database ASCAD fixed key and the Desync50 dataset.

The following tests were conducted on the variant Desync100 to further increase the difficulty of the attacks.

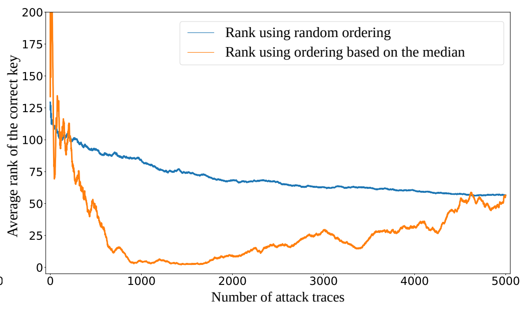
**Desync100** Like the behavior observed with the variant Desync50, the **descending** ordering had to be used to obtain the best results. Figure 9 shows the results of attacks using $5,000$ traces. As previously, the evolution of curves of the descending ordered attacks began by converging toward the value 1 before reaching the minimum rank and then increased toward the final rank. The minimum for the average rank attained using a **descending** ordering based on the minimum was 2 and the one based on the median was 3. The correct key byte was thus almost retrieved, whereas the final rank was on average 50 which is significantly worse. The following section continues with a description of the experiments conducted using a different dataset ASCAD variable key.

### 4.2.3 ASCAD variable key

The following experiments were performed using the dataset ASCAD variable key. The network was the same one as before, $\mathrm{CNN}_{\mathrm{best}}$, and it was still trained on $50,000$ synchronized traces for 75 epochs. The curves in the figures represent the evolution of the average rank computed from 100 attacks using random traces taken from the

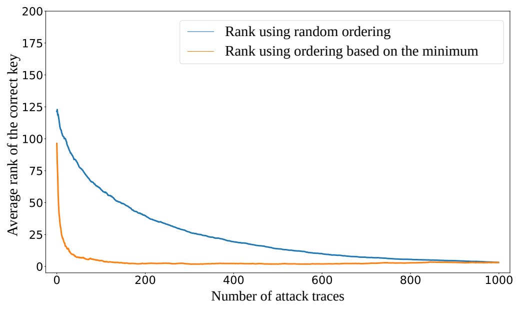


(a) Minimum

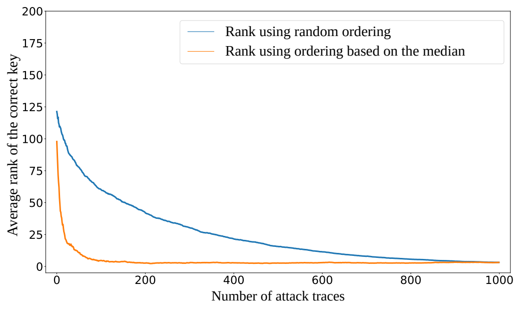


(b) Median

Figure 9: Evolution of the average rank of the correct key for attacks using random ordering and **descending** ordering based on a distinguisher (minimum and median) of the predictions of the network $\text{CNN}_{\text{best}}$ using the database ASCAD fixed key and the Desync50 dataset.



(a) Minimum



(b) Median

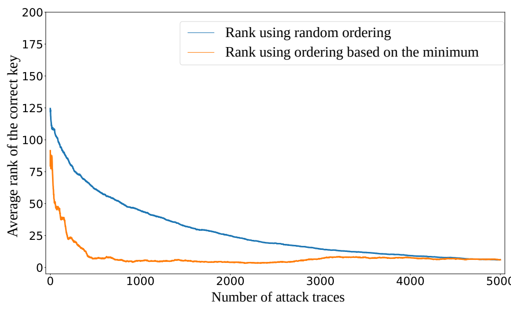
Figure 10: Evolution of the average rank of the correct key for attacks using $1,000$ traces, random ordering and **ascending** ordering based on a distinguisher (minimum and median) of the predictions of the network $\text{CNN}_{\text{best}}$ using the database ASCAD variable key and the Desync0 dataset.

$100,000$ attack traces. In contrast to the last results of the previous section, all the following attacks were performed using **ascending** ordering.

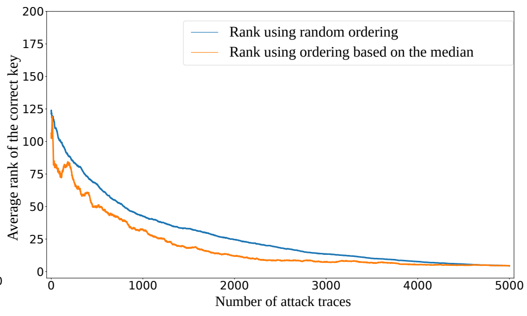
**Desync0** Figure 10 shows the evolution of the average value of the rank of the correct key for attacks on Desync0 and ordering based on the minimum and the median and **ascending** order. Figures 10a and 10b show that the minimum ranks only resulted in a slight improvement in the final rank, with respectively minimum and median order minimum ranks of 2.5 and 3, instead of 4 for the final rank. However, once again, it appears that this new method of performing the attacks makes the average ranks converge faster, thereby confirming the correct ordering of the predictions.

These results are interesting but do not provide much valuable information since the network is able to perform the attacks with little more than $1,000$ traces. Again, the Desync50 and Desync100 datasets can be used to observe the impact of the ordering more directly.

**Desync50** Figure 11 shows the evolution of the average value of the rank of the correct key for attacks using $5,000$ traces on Desync50 and either **ascending** ordering based on the minimum and the median or a random ordering. The minimum rank reached when the minimum was used for the ordering was 4.5 while the final rank

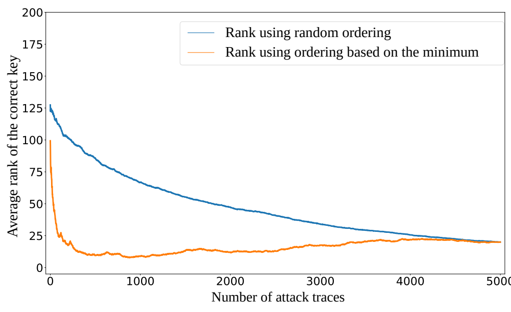


(a) Minimum

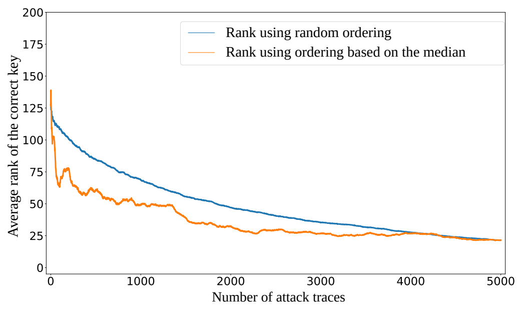


(b) Median

Figure 11: Evolution of the average rank of the correct key for attacks using $5,000$ traces, random ordering and **ascending** ordering based on a distinguisher (minimum and median) of the predictions of the network $\text{CNN}_{\text{best}}$ using the database ASCAD variable key and the Desync50 dataset.



(a) Minimum



(b) Median

Figure 12: Evolution of the average rank of the correct key for attacks using $5,000$ traces, random ordering and **descending** ordering based on a distinguisher (minimum and median) of the predictions of the network $\text{CNN}_{\text{best}}$ on the database ASCAD variable key and the Desync100 dataset.

was 7. However once again, the median did not result in any real improvement over the normal attack apart from slightly faster convergence.

Like with Desync0, only ordering based on the minimum of the predictions led to better results of the attack by reducing the rank of the correct key. The final tests were done on Desync100 which is a more difficult dataset for the network than Desync50.

**Desync100** The results obtained using this dataset are shown on Figure 12 for attacks using $5,000$ traces. The improvements obtained using these orders can be seen in Figures 12a and 12b. In this case, the reduction of the minimal rank thanks to ordering based on the minimum was bigger since it reached 9, down from a final rank of 22. The number of ranks gained led to a 50% reduction in the value of the final rank of the correct key.

This first set of experiments revealed interesting behaviors resulting from the new way of performing the attacks. It is indeed possible to consistently reach values of minimum ranks lower than the final rank, i.e. the rank reached when all the attack traces are used. However, these tests have the disadvantage of using the network $\text{CNN}_{\text{best}}$ applied to datasets it was not trained on, in order to increase the difficulty of the attacks. Although it reveals the effects of ordering the predictions, it is not the ideal context. Consequently, the next set of experiments used the classical templates attack and this time, the templates were created using different numbers of traces to determine the effects and improvements enabled by this new attack method in this context.

## 4.3 Experimental results of the template attacks

In this section, a ChipWhisperer (XMEGA 8-bit) running an AES 128-bit was used to acquire traces of power consumption. The traces were then centered on $1,000$ points including the first round of the AES. The profiling set was composed of $50,000$ traces using random keys while the attack set was composed of $1,000$ traces with a fixed key. The target byte for the attacks was the 16th byte giving an intermediate value of:

$$Y(\mathbf{k}^*) = SubBytes(\mathbf{p}[16] \oplus \mathbf{k}^*[16]),$$

where $\mathbf{p}$ is the plaintext and $\mathbf{k}^*$ is the key used. As this AES implementation is not protected it has high leakage rates, making it easy to recover the key. To increase the difficulty of the attacks, a noise was added to the power consumption traces to reduce the correlation between the intermediate value and the points of the traces. The noise followed a Gaussian distribution $\mathcal{N}(0, 0.05)$ and each point of the traces had a different noise value. Different numbers of traces in the construction of the templates were used in the experiment to observe how the ordered attacks behave in each case. The templates were created by using the 5 points with the highest SNR.

To reproduce the context of the attacks using neural networks, the templates were not applied on the fly during the attacks but in a preliminary phase. The templates of each possible intermediate value were applied to each attack trace to obtain a set of predictions similar to the predictions of a neural network. The attacks were performed using the predictions that were also used for the determination of the orderings. The results shown in the next figures are averages of 100 attacks made using 500 traces from the attack set.

**Template built using $10,000$ traces** Figure 13 shows the results of the attacks made using the template built with $10,000$ traces. The ordered attacks used **ascending** ordering based on the minimum of the predictions of the 500 attack traces. The average of the minimum rank obtained during the attacks was 47 when the final rank was 77. Although the ordering improved the attack, the curve showing the evolution of the average rank is less smooth than when the neural networks were used, indicating that even though ordering based on the minimum produces better results, this ordering is not optimal and a better ordering is possible.

**Template built using $20,000$ traces** Figure 14 shows the results of the attacks performed with a template made of $20,000$ traces and represents the evolutions of the average rank of the correct key when random ordering and descending ordering based on the minimum were used. Thanks to **descending** ordering, the minimum of the rank reached was 10 while the final rank was 26. This is a significant improvement, but the shape of the curve shows the order of the traces is still not optimal. Indeed, the evolution of the rank alternates increasing and decreasing phases. It is still able to reach a minimum lower than the final rank but could be improved even more by finding a better ordering.

**Template built using $50,000$ traces** Figure 15 shows the evolutions of the rank for the different attack methods. **Descending** ordering produced an improvement by lowering the final rank from 14 to a minimum of 7. However, the minimum rank was reached using 480 traces, i.e. almost all the attack traces. Once again, the new attack method produced better results but showed signs that it could be improved even further.

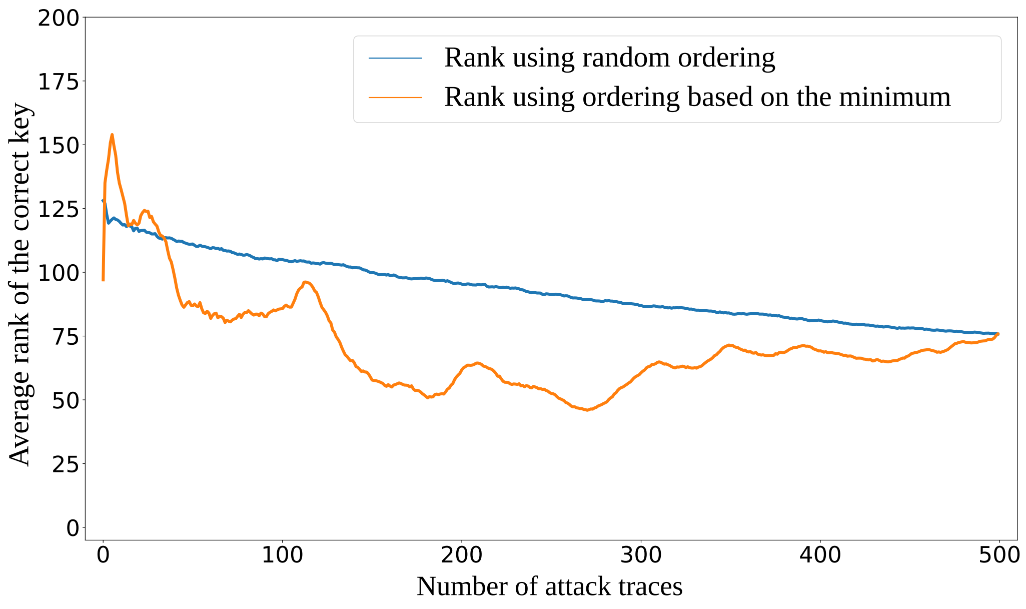


Figure 13: Evolution of the average rank of the correct key for template attacks using random ordering and **ascending** ordering based on the minimum of the predictions from a template created using $10,000$ traces.

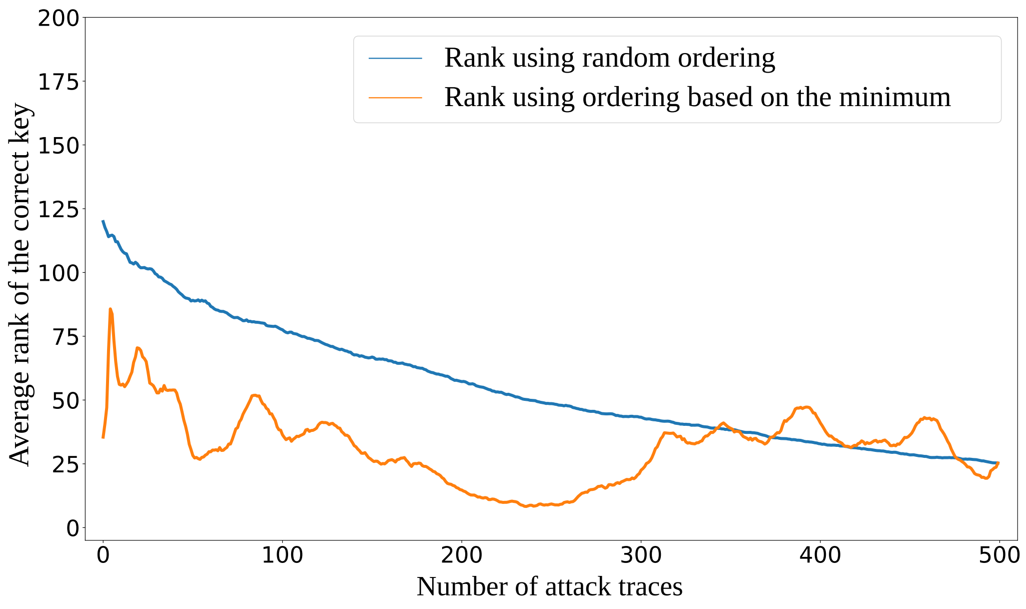


Figure 14: Evolution of the average rank of the correct key for template attacks using random ordering and **descending** ordering based on the minimum of the predictions from a template created using $20,000$ traces.

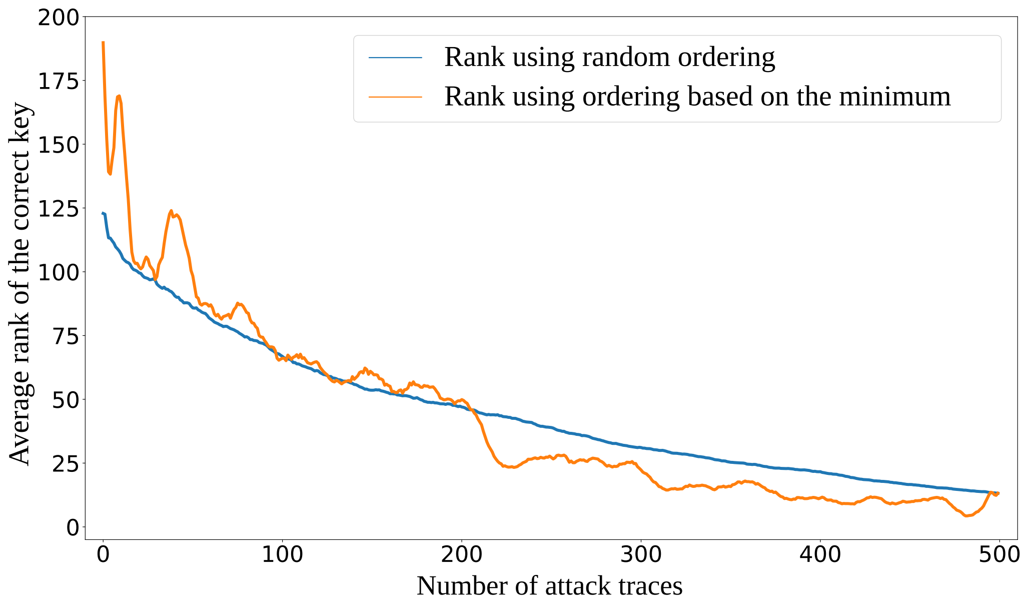


Figure 15: Evolution of the average rank of the correct key for template attacks using random ordering and **descending** ordering based on the minimum of the predictions from a template created using $50,000$ traces.

**Interpretation of the results** The results obtained with the templates are more complex to interpret. Unlike the attacks based on the neural networks for which a fixed architecture was used, the templates created for the attacks are more varied. The number of traces used to generate them and the number of points of interest considered can greatly affect the results of the attack. Another important point is that the values the predictions can take are not bounded by a fixed interval. Therefore, the minimum value of some predictions may turn out to be higher than all the values of other predictions. It is thus less effective to use the minimum as a distinguisher to order the predictions. Nevertheless, the minimum is still the value that produces the most consistent results in the experiments conducted and still produces interesting results

## 4.4 Conclusion of the first experiments

The results of the experiments reported in this section on attacks based on neural networks and templates reveal that the new attack method based on the ordering of the predictions can significantly improve the results of the attacks. However, these experiments also show that the best possible order was never achieved. The different distinguishers based on the value of the minimum and the median of the predictions did improve the final rank but the instability of the results indicates they are not optimal distinguishers. Consequently, the following section concentrates on a search for a better way to order the predictions. To this end, it focuses on a set of machine learning techniques that explores the comparison and ordering of examples and the attribution of scores. More specifically, a neural network is trained to learn to classify the predictions and to create orders.

## 5 Using neural networks to solve the ordering problem

Even though the results based on the distinguishing values were promising, it was not possible to clearly identify any one indicator that produces better results than the others. We thus turned to deep learning to find a solution and to solve the task of ordering the predictions. This was done by training a network to assign a score to each pair (trace, prediction) in the attack set. The goal is to have a score that indicates both the quality of a trace and its prediction.

In contrast to the use of neural networks in DLSCA, the problem here was not to classify the traces but to order their predictions. To this end, a score was attributed to each prediction and the predictions were then sorted according to their scores. The distinguishers used previously managed to partially solve this problem but has certain limitations as described in the previous experimental section. Training specific neural networks to solve the task of ordering the predictions was thus explored. This use of neural networks is part of the *Learning to rank* approach that is already widely studied in machine learning [BSR+05, BRL06, CLL+09].

### 5.1 The Learning to rank approach

The *Learning to rank* approach in machine learning includes techniques that allow the creation of models whose purpose is to sort data. These models are used in a wide variety of problems ranging from recommendation systems [LMK+11, XJP+10, CAS16] to information retrieval [WBSG10] or natural language processing [Li14].

A ranking problem can be formulated as follows:

**Problem** Let $\mathcal{Q}$ be a set of queries, $\mathcal{E}$ a set of entries and $\mathcal{C}$ a ranking of the entries of $\mathcal{E}$ depending on the requests of $\mathcal{Q}$: train a model $F_\theta(q, e)$ of parameters $\theta$ able to rank an entry $e \in \mathcal{E}$ based on a request $q \in \mathcal{Q}$.

The *Learning to rank* approach is mostly viewed as a supervised training problem where the ranking of the training data corresponds to the training labels. The labels show the relevance of the entries compared to the relevance the other entries. Most often, the model gives a score to each entry that can then be used to determine its relevance for a given query.

Below is a description of the whole attack problem in the form of a *Learning to rank* problem. Let $\mathcal{T}_{train}$ be the training set, $\mathcal{T}_{val}$ the validation set and $\mathcal{T}_{att}$ the attack set, respectively. The first step of the attack consists in training a model $F_\theta(t)$ that, given a trace $t$, is capable of outputting a prediction on the value of the intermediate variable used in the encryption. The model is trained using $\mathcal{T}_{train}$ and $\mathcal{T}_{val}$, used for validation purposes, to be efficient on $\mathcal{T}_{att}$. The sets $\mathcal{P}_{\mathcal{T}_{train}}$, $\mathcal{P}_{\mathcal{T}_{val}}$ and $\mathcal{P}_{\mathcal{T}_{att}}$ correspond to the sets of predictions made by the model $F_\theta$ concerning traces of the training, validation and attack sets respectively.

The goal of the *Learning to rank* approach is to train a second model $F'_{\theta'}(p)$ of parameters $\theta'$ that will give a score to each prediction $p \in \mathcal{P}$ according to which prediction is most useful for the attack. Using the predictions from $\mathcal{P}_{\mathcal{T}_{train}}$ and $\mathcal{P}_{\mathcal{T}_{val}}$, this new model learns to rank the predictions $\mathcal{P}_{\mathcal{T}_{att}}$ so that it is possible to order the predictions in a way that is better for the attack than a random order.

The choice was made to focus on the Pairwise approach to solve the ranking problem and to train the model. The Pairwise approach is defined as follows:

- the *Pairwise approach* [BSR+05, WBSG10] consists in giving the model pairs of examples $(p_i, p_j)$ for which it has to predict an order $F'_{\theta'}(p_i, p_j) = o_{p_i,p_j}$. A limit value $l$ is set so that if $o_{p_i,p_j} > l$, prediction $p_i$ is considered to be more important than prediction $p_j$ and this relation is denoted $p_i \triangleright p_j$. Using this method, it is possible to compare pairs of predictions to order the predictions of the attack set. However, it is also possible to use the other Learning to rank approaches by adapting the training method.

Two other approaches are possible: the Pointwise and Listwise approach [CQL+07, XLW+08]. They are discussed in more length in [ZBD+21]. The Listwise approach seems to be most appropriate as it takes all the examples into consideration at the same time, which is the closest methodology to side-channel analysis. However, its main drawback is the complexity of its application, which is why we decided to focus on the Pairwise approach. The following section describes the neural network used to solve the *Learning to rank* problem.

**Using a siamese network to solve the *Learning to rank* problem** The use of the Pairwise approach makes it possible to use siamese networks to learn how to rank the predictions. A siamese network is made out of an architecture that is duplicated to form two identical networks that share their weights and make parallel predictions using different examples [BBB+93]. The results of the predictions are then compared to produce the output of the siamese network. Siamese networks are often used to solve tasks including object tracking [BVH+16, GFZ+17] and *Learning to rank* problems [GSC+19].

Figure 16 depicts a whole siamese network. The two networks that share their architecture and weights are shown in green. Each network takes a prediction as an input and gives it a score. The scores are then compared and the result of the comparison is used to compute the loss function. Afterwards, the weights of each network are updated in the same way to keep the symmetry between the two networks. In this application, one of the networks is kept at the end of the training to be used on the attack prediction set to give a score to each $p \in \mathcal{P}_{\mathcal{T}_{att}}$ and hence order the predictions for the attack.

The Pairwise approach is used to train the siamese network where pairs of predictions are chosen randomly from the validation set $\mathcal{P}_{\mathcal{T}_{val}}$ and given to the network. Finally, to properly train the network, labels have to be defined for each pair. The following section introduces and discusses several ways to choose the labels and argues for the label that seems to be the best suited for this application.

## 5.2 Adaptation of the ranking loss

The *ranking loss* was introduced by Zaid *et al.* [ZBD+21] as a new loss function dedicated to the training of neural networks used in side-channel analysis. In their article, Zaid *et al.* try to answer some of the criticisms of the Categorical Cross Entropy, concerning the fact that this loss function focuses on the value associated with the correct label, by accounting for the position of the correct label among the others labels. Starting from a *Learning to rank* problem, they create a loss function that considers the position of the correct label in its computations and penalizes the network more heavily when the label is not correctly placed. We first describe this loss function before exploring how it can be adapted to the problematic at hand.

### 5.2.1 Description of the ranking loss

Let $c_{k^*}$ and $c_k$ be the classes associated with $k^*$ the good key and $k$ a key hypothesis, the probability of the relation $c_{k^*} \triangleright c_k$ establishing that $c_{k^*}$ must be ranked higher than $c_k$ is given by:

$$Pr(c_{k^*} \triangleright c_k) = \frac{1}{1 + e^{-\alpha(s_{N_a}(k^*) - s_{N_a}(k))}}, \tag{1}$$

where $s_{N_a}(k)$ represents the score outputted by the network for key $k$ and $\alpha$ is a hyperparameter to tune. The logistic function is used in this computation because the goal is to try to approximate the indicator function [QLL10].

This leads to the definition of the partial loss function:

$$l_{N_a}(c_{k^*}, c_k) = -\bar{P}_{k^*,k} \cdot \log_2(P_{k^*,k}) - (1 - \bar{P}_{k^*,k}) \cdot \log_2(1 - P_{k^*,k}), \tag{2}$$

with $P_{k^*,k} = Pr(c_{k^*} \triangleright c_k)$ and $\bar{P}_{k^*,k} = \frac{1}{2}(1 + \text{rel}_{k^*,k})$ where $\text{rel}_{k^*,k} \in \{-1, 0, 1\}$. This definition is derived from the work of Burges *et al.* [BRL06]. It can be transformed

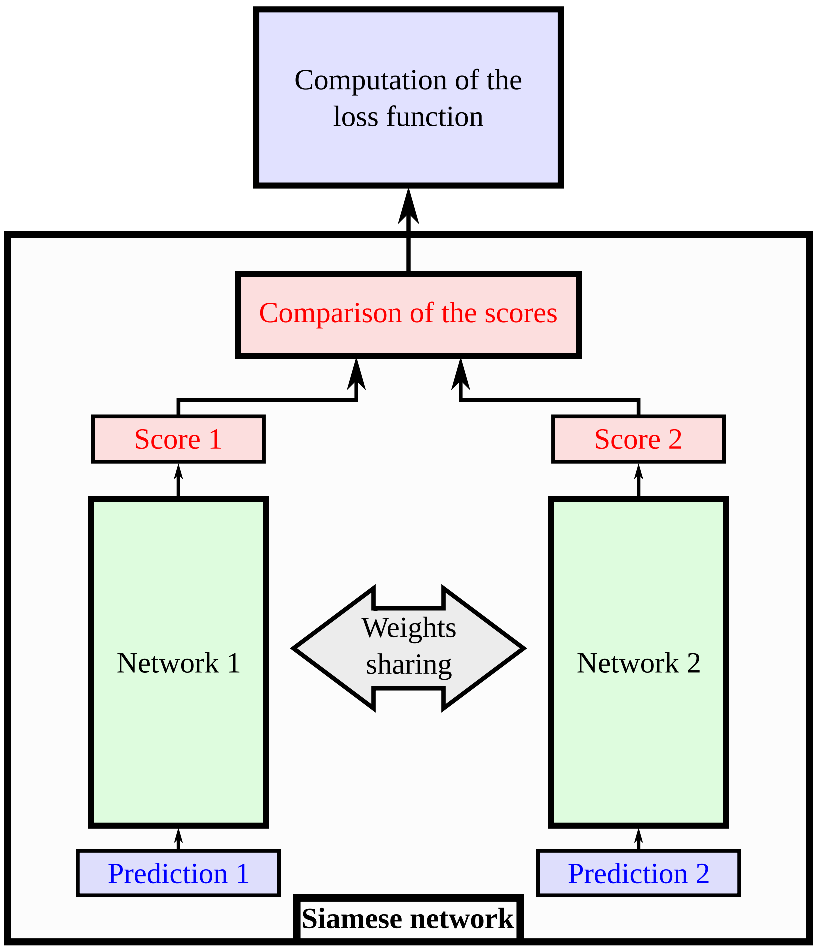


Figure 16: Description of a siamese network. The two networks composing it share their weights and each receives a different input. The scores they produce are compared and used in the loss function.

into $\bar{P}_{k^*,k} = 1$ given that the key $k^*$ should always be ranked above every other key hypothesis, so $\text{rel}_{k^*,k} = 1$ and:

$$l_{N_a}(c_{k^*}, c_k) = \log_2\Big(1 + e^{-\alpha(s_{N_a}(k^*) - s_{N_a}(k))}\Big). \tag{3}$$

From there it is possible to define the *Ranking Loss* as follows:

$$\mathcal{L}_{RkL}(F_\theta, \mathcal{T}, N_a) = \sum_{\substack{k \in \mathcal{K} \\ k \neq k^*}} \Big(\log_2\Big(1 + e^{-\alpha(s_{N_a}(k^*) - s_{N_a}(k))}\Big)\Big). \tag{4}$$

### 5.2.2 Adaptation to the problem of ordering the predictions

The first step is redefining the ranking problem to apply it the scoring problem. The relation between classes $c_{k^*} \rhd c_k$ is no longer important, instead the relation $p_i \rhd p_j$ has to be taken into consideration as it represents the fact that prediction $p_i$ is more important for the attack than prediction $p_j$. To this end, the probability $Pr(c_{k^*} \rhd c_k)$ needs to be redefined.

In the ranking loss, the difference $(s_{N_a}(k^*) - s_{N_a}(k))$ is used to represent the problem of ranking the correct key with respect to each prediction and this difference makes it possible to approximate the indicator function. To adapt it to the problem of ordering the predictions, the difference in Equation 1 can be replaced by $(s_i - s_j)$, where $s_i$ and $s_j$ represent the scores of the predictions $p_i$ and $p_j$ given by the siamese network. The value $(s_i - s_j)$ corresponds to the difference that must made between the ranking of the two predictions according to the siamese network.

These modifications lead to the following definition of the probability of the relation $p_i \rhd p_j$:

$$Pr(p_i \rhd p_j) = \frac{1}{1 + e^{-\alpha \cdot (s_i - s_j)}}. \tag{5}$$

The binary cross-entropy loss function can then be used as there are only two possibilities: either prediction $p_i$ is more relevant than prediction $p_j$ or the reverse. The following the partial loss function is obtained:

$$l_{N_a}(p_i, p_j) = -\bar{P}_{p_i,p_j} \cdot \log_2(P_{p_i,p_j}) - (1 - \bar{P}_{p_i,p_j}) \cdot \log_2(1 - P_{p_i,p_j}), \tag{6}$$

with $P_{p_i,p_j} = Pr(p_i \rhd p_j)$ and $\bar{P}_{p_i,p_j} = \frac{1}{2}(1 + \text{rel}_{p_i,p_j})$ where $\text{rel}_{p_i,p_j} \in \{-1, 0, 1\}$. Contrary to the work of Zaid *et al.*, the value $\text{rel}_{p_i,p_j}$ is no longer fixed at 1 since, depending on the formation of the pairs, prediction $p_i$ can be less important than prediction $p_j$. Let $r_i$ and $r_j$ be the rank of the correct labels in predictions $p_i$ and $p_j$. To facilitate computation, here the ranks are computed by sorting the hypothesis in ascending order of probability. This means that the best rank becomes 256 and the worst 1. It follows:

- $\text{rel}_{p_i,p_j} = 1$ when $r_i > r_j$ ;
- $\text{rel}_{p_i,p_j} = 0$ when $r_i = r_j$ ;
- $\text{rel}_{p_i,p_j} = -1$ when $r_i < r_j$.

This requires distinguishing between the three cases and observing their respective impacts on the loss function.

**Case 1: $\text{rel}_{p_i,p_j} = 1$** This case is similar to the one of the ranking loss where $r_i > r_j$ and thus:

$$l_{N_a}(p_i, p_j) = \log_2\Big(1 + e^{-\alpha \cdot (s_i - s_j)}\Big). \tag{7}$$

There are three possibilities:

- $s_i > s_j$: the scores given by the siamese network are in accordance with the relation $\text{rel}_{p_i,p_j}$ and $l_{N_a}(p_i, p_j) \approx 0$ so the network is not penalized;

- $s_i = s_j$: the scores given by the siamese network are equal, which is not in accordance with the relation $\mathrm{rel}_{p_i,p_j}$. Therefore, $l_{N_a}(p_i, p_j) = 1$ so the network is slightly penalized;
- $s_i < s_j$: the scores given by the siamese network are in contradiction with the relation $\mathrm{rel}_{p_i,p_j}$, $l_{N_a}(p_i, p_j) > 1$ so the network is heavily penalized;

**Case 2: $\mathbf{rel}_{p_i,p_j} = 0$** In this case, the ordering of the two predictions should be similar so the network should only be penalized when the scores differ. $\bar{P}_{k^*,k} = \frac{1}{2}$ thus:

$$\begin{aligned}
l_{N_a}(p_i, p_j) &= -\frac{1}{2}\log_2(P_{p_i,p_j}) - \frac{1}{2}\log_2(1 - P_{p_i,p_j}) \\
&= -\frac{1}{2}\log_2\left(\frac{1}{1 + e^{-\alpha\cdot(s_i - s_j)}}\right) - \frac{1}{2}\log_2\left(1 - \frac{1}{1 + e^{-\alpha\cdot(s_i - s_j)}}\right) \\
&= -\frac{1}{2}\Big(-\log_2(1 + e^{-\alpha\cdot(s_i - s_j)}) \\
&\qquad\qquad + \log_2\Big(\frac{e^{-\alpha\cdot(s_i - s_j)}}{1 + e^{-\alpha\cdot(s_i - s_j)}}\Big)\Big) \\
&= -\frac{1}{2}\Big(-\log_2(1 + e^{-\alpha\cdot(s_i - s_j)}) \\
&\qquad\qquad + \log_2\big(e^{-\alpha\cdot(s_i - s_j)}\big) - \log_2\big(1 + e^{-\alpha\cdot(s_i - s_j)}\big)\Big) \\
&= -\frac{1}{2}\Big(-2\log_2(1 + e^{-\alpha\cdot(s_i - s_j)}) + \log_2\big(e^{-\alpha\cdot(s_i - s_j)}\big)\Big)
\end{aligned}$$

The value of $l_{N_a}(p_i, p_j)$ depends on the difference $(s_i - s_j)$:

- If $s_i > s_j$, then $l_{N_a}(p_i, p_j) > 1$. The network is penalized, since it predicted incorrect scores;
- If $s_i = s_j$, then $l_{N_a}(p_i, p_j) = 1$. The network is slightly penalized but much less than the other situations;
- If $s_i < s_j$, then $l_{N_a}(p_i, p_j) > 1$ and once again, the network is heavily penalized.

**Case 3: $\mathbf{rel}_{p_i,p_j} = -1$** This last case represents the pairs where the correct label of $p_i$ is ranked lower than the correct label of $p_j$, i.e. $r_i < r_j$. Therefore, when the network attributes a score $s_i$ greater than $s_j$, it has to be penalized. This time, $\bar{P}_{k^*,k} = 0$ thus:

$$\begin{aligned}
l_{N_a}(p_i, p_j) &= -\log_2(1 - P_{p_i,p_j}) \\
&= -\log_2\left(1 - \frac{1}{1 + e^{-\alpha\cdot(s_i - s_j)}}\right) \\
&= -\log_2\Big(\frac{e^{-\alpha\cdot(s_i - s_j)}}{1 + e^{-\alpha\cdot(s_i - s_j)}}\Big) \\
&= -\log_2\big(e^{-\alpha\cdot(s_i - s_j)}\big) + \log_2\big(1 + e^{-\alpha\cdot(s_i - s_j)}\big)
\end{aligned}$$

In this case, the possibilities are:

- If $s_i > s_j$, then $l_{N_a}(p_i, p_j) > 1$. The network makes a mistake by giving more importance to the prediction $p_i$, consequently, it has to be penalized;
- If $s_i = s_j$, then $l_{N_a}(p_i, p_j) = 1$. The network is slightly penalized;
- If $s_i < s_j$, then $l_{N_a}(p_i, p_j) \approx 0$. The network correctly predicted the order of the predictions and is thus not penalized.

### 5.2.3 Equivalence to the case 1

This section explains how to carefully modify the loss function to reduce to cases 2 and 3 to case 1. To do so, the term $(r_i - r_j)$ is added in the computation of the partial loss function. Equation 7 becomes:

$$l_{N_a}(p_i, p_j) = \log_2\Big(1 + e^{-\alpha\cdot(r_i - r_j)\cdot(s_i - s_j)}\Big). \tag{8}$$

In this way, the relation $\text{rel}_{p_i,p_j}$ is directly included in the computation of the loss function.

This addition also preserves the characteristics of the three cases mentioned above. The conditions for penalization of the network are the following:

- Either $(r_i - r_j) = 0$ or $(s_i - s_j) = 0$ hence $l_{N_a}(p_i, p_j) = 1$. The network is slightly penalized;
- Either $(r_i - r_j)\cdot(s_i - s_j) < 0$ hence $l_{N_a}(p_i, p_j) > 1$. This time the signs of the differences are opposed, the scores of the siamese network contradict the order provided by the ranks, so it is heavily penalized;
- Either $(r_i - r_j)\cdot(s_i - s_j) > 0$ hence $l_{N_a}(p_i, p_j) < 1$, the signs of the two terms are equal, indicating that the score predicted by the siamese network correctly orders the predictions so the network is not penalized.

**Remark** The case $(r_i - r_j) = (s_i - s_j) = 0$ is marginal, therefore the slight penalization that occurs even if the network is correct does not impact the training.

## 5.3 Conclusion

Now the partial loss function has been defined, it can be used to define the loss function linked to the scores: the *Scoring Loss*.

**Definition 1.** Let $F_\theta$ be a model with parameters $\theta$ and $\mathcal{P}$ be the set of predictions of the traces of $\mathcal{T}$ by this model, the loss function *Scoring Loss* is defined by:

$$\mathcal{L}_{ScL}(F_\theta, \mathcal{P}) = \sum_{\substack{p_i,p_j\in\mathcal{P}\\ i\neq j}} \Big(\log_2\Big(1 + e^{-\alpha\cdot(r_i - r_j)\cdot(s_i - s_j)}\Big)\Big), \tag{9}$$

where $r_i$ represents the rank of the correct label of the trace $i$ in prediction $p_i$ and $s_i$ is the score given to prediction $p_i$ by the siamese network.

The following section focuses on finding an appropriate architecture for the siamese network and training it using the Scoring Loss to see if it is able to find orders of the predictions that are useful for the attacks.

# 6 Experimental results using the Scoring Loss

Once again, the experiments presented in this section were performed in a fixed context where the adversary has limited access to the attack traces and the network training and architecture are fixed. The first step in testing the Scoring Loss is defining the architecture of the network which is then duplicated in the siamese network in order to obtain the two scores to compare. Once training is complete, one network is extracted and used to predict the scores of the predictions of the attack traces. The order used to sort the predictions using the score is always **descending**, as this order is specified in the design of the Scoring Loss.

**Description of the attack method** Figure 17 describes the new way to perform attacks using the siamese network. The first steps up to the recovery of the prediction of the attack traces by the neural network are the same. The siamese network, which was trained using the predictions of the validation set, is then used to give a score to each prediction of the attack set. This makes it possible to order the predictions that can be used during the attack to improve the minimum rank of the correct key.

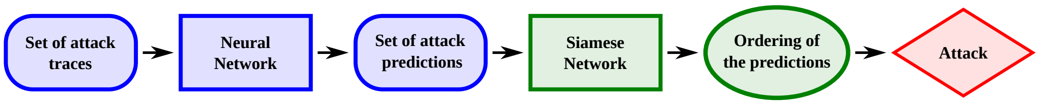


Figure 17: The steps of the new attack method using the siamese network to determine the order of the predictions.

**Description of the network** As the problem of scoring predictions is easier than predicting the intermediate value used during an AES encryption, the complexity of the network can be reduced. Moreover, the new network takes as input the predictions made by the initial convolutional neural network whose values are not deeply connected among themselves. Consequently, it is not necessary to use another convolutional neural network, which is why an MLP (MultiLayer Perceptron) composed of two layers of $1,024$ neurons is used in the siamese network. The layers used the activation function ReLU and the output of the network is a single neuron responsible for giving a score to the input prediction via a linear activation function. The full architecture of this network can be found in Table 3. The network uses the Adam optimizer [KB17] with a learning rate of 0.001. The Scoring Loss uses a base value of $\alpha = 0.1$ and the base number of epochs of the training is 10. The impact of the $\alpha$ value on the training is discussed by Zaid *et al.* in [ZBD+21].

**Description of the training** The training of the siamese network comprises the following steps. First, if there are enough, the predictions of the validation traces are used as the training set for the siamese network. If there are not enough, the predictions from the training traces can be used. So far, it is not clear which of the two sets, (i.e. the training or the validation set), is the best to use to train the siamese network. On the one hand, the predictions of the validation traces are closer to the predictions of the attack traces, which enables better generalization. On the other hand, using the predictions of the training traces allows the network to distinguish the good predictions from the bad ones as almost all the predictions from the training traces can be considered as good. This is because the network used to obtain the predictions is able to predict the training traces more accurately than the validation traces.

The second step is to create random pairs of predictions that are then used to train the siamese network. The number of pairs used will vary depending on the dataset and on the quantity of available predictions. The pairs are given to the siamese network and each prediction gets a score. The sign of the difference in the scores indicates which prediction is judged the best by the siamese network. Finally, the Scoring Loss penalizes the network depending on how it ordinates the predictions.

The following results were obtained using this methodology to train the siamese network and to order the attack predictions before the attacks.

## 6.1 ASCAD fixed key

The first experiments were performed using the dataset ASCAD fixed key with the $\text{CNN}_{\text{best}}$ neural network trained on the Desync0 dataset for 75 epochs. The curves presented in this section represent the evolution of the average rank of the correct key obtained during attacks using traces chosen randomly among the $10,000$ traces of the attack set. The attacks using $1,000$ traces were repeated 900 times to smooth out the curve of the average, and the attacks using 5000 traces were repeated 100 times.

**Desync0** Figure 18 shows the results of attacks against Desync0. These attacks used either an **ascending** order based on the minimum value of the predictions or an order based on the score of the predictions given by the siamese network. In this case, where the attacks were against synchronized traces, both orders produced similar results. The attacks were successful with fewer than $1,000$ traces which does not

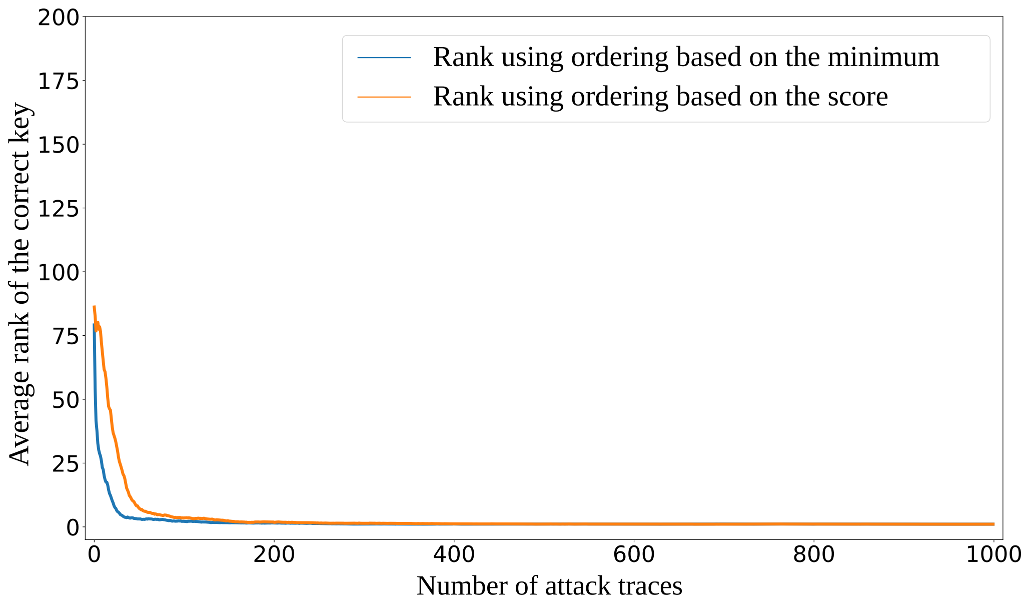


Figure 18: Evolution of the average rank of the correct key for attacks using a score order and an **ascending** order based on the minimum of the predictions of the $\text{CNN}_{\text{best}}$ network on the database ASCAD fixed key and the Desync0 dataset.

leave room for real improvement. However, it should be noted that the convergence of the two average ranks is similar, showing that in this case, the siamese network is able to come up with an order that is as good as the minimum.

**Desync50** The following results were obtained using attacks against the Desync50 dataset to increase the difficulty. The siamese network was the same as before, the only difference being that the value of $\alpha$ in the Scoring Loss is set to $\alpha = 0.001$ to guarantee efficient training.

Figure 19 shows the results of attacks in **ascending** order based on the minimum of the predictions and on an order based on the scores of the predictions for attacks using $5,000$ traces. The results show that ordering based on the scores ranks the attack predictions more efficiently. The final rank was on average 4.5 while the minimal average ranks were 1.8 and 1.6 when orders based on, respectively, the minimum and the score were used. This small difference can be explained by the fact that when $5,000$ traces were used, the attacks were almost successful, leaving little room for improvement. However, the evolution of the average rank using ordering based on the score converged much faster than using ordering based on the minimum, implying that it is better able to differentiate between correct and incorrect predictions.

**Desync100** The attacks discussed now were performed on the Desync100 dataset from the ASCAD fixed key database. The architecture of the siamese network remained the same but the number of training epochs became 5 and the value of $\alpha$ was lowered to 0.0001.

Figure 20 shows the results of the attacks against Desync100 using $5,000$ traces. The final rank value was 51 but the minimum rank reached using the orderings was 2. Looking at the shape of the curves, it appears that when ordering was based on the score, the curve representing the evolution of the average rank of the correct key converged faster toward a minimum value than when ordering was based on the minimum. This shows that ordering based on the score places the most interesting predictions in better positions than other ways of ordering.

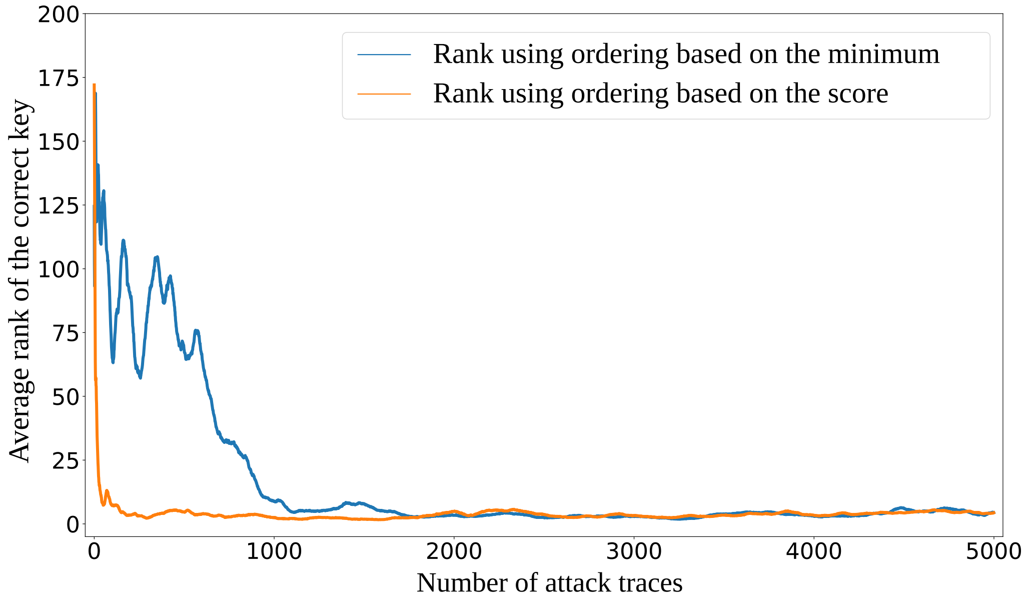


Figure 19: Evolution of the average rank of the correct key for attacks using a score order and a **descending** order based on the minimum of the predictions of the $\text{CNN}_{\text{best}}$ network on the database ASCAD fixed key and the Desync50 dataset.

In addition to the potential improvements obtained using this new attack method, these results confirm the interest of using neural networks to score the predictions. The results of the final experiments on ASCAD variable key are presented in the following section.

## 6.2 ASCAD variable key

Like the experiments described in Section 4.2, the following attacks were performed using the ASCAD variable key database and its subsets: Desync0, Desync50 and Desync100. The network $\text{CNN}_{\text{best}}$ was again trained using $50,000$ traces out of the $200,000$ traces in the training dataset while $150,000$ traces were kept for the validation set. This time, the siamese network was only trained using the predictions originating from the validation dataset; $100,000$ training pairs were generated from $50,000$ predictions of validation traces. The siamese network was trained for 10 epochs using a value $\alpha = 0.1$ for the loss function. Since the attack dataset was also larger, all the following curves were obtained by averaging the results of 900 attacks done with either $1,000$ or $5,000$ traces.

**Desync0** Figure 21 shows the evolution of the average ranks for attacks using either the minimum or the score to order the predictions. Like for ASCAD fixed key, the results of the attacks against Desync0 were very similar. The final rank was around 3 while the minima were around 2. Convergence was also similar for both orderings in this case.

**Desync50** The results of ordering of the predictions on Desync50 are shown in Figure 22. The evolution of the average rank using ordering based on the score reached a minimum value of 1.5 while the other minimum was 4.5 and the final rank was 7.5. This time too, ordering based on the score led to faster convergence of the curve and also to a lower minimum value for the rank. This once again shows that the score is able to distinguish the correct and incorrect predictions more efficiently than the minimum.

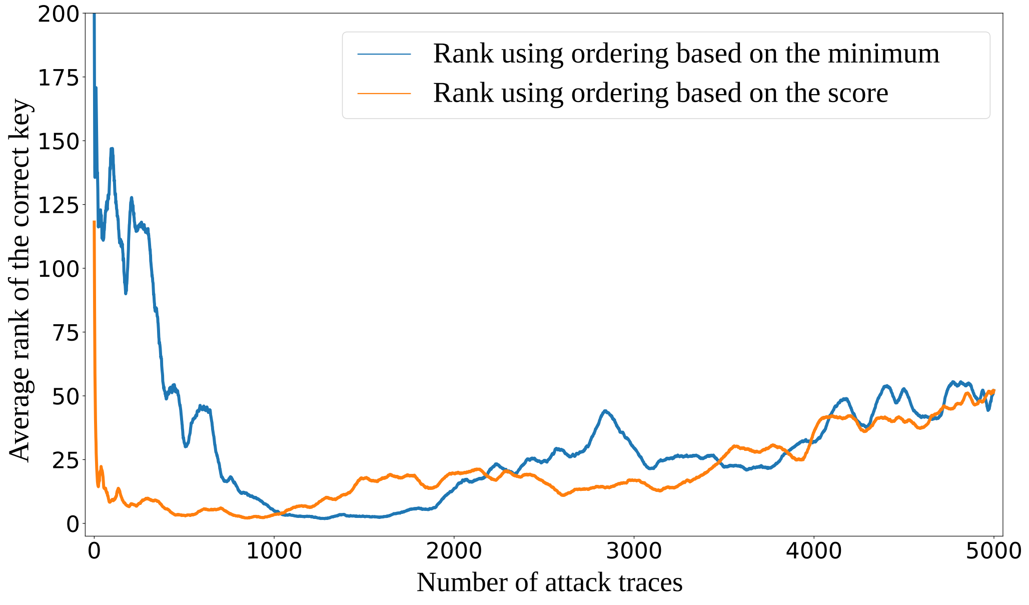


Figure 20: Evolution of the average rank of the correct key for attacks using a score order and a **descending** order based on the minimum of the predictions of the $\text{CNN}_{\text{best}}$ network on the database ASCAD fixed key and the Desync100 dataset.

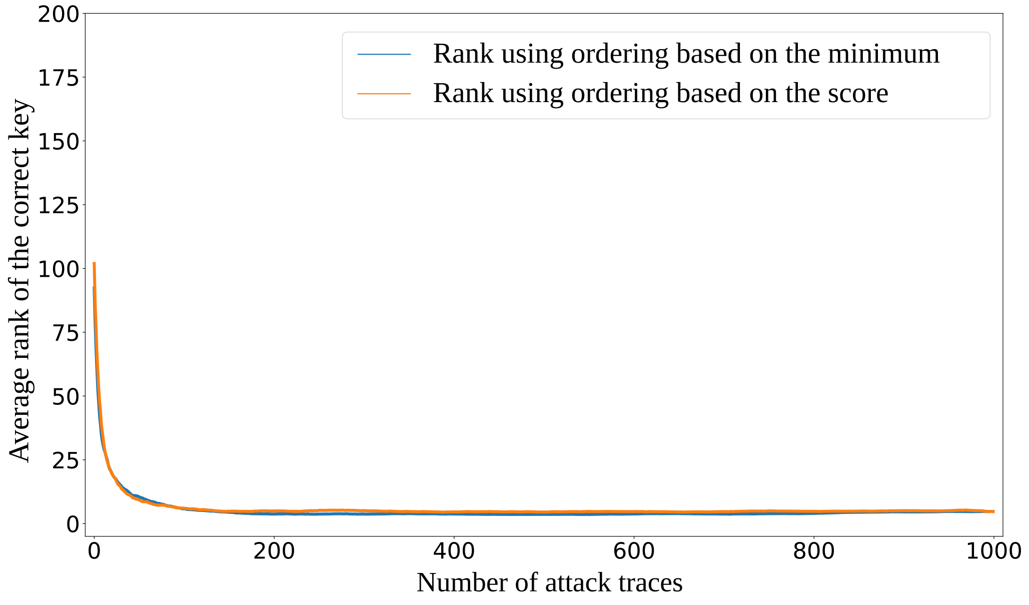


Figure 21: Evolution of the average rank of the correct key for attacks using a score order and an **ascending** order based on the minimum of the predictions of the $\text{CNN}_{\text{best}}$ network on the database ASCAD variable key and the Desync0 dataset.

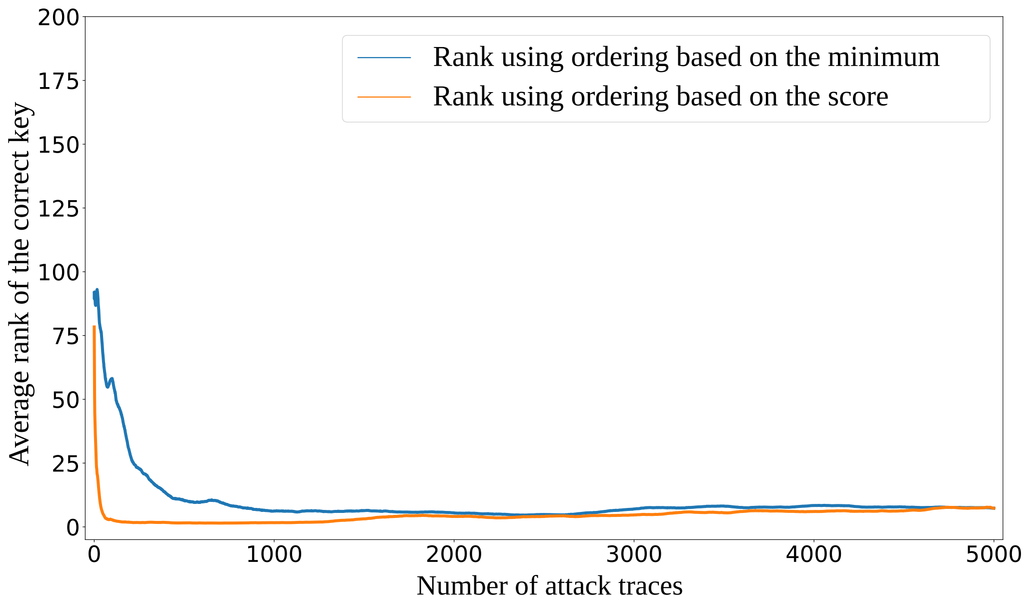


Figure 22: Evolution of the average rank of the correct key for attacks using a score order and a **descending** order based on the minimum of the predictions of the $\mathrm{CNN}_{\mathrm{best}}$ network on the database ASCAD variable key and the Desync50 dataset.

**Desync100** Figure 23 shows the evolution of the average ranks of the correct key for attacks against Desync100 when using ordering based on the minimum and the score. When $5,000$ traces were used for the attacks, the minimum ranks reached using ordering based on the minimum and the score were respectively 14 and 3, while the final rank was 25. The improvement made using the score was bigger than the improvement obtained with the Desync50 dataset and made it possible to reduce the average rank of the correct key by almost 90%.

## 6.3 Conclusion on the experiments

All the experiments performed on the ASCAD datasets showed a clear advantage of using the score of the predictions to order them. On the one hand, the scores given by the siamese network always produced better results when the **descending** order was used thanks to the design of the Scoring Loss. On the other hand, ordering based on the score led to even bigger improvements with the new attack method. We also want to mention that tests were done with other neural network architectures used in the literature such as the ones from [ZBHV20] but since the networks are already recovering the key in just a few traces, the new attack method did not bring any improvements.

Table 1 summarizes the results of the different tests done on both the ASCAD databases. The use of the score to order the predictions and to perform the attacks following the new method described in this article always improved the minimum rank when this was possible, i.e. when the attack was not already successful. In the best scenarios, it allowed the final rank to be reduced by around 90% and almost enabled recovery of the key.

Another advantage of using the score to order the predictions is the potential it has for determining the best number of traces to use when performing ordered attacks. Indeed, the best leads to answer this question came from the study of the values of the scores given by the siamese network. For example, the validation set could be used to set a threshold under which the predictions are discarded. Even if such a threshold is not the most accurate, it would still result in an improvement over

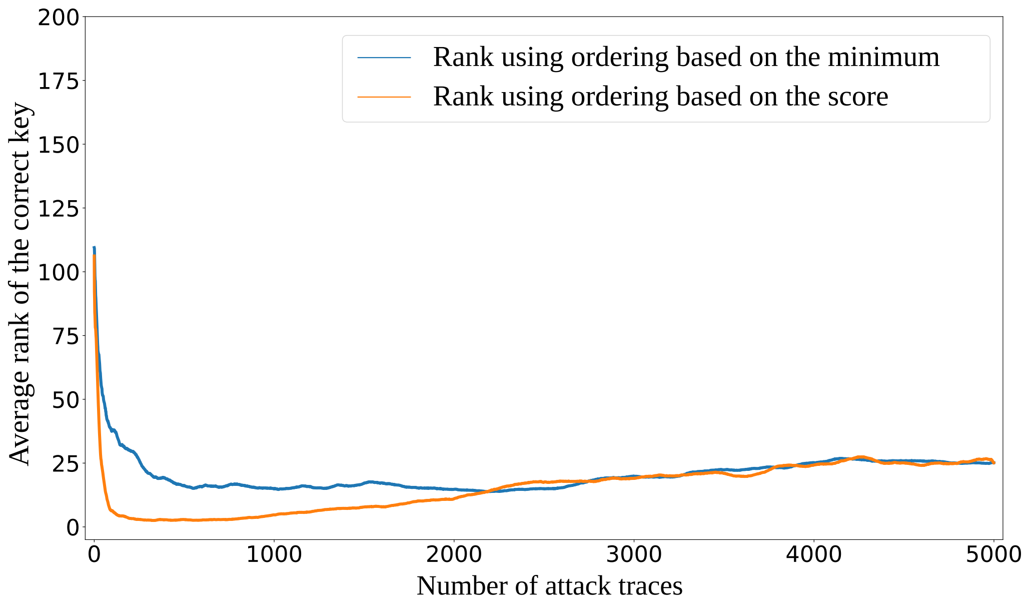


Figure 23: Evolution of the average rank of the correct key for attacks using a score order and a **descending** order based on the minimum of the predictions of the CNN$_{\text{best}}$ network on the database ASCAD variable key and the Desync100 dataset.

a classical attack, as mentioned in Remark 2 in Section 3.3. The same method could also be applied to the other means of ordering the predictions such as the values of minimum or the median. Another possible lead to follow could be to check for biases in the predictions of the validation set, more specifically if some specific intermediate values are predicted more accurately than others. If true, it would be possible to perform the attack only using these traces and look for improvements. Finally, we wanted to mention that, in this work, we focused on the predictions as a mean to order the traces and identify the best subset to use but one could also look at the traces themselves in order to find some information or biases that could lead to better attack results.

Finally, even though the attacks presented here were performed in a fixed context, one would expect that, if the adversary was able to obtain more attack traces, the new attack method would still produce better results than the classical one. The use of ordering would act as a filter for the new traces and make it possible to use the best new prediction early in the attack, which would lower the rank even further.

## 7 Conclusion

With the popularization of the use of machine learning algorithms to perform side-channel analysis, many techniques originating from the machine learning field have been used to improve the results of those new attacks. This article reports on a different approach. It asks a more fundamental question about the way side-channel attacks are performed and tries to answer the question using different methods. This new problem focuses on the way profiling attacks consider the traces they use and their corresponding predictions. Whereas before, the traces were chosen randomly to accumulate the information and retrieve the key, this article shows that not all traces contribute to the success of the attacks. This fact makes it possible to redefine profiling attacks by adding a new step that establishes the order in which the traces and their predictions are to be used. In addition, it defines two different cases for an attack: the worst case scenario, where the traces are used in random order and

Table 1: Summary of the result on $\text{CNN}_{\text{best}}$ in terms of reduction of the minimal rank over the final rank depending on the method used to order the predictions (minimum or score), the dataset and the number of traces used. Numbers in **bold** show the best results.

| Dataset | Desync | Number of traces | Min. val. of the rank using the min/Reduction over the final rank | Min. val. of the rank using the score/Reduction over the final rank | Final rank |
|---|---|---|---|---|---|
| ASCAD fixed key | 0 | 1000 | 1<br>- | 1<br>- | 1 |
| | 50 | 1000 | 21<br>-40% | **16**<br>**-54%** | 35 |
| | | 5000 | 1.8<br>-60% | **1.6**<br>**-64%** | 4.5 |
| | 100 | 1000 | 33<br>-61% | **23**<br>**-73%** | 85 |
| | | 5000 | **2**<br>**-96%** | **2**<br>**-96%** | 51 |
| ASCAD variable key | 0 | 1000 | 1<br>- | 1<br>- | 1 |
| | 50 | 1000 | 41<br>-8% | **24**<br>**-46%** | 45 |
| | | 5000 | 4.5<br>-40% | **1.5**<br>**-80%** | 7.5 |
| | 100 | 1000 | 59<br>-13% | **34**<br>**-50%** | 68 |
| | | 5000 | 14<br>-44% | **3**<br>**-88%** | 25 |

only the final rank is considered, and the best case scenario, where the best order is chosen and the number of traces used in the attack is restricted, which leads to a lower minimum rank than the final rank. In this article, different ways of ordering the traces are explored and enable a reduction in the minimum rank of the correct key during the attack. By defining a new loss function based on the *Learning to Rank* approach from machine learning, the article shows that it is possible to train a neural network to score the predictions and use this score to order the traces. This method of ordering the predictions yields the best results as the minimum rank is an improvement on the final rank in all the scenarios considered and even reduces the final rank by up to 90%.

However, some important problems remain and pose interesting challenges for future work. For example, the search for better architectures for the scoring network to be able to obtain better orders. It is also important to note that the work reported in this article is theoretical as no real attacks were performed and the results represent an evaluation of the new method proposed. As such, an interesting research problem would be to study the combination of the present results and those reported in [APSV20].

## A Networks

Table 2: **Network hyperparameters for $\mathbf{CNN_{best}}$** [BPS+19]

| Layer type | Hyperparameters |
|---|---|
| Trace input | 700 or 1400 |
| Convolution 1D | Filter = 64, Filter length = 11, Padding = Same, Activation = ReLU |
| Average Pooling | Pool length = 2 |
| Convolution 1D | Filter = 128, Filter length = 11, Padding = Same, Activation = ReLU |
| Average Pooling | Pool length = 2 |
| Convolution 1D | Filter = 256, Filter length = 11, Padding = Same, Activation = ReLU |
| Average Pooling | Pool length = 2 |
| Convolution 1D | Filter = 512, Filter length = 11, Padding = Same, Activation = ReLU |
| Average Pooling | Pool length = 2 |
| Convolution 1D | Filter = 512, Filter length = 11, Padding = Same, Activation = ReLU |
| Average Pooling | Pool length = 2 |
| Flatten | - |
| Fully-connected | Neurons = 4096 |
| Fully-connected | Neurons = 4096 |
| Output | Softmax: 256 classes |

Table 3: **Hyperparameters of the siamese network.**

| Layer | Hyperparameters |
|---|---|
| Input prediction | Size : 256 |
| Fully-connected | Number of neurons = 1024, Activation = ReLU |
| Fully-connected | Number of neurons = 1024, Activation = ReLU |
| Output | Linear : 1 score value |

## References

[AGF22] Rabin Y. Acharya, Fatemeh Ganji, and Domenic Forte. Information theory-based evolution of neural networks for side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):401–437, Nov. 2022.

[APSV20] Melissa Azouaoui, Romain Poussier, François-Xavier Standaert, and Vincent Verneuil. Key enumeration from the adversarial viewpoint. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications*, pages 252–267, Cham, 2020. Springer International Publishing.

[BBB$^+$93] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a “siamese” time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688, 1993.

[BPS$^+$19] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ascad database. *Journal of Cryptographic Engineering*, 10:163–188, 2019.

[BRL06] Christopher Burges, Robert Ragno, and Quoc Le. Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems*, 19:193–200, 2006.

[BSR$^+$05] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96, 2005.

[BVH$^+$16] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-convolutional siamese networks for object tracking. In Gang Hua and Hervé Jégou, editors, *Computer Vision – ECCV 2016 Workshops*, pages 850–865, Cham, 2016. Springer International Publishing.

[CAS16] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys ’16, page 191–198. Association for Computing Machinery, 2016.

[CLL$^+$09] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. *Advances in Neural Information Processing Systems*, 22:315–323, 2009.

[CQL$^+$07] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[GFZ$^+$17] Qing Guo, Wei Feng, Ce Zhou, Rui Huang, Liang Wan, and Song Wang. Learning dynamic siamese network for visual object tracking. In *Proceedings of the IEEE international conference on computer vision*, pages 1763–1771, 2017.

[GSC$^+$19] Martin Gleize, Eyal Shnarch, Leshem Choshen, Lena Dankin, Guy Moshkowich, Ranit Aharonov, and Noam Slonim. Are you convinced? choosing the more convincing evidence with a siamese network. *arXiv preprint arXiv:1907.08971*, 2019.

[HHO20] Anh-Tuan Hoang, Neil Hanley, and Maire O'Neill. Plaintext: A missing feature for enhancing the power of deep learning in side-channel analysis? breaking multiple layers of side-channel countermeasures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):49–85, Aug. 2020.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[Li14] Hang Li. Learning to rank for information retrieval and natural language processing. *Synthesis lectures on human language technologies*, 7(3):1–121, 2014.

[LMK$^+$11] Yuanhua Lv, Taesup Moon, Pranam Kolari, Zhaohui Zheng, Xuanhui Wang, and Yi Chang. Learning to model relatedness for news recommendation. In *Proceedings of the 20th international conference on World wide web*, pages 57–66, 2011.

[QLL10] Tao Qin, Tie-Yan Liu, and Hang Li. A general approximation framework for direct optimization of information retrieval measures. *Information retrieval*, 13(4):375–397, 2010.

[RBHG21] Damien Robissout, Lilian Bossuet, Amaury Habrard, and Vincent Grosso. Improving deep learning networks for profiled side-channel analysis using performance improvement techniques. *J. Emerg. Technol. Comput. Syst.*, 17(3), June 2021.

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

[WBSG10] Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Information Retrieval*, 13(3):254–270, 2010.

[XJP$^+$10] Biao Xiang, Daxin Jiang, Jian Pei, Xiaohui Sun, Enhong Chen, and Hang Li. Context-aware ranking in web search. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 451–458, 2010.

[XLW$^+$08] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199, 2008.

[ZBD$^+$21] Gabriel Zaid, Lilian Bossuet, François Dassance, Amaury Habrard, and Alexandre Venelli. Ranking loss: Maximizing the success rate in deep learning side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 25–55, 2021.

[ZBHV20] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, 2020.