Leakage Assessment through Neural Estimation of the Mutual Information

Valence Cristiani, Maxime Lecomte, Philippe Maurine

Leakage Assessment through Neural Estimation of the Mutual Information

Valence Cristiani[1], Maxime Lecomte[1] and Philippe Maurine[2]

[1] CEA, France
[2] LIRMM, France

Abstract. A large variety of side-channel attacks have been developed to extract secrets from electronic devices through their physical leakages. Whatever the utilized strategy, the amount of information one could gain from a side-channel trace is always bounded by the Mutual Information (MI) between the secret and the trace. This makes it, all punning aside, a key quantity for leakage evaluation. Unfortunately, traces are usually of too high dimension for existing statistical estimators to stay sound when computing the MI over full traces. However, recent works from the machine learning community have shown that it is possible to evaluate the MI in high dimensional space thanks to newest deep learning techniques. This paper explores how this new estimator could impact the side channel domain. It presents an analysis which aim is to derive the best way of using this estimator in practice. Then, it shows how such a tool can be used to assess the leakage of any device.

Keywords: Side channel analysis, Mutual information, Deep learning

1 Introduction

Side Channel Analysis (SCA) could be defined as the process of gaining information on a device holding a secret through its physical leakage such as power consumption [11] or Electromagnetic (EM) emanations [16]. The secret is usually a cryptographic key but could be as well basic block execution, assembly instructions, or even the value of an arbitrary register. The basic assumption is that the secret and the side-channel data are statistically dependent. Many techniques have been developed to extract part of these dependencies such as DPA [11], CPA [3], MIA [15], and profiling attacks [6]. This diversity makes it hard for designers and evaluators to draw an objective metric in order to assess leakage. Testing all the existing attacks is a possible strategy but the incentives to develop a leakage assessment protocol can be found in a couple of works [4, 13, 18].

From an information theory point of view, the maximum amount of information one could extract from a side-channel trace is bounded by the mutual information, $\mathcal{I}(S, X)$ between the secret S and the trace X, seen as random variables. This quantity is, indeed, central in the side-channel domain. The goals of the different actors could be summarized as follows:

- **Designers** aim at implementing countermeasures to decrease as far as possible $\mathcal{I}(S, X)$, with computational, spatial and efficiency constraints.
- **Evaluators** aim at estimating $\mathcal{I}(S, X)$ as closely as possible to assess leakages in a worst-case scenario.
- **Attackers** aim at developing strategies to partially or fully exploit $\mathcal{I}(S, X)$ in order to recover a secret.

The main problem for designers and evaluators is that $\mathcal{I}(S, X)$ is known to be hard to estimate from drawn samples when the variables live in a high dimensional space, which is generally the case of X. Indeed, computing $\mathcal{I}(S, X)$ usually requires an estimation of the conditional density $\Pr(S|X)$ which is hard because of the well-known curse of dimensionality. This explains why conventional leakage assessment tools [18] (Signal to Noise Ratio (SNR), T-tests) and classical attack strategies such as CPA and MIA typically focus on one (or a few) samples at a time in the trace. As a result, the amount of information effectively used may be significantly lower than $\mathcal{I}(S, X)$.

Latest deep learning attacks have proved that neural networks are a very interesting tool able to combine information from many samples of the traces without any prior knowledge on the leakage model. For instance, [14] recently proposed a way to derive an estimation of $\mathcal{I}(S, X)$ from the success rate of their attacks and showed that in a supervised context, neural networks are close to optimal at extracting information from traces.

In a completely unrelated context, Belghazi *et al.* [1] lately introduced a Mutual Information Neural Estimator (MINE) which uses the power of deep learning to compute mutual information in high dimension. They have proposed applications in a pure machine learning context but we argue that this tool might be of great interest in the side-channel domain. Indeed, being able to efficiently compute $\mathcal{I}(S, X)$ in an unsupervised way (no profiling of the target needed), no matter the target, the implementation, or the countermeasures used, would be highly relevant for all the different parties.

Paper organization. For this paper to be self-contained, the general method and the mathematical ideas behind MINE are recalled in section 2. Section 3 proposes an in-depth analysis of MINE in a side-channel context supported with synthetic traces, and suggests ways of dealing with the overfitting problem. Section 4 provides some real case applications. It shows how this estimator constitutes a reliable leakage assessment tool that can be used to compare leakage from different implementations and devices. This section also shows that such an estimator can be used as a guide for an evaluator/attacker to maximize the MI captured from different hardware side-channel setups.

2 Background and theory behind MINE

Notations. Random variables are represented as upper case letters such as X. They take their values in the corresponding set $\mathcal{X}$ depicted with a calligraphic letter. Lower case letters such as x stand for elements of $\mathcal{X}$. Probability density function associated to the variable X is denoted by p_X.

Background. The entropy $H(X)$ [19] of a random variable is a fundamental quantity in information theory which typically tells how much information one would get in average by learning a particular realization x of X. It is defined as the expectation of the self-information $log_2(1/p_X)$. In a discrete context:

$$H(X) = \sum_{x \in \mathcal{X}} p_X(x) \cdot log_2\left(\frac{1}{p_X(x)}\right) \tag{1}$$

In a side-channel environment where X represents the acquired data, one is not interested in the absolute information provided by X but rather in the amount of information revealed about a second variable such as a secret S. This is exactly what is measured by the mutual information $\mathcal{I}(S, X)$. It is defined as: $\mathcal{I}(S, X) = H(S) - H(S|X)$ where $H(S|X)$ stands for the conditional entropy of S knowing X:

$$H(S|X) = \sum_{x \in \mathcal{X}} p_X(x) \cdot H(S|X = x) \tag{2}$$

The most common ways to estimate MI are the histogram method and the kernel density estimation both described in [15]. There also exists a non parametric estimation based on k-nearest neighbors [12]. This paper is interested in MINE [1], a new estimator based on deep learning techniques, which claims to scale well with high dimensions. Technical details about MINE are given hereafter.

MINE. A well known property of $\mathcal{I}(S, X)$ is its equivalence with the Kullback-Leibler (KL) divergence between the joint probability $p_{S,X} = \Pr(S, X)$ and the product of the marginals $p_S \otimes p_X = \Pr(S) \cdot \Pr(X)$:

$$\mathcal{I}(S, X) = D_{KL}(p_{S,X} \,||\, p_S \otimes p_X) \tag{3}$$

where $D_{KL}(p, q)$ is defined as follow:

$$D_{KL}(p \,||\, q) = \mathbb{E}_p[log\left(\frac{p}{q}\right)] \tag{4}$$

whenever p is absolutely continuous with respect to q. This property guarantees that when q is equal to 0, p is also equal to 0 and there is no division by 0 in the logarithm. By definition, $p_{S,X}$ is absolutely continuous with respect to $p_S \otimes p_X$.

The key technical ingredient of MINE is to express the KL-divergence with variational representations, especially the Donsker-Varadhan representation that is given hereafter. Let p and q be two densities over a compact set $\Omega \in \mathbb{R}^d$.

Theorem 1. *(Donsker-Varadhan, 1983) The KL-divergence admits the following dual representation:*

$$D_{KL}(p \,||\, q) = \sup_{T:\, \Omega \to \mathbb{R}} \mathbb{E}_p[T] - log(\mathbb{E}_q[e^T]) \tag{5}$$

where the supremum is taken over all functions T such that the two expectations are finite.

A straightforward consequence of this theorem is that for any set $\mathcal{F}$ of functions $T : \Omega \to \mathbb{R}$ satisfying the integrability constraint of the theorem we have the following lower bound:

$$D_{KL}(p \,||\, q) \geq \sup_{T \in \mathcal{F}} \mathbb{E}_p[T] - log(\mathbb{E}_q[e^T]) \tag{6}$$

Thus, using (3), one have the following lower bound for $\mathcal{I}(S, X)$:

$$\mathcal{I}(S, X) \geq \sup_{T \in \mathcal{F}} \mathbb{E}_{p_{S,X}}[T] - log(\mathbb{E}_{p_S \otimes p_X)}[e^T]) \tag{7}$$

How to compute $\mathcal{I}(S, X)$. To put it short, the idea is to define $\mathcal{F}$ as the set of all functions T_θ parametrized by a neural network with parameters $\theta \in \Theta$ and to look for the parameters maximizing the loss function $\mathcal{L} : \Theta \to \mathbb{R}$:

$$\mathcal{L}(\theta) = \mathbb{E}_{p_{S,X}}[T_\theta] - log(\mathbb{E}_{p_S \otimes p_X)}[e^{T_\theta}]) \tag{8}$$

This loss function is itself bounded by $\mathcal{I}(S, X)$. The universal approximation theorem for neural networks guarantees that this bound can be made arbitrarily tight for some well-chosen parameters θ. The goal is then to find the best θ, potentially using all the deep learning techniques and, more generally, all the tools for optimization problem-solving. The expectations in (8) can be estimated using empirical samples from $p_{S,X}$ and $p_S \otimes p_X$ and the maximization can be done with the classical gradient ascent. A noticeable difference with a classical deep learning setup is that the trained network is not used for any kind of prediction. Instead, the evaluation of the loss function at the end of the training gives an estimation of $\mathcal{I}(S, X)$. We give hereafter the formal definition of the estimator as stated in the original paper [1].

Definition 1. *(MINE) Let $\mathcal{A} = \{(s_1, x_1), \ldots, (s_n, x_n)\}$ and $\mathcal{B} = \{(\widetilde{s}_1, \widetilde{x}_1), \ldots, (\widetilde{s}_n, \widetilde{x}_n)\}$ be two sets of n empirical samples respectively from $p_{S,X}$ and $p_S \otimes p_X$. Let $\mathcal{F} = \{T_\theta\}_{\theta \in \Theta}$ be the set of functions parametrized by a neural network. MINE is defined as follows:*

$$\widehat{\mathcal{I}(S, X)}_n = \sup_{T \in \mathcal{F}} \overline{\mathbb{E}_{\mathcal{A}}[T]} - log(\overline{\mathbb{E}_{\mathcal{B}}[e^T]}) \tag{9}$$

where $\overline{\mathbb{E}_{\mathcal{S}}[\cdot]}$ stands for the expectation empirically estimated over the set $\mathcal{S}$.

The main theoretical result proved in [1] is that MINE is strongly consistent.

Theorem 2. *(Strong consistency) For all $\epsilon > 0$ there exist a positive integer N such that:*

$$\forall n > N, \quad |\mathcal{I}(S, X) - \widehat{\mathcal{I}(S, X)}_n| < \epsilon \tag{10}$$

In practice one often only have samples from the joint distribution: $\mathcal{A} = \{(s_1, x_1), \ldots (s_n, x_n)\}$. Samples from the product of the marginals can be artificially generated by shuffling the variable X using a random permutation σ:

$\mathcal{B} = \{(s_1, x_{\sigma(1)}), \ldots, (s_n, x_{\sigma(n)})\}$. We provide hereafter an implementation of MINE that uses minibatch gradient ascent. Note that $\mathcal{B}$ is regenerated after each epoch. Thus, this algorithm is not strictly implementing MINE as defined in (9) because $\mathcal{B}$ is fixed in this definition. Theoretical arguments are provided in section 3.4 to explain why this regeneration limit overfitting in practice and is therefore mandatory.

Algorithm 1: Mine implementation

Input: $\mathcal{A} = \{(s_1, x_1), \ldots (s_n, x_n)\}$
$\theta \leftarrow$ Initialize network parameters
Choose b a batch size such that $b \mid n$
repeat
  Generate $\mathcal{B} = \{(s_1, x_{\sigma(1)}), \ldots, (s_n, x_{\sigma(n)})\}$ with a random permutation σ
  Divide $\mathcal{A}$ and $\mathcal{B}$ into $\frac{n}{b}$ packs of b elements: $A_1, \ldots, A_{\frac{n}{b}}$ and $B_1, \ldots, B_{\frac{n}{b}}$
  for $i = 1$; $i = \frac{n}{b}$ **do**
    $\mathcal{L}(\theta) \leftarrow \overline{\mathbb{E}_{A_i}[T_\theta]} - log(\overline{\mathbb{E}_{B_i}[e^{T_\theta}]})$, *Evaluate the loss function*
    $G(\theta) \leftarrow \nabla_\theta \mathcal{L}(\theta)$, *Compute the gradient*
    $\theta \leftarrow \theta + \mu G(\theta)$, *Update the network parameters (μ is the learning rate)*
  end
until *convergence of* $\mathcal{L}(\theta)$

3 Analysis of MINE in a side-channel context

MI has found applications in a wide range of disciplines and it is not surprising that it is also of great interest for side-channel analysis. Unlike Pearson coefficient, it detects non-linear dependencies and thus does not require any assumptions on the leakage model. Another key property of the MI is that it is invariant to bijective transformations of the variables. This is of interest for side-channel as S usually represents the state of an internal variable (ex: $S = K \oplus P$ for an AES) and is therefore unknown but bijectively related to a known variable such as the plaintext P. In that case, there exists a bijective function f such that $S = f(P)$ and:

$$\mathcal{I}(S, X) = \mathcal{I}(f^{-1}(S), X) = \mathcal{I}(P, X) \tag{11}$$

Thus, one may estimate $\mathcal{I}(S, X)$ with only the knowledge of P and X and therefore quickly get the amount of leakage an attacker could potentially exploit.

In what follows, we consider that we are granted n samples $(s_1, x_1), \ldots, (s_n, x_n)$ of traces associated to the sensitive variable being processed in the device (or as stated above, to any bijection of this variable). These samples will be either generated on simulation or measured from real case experiments. The goal is to derive the best way to use MINE in a side-channel context in order to compute a reliable estimation of $\mathcal{I}(P, X)$.

3.1 Simulated traces environment

In order to assess the capabilities of MINE experiments on synthetic traces were first conducted. These traces have been generated using a leakage model which may seem awkward since the whole point of conducting MI analysis is to avoid any assumption on the leakage model. But we argue that as a first step, it brings a valuable advantage: the environment is perfectly controlled, thus the true MI is known and can be used to evaluate the results and compare different settings.

Trace generation. To generate traces, featuring $n_l + n_r$ independent samples, a sensitive byte $0 \leq s \leq 255$ was first drawn uniformly. The leakage was spread over the n_l samples drawn from a normal distribution centered in the Hamming Weight (HW) of s and with noise $\sigma \sim \mathcal{N}(HW(s), \sigma)$. The n_r remaining samples are random points added to the trace to artificially increase the dimension and be closer from a real scenario. Each of the n_r samples is drawn from a normal distribution centered in c and with noise $\sigma \sim \mathcal{N}(c, \sigma)$, where c is an integer itself drawn uniformly between 0 and 8. A very simple yet informative case is to set $n_l = 1, n_r = 0$ and $\sigma = 1$. In that case, the true mutual information $\mathcal{I}(S, X)$ is equal to:

$$\begin{aligned}
\mathcal{I}(S, X) &= H(S) - H(S|X) \\
&= 8 - \sum_{s=0}^{255} \int_{-\infty}^{\infty} \Pr(s, x) \cdot log_2\left(\frac{1}{\Pr(s|x)}\right) dx \\
&= 8 - \sum_{s=0}^{255} \int_{-\infty}^{\infty} \frac{1}{2^8} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x - HW(s))^2} \cdot log_2\left(\frac{\sum_{s'=0}^{255} e^{-\frac{1}{2}(x - HW(s'))^2}}{e^{-\frac{1}{2}(x - HW(s))^2}}\right) dx \\
&\approx 0.8 \text{ bits}
\end{aligned} \tag{12}$$

As a first step, we applied MINE to a set of 10k synthetic traces generated with these parameters. The network was set to be a simple Multi Layer Perceptron (MLP) with two hidden layers of size 20. The Exponential Linear Unit (ELU) was used as the activation function. The input layer was composed of two neurons, representing the value of the sensitive variable S and the one-dimensional trace X. The output was a single neuron giving the value of the function T_θ. The batch size was set to 500. The value of the loss function $\mathcal{L}(\theta)$ over time is plotted in Fig. 1. An epoch represents the processing of all the data so the parameters are updated 20 times per epoch.

As shown, the results are mixed. On one hand, the loss function is always under the true MI and it seems that the limit superior of MINE is converging over time towards 0.8, *i.e.* the true MI. On the other hand, the loss function experiences a lot of drops and the convergence is very slow (above 200k epochs). Drops may be due to the optimizer used (ADAM [10]) and happens when the gradient is very close to 0. Increasing the size/number of hidden layers did not produce any significantly better results. In that state MINE is clearly not of any use for side-channel: the convergence is not clear and a classical histogram method would compute the MI faster and better for one-dimensional traces.

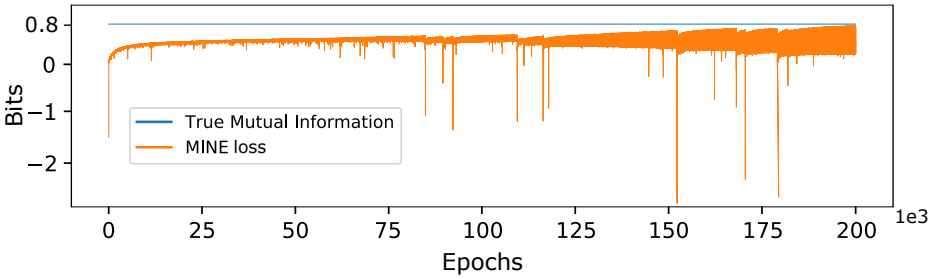


Fig. 1: Evolution of MINE's loss function over time

3.2 Input decompression

Trying to gain intuition about the reasons causing the network to perform poorly in this situation, we hypothesized that the information in the first layer, especially the value of s, could be too condensed in the sense that only one neuron is used to describe it. Intuitively, the information provided by s about the corresponding trace x is not continuous in s. The meaning of this statement is that there is no reason that two close values of s induce two close values of x. For example, in a noise-free Hamming weight leakage model, the traces corresponding to a sensitive value of 127 and 128 would be very different since $HW(127) = 7$ and $HW(128) = 1$. Since neural networks are built using a succession of linear and activation functions which are all continuous, approximating functions with quick and high variations may be harder for them. Indeed, building a neural classifier that extracts the Hamming weight of an integer is not an easy task. However, if the value of this integer is split into multiple neurons holdings its binary representation, the problem becomes trivial as it ends up being a simple sum.

This observation led us to increase the input size to represent the value of s in its binary form, thus using 8 neurons. However, computing $\mathcal{I}(S, X)$ in that case gives an unfair advantage to the arbitrarily chosen Hamming weight model. Indeed, the value of X would be closely related to the sum of the input bits. So we decided to compute $\mathcal{I}(S \oplus k, X)$ instead of $\mathcal{I}(S, X)$, with a fixed k. As stated above this bijective transformation does not change the MI anyway and removes a possible confusion factor in the analysis. Results with the same parameters as before ($n_l = 1, n_r = 0$, $\sigma = 1$) are presented in Fig. 2. They bear no comparison with the previous ones. With this simple trick, MINE quickly converges toward the true MI. The estimation seems robust as restarting the training from different initializations of the network always produces the same results. The order of magnitude of the computational time for the 500 epochs is around two minutes.

Remark 1. Note that any constant function T_θ would produce a loss function of 0. We argue that this could explain the knee point around 0 bit in the learning curve: it is indeed, easy for the network to quickly reach 0 tuning the parameters towards a constant function, before learning anything interesting.

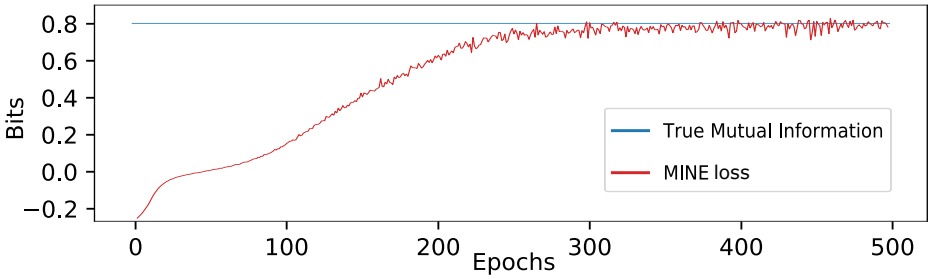


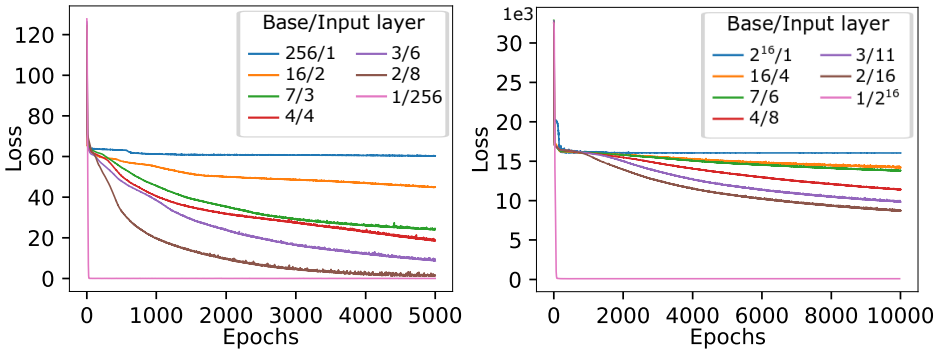
Fig. 2: MINE with input decompression

Before testing this method for higher dimension traces, we propose to analyze more in-depth this Input Decompression (ID). The goal is to understand if this result was related in any way to our simulation setup or if ID could be applied to more generic cases. As a first step, we tried to decompress X instead of S, binning the value of X to the closest integers, then using its binary representation as input neurons. As expected, it did not work: results looked like Fig. 1, as X is by essence a continuous variable. If there is no interest in splitting into multiple neurons an intrinsically continuous variable, our hypothesis is that, for categorical variables, the greater the decompression, the faster the training.

Learning random permutations In order to test this hypothesis in a more generic case, this section proposes to build a neural network P_θ which goal is to learn a random permutation P of $\{0, \dots, n-1\}$ and to analyze its performance in terms of ID. Permutations have been chosen because they are arbitrary functions with no relation between close inputs. For an integer $m < n$ the network returns a float $P_\theta(m)$ which has to be as close as possible to $P(m)$. The loss function was defined as error: $\mathcal{L}(\theta) = |P_\theta(m) - P(m)|$. Network architecture was again a simple MLP but with 3 hidden layers of size 100. To study the effect of ID the input layer was defined to be the representation of m in different bases, with one neuron per digit. For example, with n = 256, all bases in $256, 16, 7, 4, 3, 2, 1$ were considered resulting in a first layer of respectively $\{1, 2, 3, 4, 6, 8, 256\}$ neurons (base 1 is actually the One Hot Encoding (OHE)). The training dataset was a list of 10k integers uniformly drawn from $\{0, \dots, n-1\}$. Loss functions in terms of ID are depicted in Fig. 3. At the end of the training, plots are exactly ordered in the expected way: greater decompression leads to faster and better training.

In a recent analysis, Bronchain *et al.* [5] have shown that it was hard for a MLP to learn the Galois multiplication in $GF(2^n)$ when $n \geq 8$. As Galois multiplication suffers from the same non-continuity than random permutations, we argue that blue plots confirm this result. With no ID our MLP did not show the beginning of a convergence towards 0. But we do think their network may have been successful with ID. Pink plots show that the best choice is to use the OHE. The problem with OHE is that the number of neurons (and therefore the computational time) scales linearly with n (the number of categories of the

underlying problem), where it only scales logarithmically with any other base. In side-channel, one mostly deals with bytes (256 categories) and will therefore use the OHE. However, section 4.3 presents a scenario where using base 2 is a better choice, when computing the MI with assembly instructions.



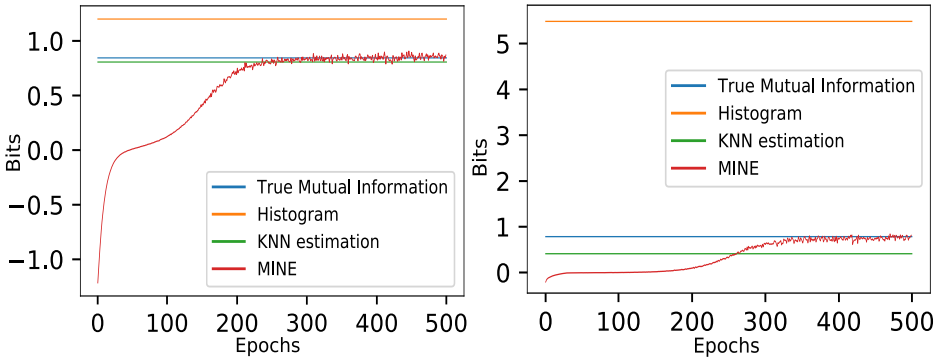
(a) $n = 2^8$ (b) $n = 2^{16}$

Fig. 3: Impact of input decompression on learning random permutations

Remark 2. Note that the constant function $P_\theta = \frac{n}{2}$ would result in an average loss function of $\frac{n}{4}$, which explains the knee point around $\frac{n}{4}$ observable in most of the curves: quickly converging towards this function is an efficient strategy for the network to minimize its loss at the beginning of the learning phase. We verified this statement by looking at the predictions of the network which were all close from $\frac{n}{2}$ in the early stage of the training.

3.3 MINE in higher dimension

This section presents results of simulations in higher dimension and compare MINE estimation to that provided by the classical histogram and KNN methods. The histogram estimator has been implemented following the description from [15] and Steeg's implementation [21] has been utilized for KNN. Network architecture described in 3.1 has been used with OHE to encode the S variable. We have kept $n_l = \sigma = 1$ so the true MI is still around 0.8 bits but n_r was no longer set to 0 in order to increase the traces dimension. Fig. 4 shows the results for $n_r = 1$ and $n_r = 9$. In both case MINE correctly converges toward the true MI. The histogram method tends to overestimate the MI (as explained in [23]) while KNN method underestimates it. With dimension greater than 10 these methods are not reliable anymore. One could argue that any dimension reduction technique applied in the above experiments would allow to compute the MI with classical estimators. While this is true in this case it may result in a loss of information in a real case scenario where the information could be split into multiple samples of the traces. We have conducted many experiments with different parameters and MINE always returned reliable estimations even in very high dimension (ex: Fig. 5b with $n_l = 5$ and $n_r = 1000$).



(a) Trace dimension $= 2$
$(n_l = 1, n_r = 1, \sigma = 1)$

(b) Trace dimension $= 10$
$(n_l = 1, n_r = 9, \sigma = 1)$

Fig. 4: Comparison of MINE with classical estimators in higher dimension

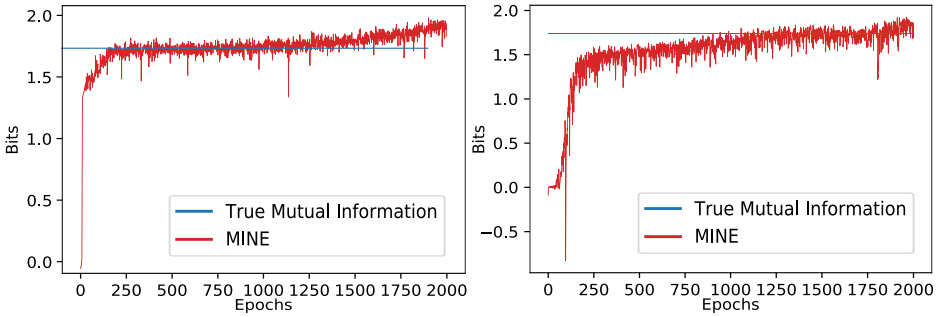
3.4 Analysis of the overfitting problem

Results of simulations are encouraging as they seem accurate with a lot of different parameters but one problem still has to be solved before testing MINE on real traces: when to stop the training ? Until now, training has been manually stopped when the loss had converged towards the true MI. No such threshold value will be granted in real cases. One could argue that since the loss function is theoretically bounded by the true MI, a good strategy would be to let the training happen during a sufficiently long time and to retain the supremum of the loss function as the MI estimation. We argue that this strategy is not viable: in practice the bound does not hold as expectations in the loss are not the true expectations but are only estimated through empirical data. Thus, MINE can still produce output above the true MI. Fig. 5 shows this phenomenon: training has been intentionally let running for a longer time in these experiments and MINE overestimates the MI at the end of the training. In other terms, MINE is no exception to the rule when it comes to the overfitting problem: the network can learn ways to exploit specificities of the data it is using to train, in order to maximize its loss function. We propose hereafter a detailed analysis of this problem and answer to the following question: is it possible to control (for example to bound with a certain probability) the error made by the network ?

Let us return to the definition of MINE estimator:

$$\widehat{\mathcal{I}(S,X)}_n = \sup_{\theta \in \Theta} \overline{\mathbb{E}_{\mathcal{A}}[T_\theta]} - log(\overline{\mathbb{E}_{\mathcal{B}}[e^{T_\theta}]}) \tag{13}$$

The problem comes from the fact that the two expectations are estimated over the set of empirical data $\mathcal{A}$ and $\mathcal{B}$. The error can not be controlled in the classical way with the central limit theorem because there is a notion of order that is important: the two sets $\mathcal{A}$ and $\mathcal{B}$ are selected *before* the network tries to find the supremum over Θ. Thus, the network can exploit specificities of $\mathcal{A}$ and $\mathcal{B}$ in



(a) Trace dimension = 15 ($n_l = 5, n_r = 10, \sigma = 1$)

(b) Trace dimension = 1005 ($n_l = 5, n_r = 1000, \sigma = 1$)

Fig. 5: Over estimation of MINE at the end the training (overfitting)

its research. We show in theorem 3 that given two sets $\mathcal{A}$ and $\mathcal{B}$, the supremum may not even be bounded.

Theorem 3. *Let X, Y be two random variables over Ω. Let $\boldsymbol{x} = (x_1, \ldots, x_n) \in \Omega^n$ and $\boldsymbol{y} = (y_1, \ldots, y_n) \in \Omega^n$ be two samples of n realizations of respectively X and Y. Then,*

$$\sup_{T:\, \Omega \to \mathbb{R}} \overline{\mathbb{E}_{\boldsymbol{x}}[T(X)]} - log(\overline{\mathbb{E}_{\boldsymbol{y}}[e^{T(Y)}]}) < \infty \Leftrightarrow \forall i,\ \exists j \text{ such that } x_i = y_j$$

Proof. Let us introduce two new random variables, X' and Y' defined as follows:

$$\forall \omega \in \Omega,\ \mathbb{P}(X' = \omega) = \frac{1}{n} \cdot |\{x_i = \omega\}| \quad \text{and} \quad \mathbb{P}(Y' = \omega) = \frac{1}{m} \cdot |\{y_i = \omega\}|$$

The samples $\boldsymbol{x}$ and $\boldsymbol{y}$ are perfect samples of X' and Y' (by definition of X' and Y'), thus, the estimated expectations are equal to the true expectations computed over this new variables:

$$\sup_{T:\, \Omega \to \mathbb{R}} \overline{\mathbb{E}_{\boldsymbol{x}}[T(X)]} - log(\overline{\mathbb{E}_{\boldsymbol{y}}[e^{T(Y)}]}) = \sup_{T:\, \Omega \to \mathbb{R}} \mathbb{E}_{X'}[T(X')] - log(\mathbb{E}_{Y'}[e^{T(Y')}])$$

Now let us assume the right part of the equivalence. This condition means that there is no isolated x_i, or in other words: $\forall \omega, \Pr(Y' = \omega) = 0 \Rightarrow \Pr(X' = \omega) = 0$. This guarantees the absolute continuity of $p_{X'}$ with respect to $p_{Y'}$ and thus, that $D_{KL}(p_{X'} || \, p_{Y'})$ exists. Therefore, using Th. 1:

$$\sup_{T:\, \Omega \to \mathbb{R}} \overline{\mathbb{E}_{\boldsymbol{x}}[T(X)]} - log(\overline{\mathbb{E}_{\boldsymbol{y}}[e^{T(Y)}]}) = D_{KL}(p_{X'} || \, p_{Y'}) < \infty$$

If, on the other hand, this condition is false: $\exists i$ such that $\forall j,\ x_i \neq y_j$. For any given function T one can exploit this isolated x_i modifying $T(x_i)$ without influencing the second expectation. In particular, if $T(x_i)$ tends towards infinity:

$$\begin{aligned}\lim_{T(x_i)\to\infty}\left[\overline{\mathbb{E}_{\boldsymbol{x}}[T(X)]}\text{–}log(\overline{\mathbb{E}_{\boldsymbol{y}}[e^{T(Y)}]})\right] &= \lim_{T(x_i)\to\infty}\left[\frac{1}{n}\sum_{k=1}^{n}x_kT(x_k)\text{–}log(\frac{1}{n}\sum_{k=1}^{n}y_ke^{T(y_k)})\right]\\ &= \lim_{T(x_i)\to\infty}\left[\frac{1}{n}\sum_{k=1}^{n}x_kT(x_k)\right]\text{–}log(\frac{1}{n}\sum_{k=1}^{n}y_ke^{T(y_k)})\\ &= \infty\end{aligned}$$

So in that case:

$$\sup_{T\colon \Omega\to\mathbb{R}}\overline{\mathbb{E}_{\boldsymbol{x}}[T(X)]} - log(\overline{\mathbb{E}_{\boldsymbol{y}}[e^{T(Y)}]}) = \infty$$

□

This theorem means that most of the time (and especially for high dimensional variables) the supremum is infinite and MINE is not even well defined. The natural question that comes now is: why does MINE seem to work in practice?

We claim that this is due to the implementation and especially to the randomization of the set $\mathcal{B}$ evoked in section 2: after each epoch a new permutation σ is drawn to generate samples from $p_S \otimes p_X$: $\mathcal{B} = \{(s_1, x_{\sigma(1)}), \ldots, (s_n, x_{\sigma(n)})\}$. Thus, the isolated samples from $\mathcal{A}$ are not always the same at each epoch which does not leave time for the network to exploit them. To verify that this was a key element, MINE was run without this randomization process. The loss function diverged towards infinity, as predicted by theorem 3.

In the long run, the network can still learn statistical specificities of the dataset such as samples from $\mathcal{A}$ that has a greater probability of being isolated, and exploit them. This explains why MINE may overfit when it has a long time to train. That is why we suggest to add a validation loss function.

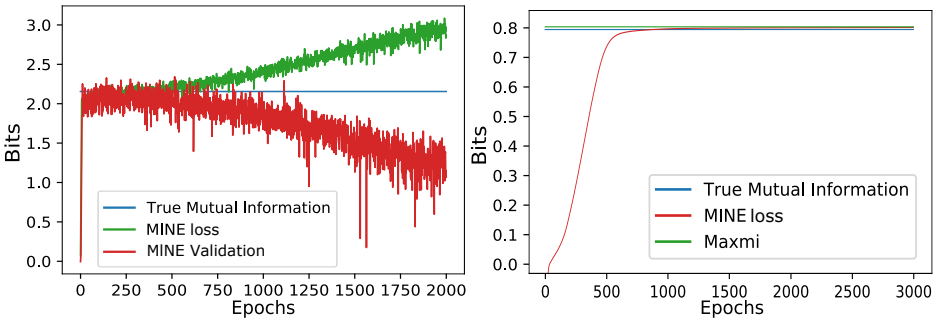
Validation loss function. A validation loss function is a common tool when it comes to detect overfitting and stop the training at the right time. The idea is to split the dataset $\mathcal{A}$ into a training dataset $\mathcal{A}_t$ and a validation one $\mathcal{A}_v$ and to only use A_t for the training. At the end of each epoch, the loss function is computed both on A_t and A_v. As the data from A_v are never used during training, MINE can not overfit on them. Thus, it is safe to take the supremum of the loss computed over A_v as our MI estimation. It also provides a useful condition to stop the training as the decrease of the validation loss function is usually a sign of overfitting. Fig. 6a shows an example where the true loss function and the validation one (computed on 80% and 20% of the data) respectively increase and decrease after a while. Training could have been stopped after the 500^{th} epoch.

Fill the holes. Th. 3 states that there is still a case where the supremum is bounded: when $\forall a \in \mathcal{A},\ \exists b \in \mathcal{B}$ such that $a = b$, or in other words, when there is no isolated value in $\mathcal{A}$. An alternative solution to prevent overfitting is thus to

force this condition to be true instead of regenerating $\mathcal{B}$ after each epoch. Naively filling the holes by adding to $\mathcal{B}$ all the isolated values is not a good idea because the resulting set would be biased, not containing *stricto sensu* samples drawn from $p_S \otimes p_X$. However, with $\mathcal{A} = \{(s_1, x_1), \dots (s_n, x_n)\}$, $\mathcal{A}_s = \{s_1, \dots, s_n\}$ and $\mathcal{A}_x = \{x_1, \dots, x_n\}$ one can define $\mathcal{B}'$ as the Cartesian product[3] $\mathcal{B}' = \mathcal{A}_s \times \mathcal{A}_x$ which is by definition a non-biased dataset that covers all the elements of $\mathcal{A}$. The problem is that its size is no longer n but n^2 which drastically impacts the computational time of MINE as the network has to compute $T_\theta(b)$ for all $b \in \mathcal{B}'$ at each epoch. However, the number of network evaluations can be reduced to $c \cdot n$ where c is the cardinality of the set $\mathcal{S}$ made up of all the possible values taken by the sensitive variable S. For example, if S is a byte, $c = 256$. The idea is to evaluate T_θ on the $c \cdot n$ elements of the set $\mathcal{S} \times \mathcal{A}_x$ which is sufficient to cover all the couples from $\mathcal{B}'$ as elements from $\mathcal{A}_s \times \mathcal{A}_x$ can all be found in $\mathcal{S} \times \mathcal{A}_x$.

With this implementation MINE is fundamentally bounded by a quantity denoted $Maxmi$ equal to the KL-divergence between the empirical distributions associated to $\mathcal{A}$ and $\mathcal{B}$ as stated in the proof of Th. 3. Fig. 6b shows an example of MINE with this implementation applied to the already considered case ($n_l = 1, n_r = 0, \sigma = 1$). One may observe that the loss function is a lot smoother and is effectively bounded by $Maxmi$ (we tried to let the network train for more than 100k epochs) which is another empirical confirmation of Th. 3.

However, when the dimension of the variables increases samples tend to be more and more unique. At the limit, they hold the full information about the corresponding secret s which means that $Maxmi$ will tends towards $H(S)$. Knowing if the network will always converge towards his supremum or will stabilize to a value close to the true MI is an open question. We do think that the randomization proposed in the precedent strategy may help to that aim and that is why we will stick to the validation method for our real-life experiments, which is faster anyway.



(a) Validitaion strategy ($n_l = 10, n_r = 0, \sigma = 1$)

(b) Fill the holes strategy ($n_l = 1, n_r = 0, \sigma = 1$)

Fig. 6: Two possible strategies against overfitting

[3] These sets are actually multisets as they may contains repetitions of a single elements but the Cartesian product can be canonicaly extended to multisets.

4 Application of MINE in an evaluation context

This section provides real case examples where MINE could be useful especially in an evaluation context. Its most straightforward utilization is probably to assess the quantity of information leaking from a device when it computes a cryptographic algorithm. It can be seen as a first security metric, easy to compute whatever the target and the implementation, with low expertise required. However, MINE only returns an upper bound on the amount of leakage potentially usable. In practice, an attacker may not be able to fully exploit this information, depending on his strategy, and that is why classical evaluation methods still have to be performed.

That being said, MINE is also a great comparison tool. Indeed, its output is an interpretable number that allows to objectively rank different devices or implementations in terms of their leakage. It can be used to analyze the effect of a countermeasure or even to compare different hardware setup in order to maximize the MI for future attacks or evaluations.

4.1 Leakage evaluation of an unprotected AES

As a first real case experiment, our target was an unprotected AES implemented on a cortex M4 device. 20k EM traces centered on the first round of the AES have been acquired through a Langer probe (RF-B 0,3-3) linked to an LNA and a Tektronix oscilloscope (MSO64, 2.5GHz) with a sample rate of 1GS/s. Resulting traces had a length of 50k samples. They have been labeled with the first byte of the corresponding plaintext which was drawn randomly for each computation of the AES.

The main goal of this first experiment was to demonstrate how adding more samples to the analysis, which is the purpose of MINE, increases the amount of information one can recover. To this end, only the n samples with the maximum SNR were kept in the traces, with n in $\{1, 5, 500\}$. Network architecture was the same as in the simulated experiments. Results are presented in Fig. 7. The thick blue plot shows that if one only uses one sample in his analysis (for example with a CPA or histogram-based MIA) he would be able to extract at most 1.15 bits about the secret, per trace. While it is a huge amount of information (it is an unprotected AES) it is possible to extract almost 4 times more using 500 samples. For clarity reasons, only the validation loss has been plotted. Training has been stopped after epoch 500 as these validations (especially the green one) started to decrease. Going further with $n \geq 500$ did not produce better results as it seems that the remaining samples were absolutely not informative about the secret.

ADC comparison. The oscilloscope used in the former experiment offers the possibility to set the ADC precision to either 8 or 15 bits. This is a good opportunity to show MINE comparative interest and its ability to answer questions such as "Is it really worth it to buy the newest scope with the enhanced ADC

precision?" in a quantitative and objective way. 10k traces instead of 20k (so that the occupied memory stayed constant) were thus acquired with the 15 bits precision. Results are represented by the thin plots on Fig. 7. In this case, the answer is that there is a slight improvement (around 10%) working with the 15 bits precision rather than the 8 bits one.

4.2 Leakage evaluation of a masked AES from the ASCAD database

One of the main difficulties of side-channel analysis is to extract information even when the target algorithm has been masked. Indeed, masking removes all the first-order leakage and thus, obliges one to combine samples together to detect a dependency with the secret. This is usually very long as all the couples of samples (or n-tuple) have to be tested.

It is thus a great challenge for MINE to see if it is able to automatically perform this recombination, and detect higher-order leakages. For that purpose, the public dataset ASCAD [2] (with no jitter) has been used. It provides a database of 50k EM traces of 700 samples each, of an AES protected with a Boolean masking. A MI estimation, derived from deep learning attack results, has already been done on this dataset [14]. They reported a MI of 0.065 bit between the traces and the third key byte (the first two were not masked) which provides a reference point.

At first, MINE was not successful: the loss function increased but the validation started to decrease very early which is a direct sign of overfitting. Intuitively, when the underlying problem is more complex, it may be easier for the network to learn properties of the empirical data before the true structure of these data. Then, classical solutions against overfitting have been applied to MINE. These include Batch Normalization (BN) layers, dropout, and regularization techniques. While the last, did not impact performances significantly, the combination of BN and dropout greatly improved the results. A BN layer has been applied to the inputs in order to normalize them. This is known to make the loss function smoother and thus the optimization easier [17]. Dropout was activated with $p = 0.2$ so that each neuron has a probability p of being set to 0 when an output of the network is computed (except when the validation is computed). This is also known to reduce overfitting and make the training more robust [20].

Results are presented in Fig. 8: validation loss reached a value of 0.2 bits which is about three times bigger than the MI reported in [14]. Due to how validation is computed this value can not be an overestimation of the MI. Our MLP structure may be a little more adapted than the CNN used in [14] as there is no jitter in that case. We also suggest that the input decompression technique, only usable with MINE, could help the network to learn, especially for complex problems such as when the algorithm is masked. This may explain why MINE was able to extract more information in that case. One can observe that it took 100 epochs for the network to start to learn something. It may seem random but this period of 100 epochs was surprisingly repeatable across experiments.

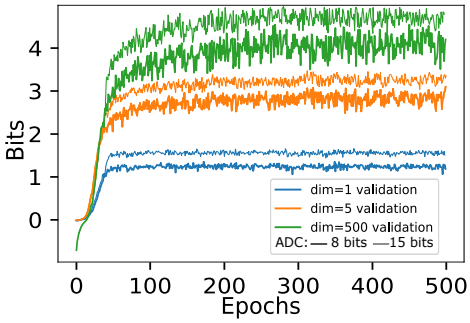


Fig. 7: Leakage evaluation of an unprotected AES

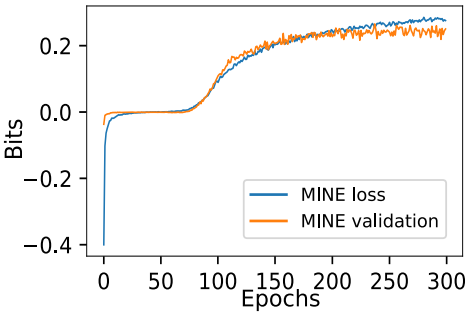


Fig. 8: Leakage evaluation of a masked AES (ASCAD)

4.3 Instructions leakage

Another advantage of MINE is that it cannot only compute MI for high dimensional traces but also for secrets with a high number of classes. This is the case if one is interested in recovering information about the raw assembly instructions that are being executed. This branch of SCA is called Side Channel Based Disassembling (SCBD) [7–9, 22] and the main difficulty in this domain is the size of the attacked variable, generally the couple (opcode, operands) which is no longer a simple byte. For example the target device in [7,8,22] is a PIC16F from Microchip which encodes its instruction on 14 bits. Even though some opcodes are not valid, the number of possible couples (opcode, operands) is around 2^{12}. It is even worse for more complex processors encoding their instruction on 16 or 32 bits. MINE treats the attacked variable as an input and the number of neurons used to encode it can be adjusted with ID as stated in section 3.2. Using base 2, one only need 14 neurons to encode an instruction in the PIC example.

In order to test MINE in this context, we have generated a program with 12k randoms instructions for the PIC. Using the same experimental setup described in section 4.2 of [7], an EM trace of the whole execution has been acquired (it was averaged on 500 traces as the program is repeatable). This trace has then been separated into 12k sub-traces of 2000 samples each. Each sub-trace was labeled with the executed instruction. As it has been shown in [7] that the probe position may be very important, MINE has been applied at 100 different positions (using a (10×10) grid) resulting in the MI cartography given in Fig. 9. The value at each position is the mean of the network's validation computed over the 100 last epochs of the training (all the training lasted 500 epochs and a Gaussian filter has been applied to the figure). Up to 8 bits of information have been found for the best positions which is a high amount if one compares to the full entropy of an instruction which is approximately 12 bits. This shows that MINE stays sound even when the target variable has a high number of classes.

Coil comparison. Similar to what has been done regarding the selection of the oscilloscope precision, another hardware comparative experiment was conducted. Two probes with two different coil orientations (Langer ICR HH and HV 100-27) have been used. While the "hot" zones are globally the same, one may observe that the coil orientation may have a significant impact on the captured information for some specific positions. This experiment suggests that MINE could be used to guide the positioning of EM probes during evaluations.

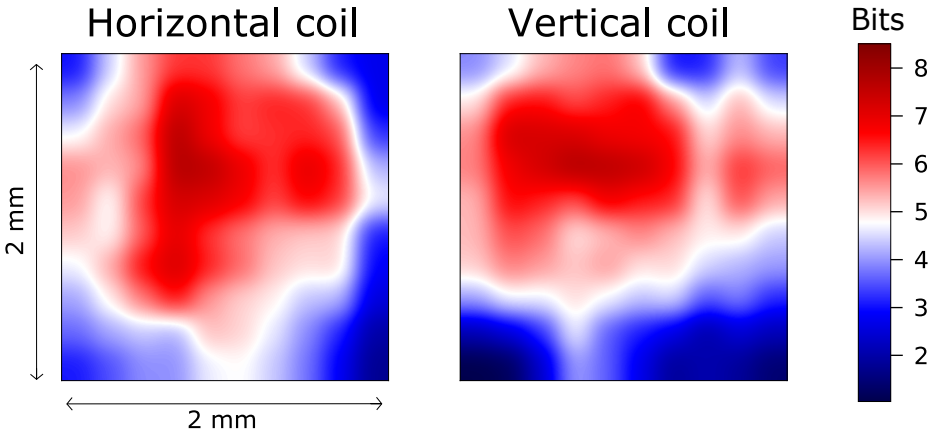


Fig. 9: Cartography of the MI between instructions and traces estimated by MINE on a PIC16F

5 Conclusion

This paper suggests ways MINE, a new deep learning technique to estimate mutual information, could constitute a new tool for side-channel analysis. The main advantage is its ability to estimate MI between high dimensional variables. Indeed, being able to consider full (or large part of) traces as a variable, allows to exploit all potential leakage sources with no *a priori* on the leakage model neither on the implementation. It seems that MINE could be used as a very simple tool to obtain an objective leakage evaluation from traces. Thus, it may be employed for massive and quick evaluations for designers in their development process as well as for evaluators as a first leakage metric.

These suggestions result from a theoretical and practical analysis of MINE in a side-channel context. MINE's overfitting problem has been deeply investigated as well as the way input representation may have a huge impact on performances. Our upcoming works will aim at investigating possible usages of MINE for extracting secrets in an unsupervised way *i.e.* in an attack context.

References

1. Belghazi, M.I., Baratin, A., Rajeswar, S., Ozair, S., Bengio, Y., Courville, A., Hjelm, R.D.: Mine: Mutual information neural estimation (2018)

2. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ascad database. ANSSI, France & CEA, LETI, MINATEC Campus, France. (2018)
3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
4. Bronchain, O., Hendrickx, J.M., Massart, C., Olshevsky, A., Standaert, F.X.: Leakage certification revisited: Bounding model errors in side-channel security evaluations. Cryptology ePrint Archive, Report 2019/132 (2019)
5. Bronchain, O., Standaert, F.X.: Side-channel countermeasures' dissection and the limits of closed source security evaluations. Cryptology ePrint Archive (2019)
6. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International Workshop on Cryptographic Hardware and Embedded Systems (2002)
7. Cristiani, V., Lecomte, M., Hiscock, T.: A Bit-Level Approach to Side Channel Based Disassembling. In: CARDIS 2019. Prague, Czech Republic (Nov 2019), https://hal.archives-ouvertes.fr/hal-02338644
8. Eisenbarth, T., Paar, C., Weghenkel, B.: Building a side channel based disassembler. In: Transactions on computational science (2010)
9. Goldack, M., Paar, I.C.: Side-channel based reverse engineering for microcontrollers. Master's thesis, Ruhr-Universität Bochum, Germany (2008)
10. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2014)
11. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual International Cryptology Conference (1999)
12. Kraskov, A., Stögbauer, H., Grassberger, P.: Estimating mutual information. Physical Review (Jun 2004)
13. Macé, F., Standaert, F.X., Quisquater, J.J.: Information theoretic evaluation of side-channel resistant logic styles. vol. 2008, p. 5 (01 2008)
14. Masure, L., Dumas, C., Prouff, E.: A comprehensive study of deep learning for side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020** (2019)
15. Prouff, E., Rivain, M.: Theoretical and practical aspects of mutual information based side channel analysis. pp. 499–518 (01 2009)
16. Quisquater, J.J., Samyde, D.: Electromagnetic analysis: Measures and countermeasures for smart cards (2001)
17. Santurkar, S., Tsipras, D., Ilyas, A., Madry, A.: How does batch normalization help optimization? (2018)
18. Schneider, T., Moradi, A.: Leakage assessment methodology. In: International Workshop on Cryptographic Hardware and Embedded Systems (2015)
19. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(3), 379–423 (1948)
20. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**(56), 1929–1958 (2014)
21. Steeg, G.V.: Non-parametric entropy estimation toolbox (2014), https://github.com/gregversteeg/NPEET
22. Strobel, D., Bache, F., Oswald, D., Schellenberg, F., Paar, C.: Scandalee: a side-channel-based disassembler using local electromagnetic emanations. In: Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (2015)
23. Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. Journal of Machine Learning Research **11**(95), 2837–2854 (2010)