

Improving Side-Channel Analysis through Semi-Supervised Learning

Stjepan Picek, Annelie Heuser, Alan Jovic, Karlo Knezevic, Tania Richmond

▶ **To cite this version:**

Stjepan Picek, Annelie Heuser, Alan Jovic, Karlo Knezevic, Tania Richmond. Improving Side-Channel Analysis through Semi-Supervised Learning. CARDIS 2018 - 17th Smart Card Research and Advanced Application Conference, Nov 2018, Montpellier, France. hal-02011351

HAL Id: hal-02011351

https://inria.hal.science/hal-02011351

Submitted on 7 Feb 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Improving Side-channel Analysis through Semi-supervised Learning

Stjepan Picek[1], Annelie Heuser[2], Alan Jovic[3],
Karlo Knezevic[3], and Tania Richmond[2]

[1] Delft University of Technology, Delft, The Netherlands
[2] Univ Rennes, Inria, CNRS, IRISA, France
[3] University of Zagreb Faculty of Electrical Engineering and Computing, Croatia

Abstract. The profiled side-channel analysis represents the most powerful category of side-channel attacks. In this context, the security evaluator (i.e., attacker) gains access to a profiling device to build a precise model which is used to attack another device in the attacking phase. Mostly, it is assumed that the attacker has significant capabilities in the profiling phase, whereas the attacking phase is very restricted. We step away from this assumption and consider an attacker restricted in the profiling phase, while the attacking phase is less limited. We propose the concept of semi-supervised learning for side-channel analysis, where the attacker uses a small number of labeled measurements from the profiling phase as well as the unlabeled measurements from the attacking phase to build a more reliable model. Our results show that the semi-supervised concept significantly helps the template attack (TA) and its pooled version (TA_p). More specifically, for low noise scenario, the results for machine learning techniques and TA are often improved when only a small number of measurements is available in the profiling phase, while there is no significant difference in scenarios where the supervised set is large enough for reliable classification. For high noise scenario, TA_p and multilayer perceptron results are improved for the majority of inspected dataset sizes, while for high noise scenario with added countermeasures, we show a small improvement for TA_p, Naive Bayes and multilayer perceptron approaches for most inspected dataset sizes. Current results go in favor of using semi-supervised learning, especially self-training approach, in side-channel attacks.

1 Introduction

Side-channel analysis (SCA) consists of extracting secret data from (noisy) measurements. It is made up of a collection of miscellaneous techniques, combined in order to maximize the probability of success, for a low number of trace measurements, and as low computation complexity as possible. The most powerful attacks currently known are based on a profiling phase, where the link between the leakage and the secret is learned under the assumption that the attacker knows the secret on a profiling device. This knowledge is subsequently exploited

to extract another secret using fresh measurements from a different device. In order to run such an attack, one has a plethora of techniques and options to choose from, where the two main types of attacks are based on 1) template attack (relying on probability estimation), and 2) machine learning (ML) techniques. When working with the typical assumption for profiled SCA that the profiling phase is not bounded, the situation actually becomes rather simple if neglecting computational costs. If the attacker is able to acquire an unlimited (or, in real-world very large) amount of traces, the template attack (TA) is proven to be optimal from an information theoretic point of view (see e.g., [1, 2]). In that context of unbounded and unrestricted profiling phase, ML techniques seem not needed.

Stepping away from the assumption of an unbounded number of traces, the situation becomes much more interesting and of practical relevance. A number of results in recent years showed that in those cases, machine learning techniques can actually significantly outperform template attack (see e.g., [3–5]). Still, the aforesaid attacks work under the assumption that the attacker has a large amount of traces from which a model is learned. The opposite case would be to learn a model without any labeled examples. Machine learning approaches (mostly based on clustering) have been proposed, for instance, for public key encryption schemes where only two possible classes are present – 0 and 1 – and where the key is guessed using only a single-trace (see e.g., [6]). In the case of differential attacks (using more than one encryption) and using more than two classes, to the best of our knowledge, unsupervised machine learning techniques have not been studied yet.

In this paper, we aim to address a scenario positioned between supervised and unsupervised learning, the so-called semi-supervised learning in the context of SCA. Figure 1 illustrates the different approaches of supervised (on the left) and semi-supervised learning (on the right). Supervised learning assumes that the security evaluator first possesses a device similar to the one under attack. Having this additional device, he is then able to build a precise profiling model using a set of measurement traces and knowing the plaintext/ciphertext and the secret key of this device. In the second step, the attacker uses the obtained profiling model to reveal the secret key of the device under attack. For this, he measures a new, additional set of traces, but as the key is secret, he has no further information about the intermediate processed data and thus builds hypotheses. Accordingly, the only information which the attacker transfers between the profiling phase and the attacking phase is the profiling model he builds. We note there is a number of papers considering supervised machine learning in SCA, see e.g., [7–9].

In realistic settings, the attacker is not obliged to view the profiling phase independently from the attacking phase. He can rather combine all available resources to make the attack as effective as possible. In particular, he has at hand a set of traces for which he precisely knows the intermediate processed states (i.e., labeled data) and another set of traces with a secret unknown key and thus no information about the intermediate variable (i.e., unlabeled data). To take advantage of both sets at once, we propose a new strategy of conducting profiled side-channel analysis to build a more reliable model (see Figure 1 on

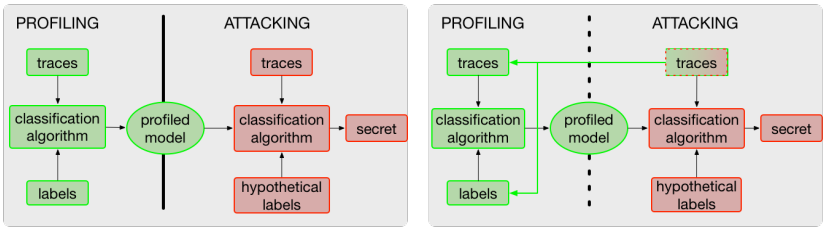


Fig. 1: Profiling side-channel scenario: traditional (left), semi-supervised (right).

the right). This new view is of particular interest when the number of profiling traces is (very) low, and thus any additional data is helpful to improve the model estimation.

To show the efficiency and applicability of semi-supervised learning for SCA, we conduct extensive experiments where semi-supervised learning outperforms supervised learning if certain assumptions are satisfied. More precisely, the results show a number of scenarios where guessing entropy on the test set is significantly lower when semi-supervised learning is used (when compared to the "classical" supervised approach). We start with the scenario that we call "extreme profiling", where the attacker has only a very limited number of traces to learn the model. From there, we increase the number of available traces, making the attacker more powerful, until we reach a setting where there is no more need for semi-supervised learning. Still, even when the supervised learning works good (i.e., succeeds in breaking an implementation), we can observe a number of scenarios where semi-supervised learning can still improve the results or at least not deteriorate them.

To the best of our knowledge, the only work up till now implementing a semi-supervised analysis in SCA is [10], where the authors conclude that the semi-supervised setting cannot compete with a supervised setting. Unfortunately, the assumed scenario is hard to justify and consequently their results are expected (but without much implication for SCA). More precisely, the authors compared the supervised attack that has more available measurements (and corresponding labels) than the semi-supervised attack. On the basis of such experiments, they concluded that the supervised attack is better, which is intuitive and straightforward. A proper comparison would be between the supervised attack that has at most the same number of labeled measurements as the semi-supervised one. Additionally, our analysis is not restricted to only one labeled class in the learning phase.

Note, we primarily focus on improving the results if the profiling phase is limited. Since we are considering extremely difficult scenarios, the improvements one can realistically expect are often not too big. Still, we consider any improvement to be relevant since it makes the attack easier, while not requiring any additional knowledge or measurements.

2 Semi-supervised Learning Types and Notation

Semi-supervised learning (denoted as SSL in the rest of the paper) is positioned in the middle between supervised and unsupervised learning. There, the basic idea is to take advantage of a large quantity of unlabeled data during a supervised learning procedure [11]. This approach assumes that the attacker is able to possess a device to conduct a profiling phase but has limited capacities. This may reflect a more realistic scenario in some practical applications, as the attacker may be limited by time, resources, and also face implemented countermeasures which prevent him from taking an arbitrarily large amount of side-channel measurements, while knowing the secret key of the device.

Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be a set of n samples where each sample x_i is assumed to be drawn i.i.d. from a common distribution $\mathcal{X}$ with probability $P(x)$. This set $\boldsymbol{x}$ can be divided into three parts: the points $\boldsymbol{x}_l = (x_1, \ldots, x_l)$ for which we know the labels $\boldsymbol{y}_l = (y_1, \ldots, y_l)$ and the points $\boldsymbol{x}_u = (x_{l+1}, \ldots, x_{l+u})$ for which we do not know the labels. Additionally, the third part is the test set $\boldsymbol{x}_t = (x_{l+u+1}, \ldots, x_n)$ for which labels are also not known. We see that differing from the supervised case, where we also do not know labels in the test phase, here unknown labels appear already in the training phase. As for supervised learning, its goal is to predict a class for each sample in the test set $\boldsymbol{x}_t = (x_{l+u+1}, \ldots, x_n)$. For SSL, two learning paradigms can be discussed: transductive and inductive learning [12]. In transductive learning (which is a natural setting for some SSL algorithms), predictions are performed only for the unlabeled data on a known test set. The goal is to optimize the classification performance. More formally, the algorithm makes predictions $\boldsymbol{y}_t = (y_{l+u+1}, \ldots, y_n)$ on $\boldsymbol{x}_t = (x_{l+u+1}, \ldots, x_n)$. In inductive learning, the goal is to find a prediction function defined on the complete space $\mathcal{X}$, i.e., to find a function $f : \mathcal{X} \to \mathcal{Y}$. This function is then used to make predictions $f(x_i)$ for each sample x_i in the test set. Obviously, transductive learning is easier, since no general rule needs to be inferred, and, consequently, we opt to conduct it. From the algorithm class perspective, we will use two approaches in order to achieve successful SSL, namely: self-training [12] (Section 2.1) and graph-based algorithms [12, 13] (Section 2.2).

On an intuitive level, semi-supervised learning sounds like an extremely powerful paradigm (after all, humans learn through SSL), the results show that it is not always the case. More precisely, when comparing SSL with supervised learning, it is not always possible to obtain more accurate predictions. Consequently, we are interested in the cases where SSL can outperform supervised learning. In order for that to be possible, the following needs to hold: the knowledge on $p(x)$ one gains through unlabeled data has to carry useful information for inference of $p(y|x)$. In the case where this is not true, SSL will not be better than supervised learning and can even lead to worse results. To assume a structure about the underlying distribution of data and to have useful information in the process of inference, we use two assumptions which should hold when conducting SSL [12].

Smoothness Assumption. If two points x_1 and x_2 are close, then their corresponding labels y_1 and y_2 are close. The smoothness assumption can be generalized in

order to be useful for SSL: if two points x_1 and x_2 in a high density region are close, then so should the corresponding labels y_1 and y_2 be.

Intuitively, this assumption tells us that if two samples (measurements) belong to the same cluster, then their labels (e.g., their Hamming weight or intermediate value) should be close. Note that, this assumption also implies that, if two points are separated by a low-density region, then their labels need not be close. The smoothness assumption should generally hold for SCA, as the power consumption (or electromagnetic emanation) is related to the activity of the device. For example, a low Hamming weight or a low intermediate value should result in a low side-channel measurement.

Manifold Assumption. The high-dimensional data lie on or close to a low-dimensional manifold. If the data really lie on a low-dimensional manifold, then the classifier can operate in a space of the corresponding (low) dimension. Intuitively, the manifold assumption tells us that a set of samples is connected in some way: e.g., all measurements with the Hamming weight 4 lie on their own manifold, while all measurements with the Hamming weight 5 lie on a different, but nearby, manifold. Then, we can try to develop representations for each of these manifolds using just the unlabeled data, while assuming that the different manifolds will be represented using different learned features of the data.

2.1 Self-training

In self-training (or self-learning), any classification method is selected and the classifier is trained with the labeled data. Afterward, the classifier is used to classify the unlabeled data. From the obtained predictions, one selects only those instances with the highest output probabilities (i.e., where the output probability is higher than a given threshold σ) and then adds them to the labeled data. This procedure is repeated k times.

Self-training is a well-known semi-supervised technique and one that is probably the most natural choice to start with [12]. The biggest drawback with this technique is that it depends on the choice of the underlying classifier and that possible mistakes reinforce themselves as the number of repeats increase. Naturally, one expects that the first step of self-learning will introduce errors (wrongly predicted classes). It is therefore important to retain only those instances for which the prediction probability of the class is high. Unfortunately, a very high class prediction probability (even 100%) does not guarantee that the actual class is correctly predicted. Additionally, we use adaptive threshold σ for predicted class probability, as explained in Section 4.

2.2 Graph-based Learning

In graph-based learning, the data are represented as nodes in graphs, where a node is both labeled and unlabeled example. The edges are labeled with the pairwise distance of incident nodes. If an edge is not labeled, it corresponds to the infinite distance. Most of the graph-based learning methods depend on the

manifold assumption and refer to the graph by utilizing the graph Laplacian. Let $G = (E, V)$ be a graph with edge weights given by $w : E \to \mathbb{R}$. The weight $w(e)$ of an edge e corresponds to the similarity of the incident nodes and a missing edge means no similarity. The similarity matrix W of graph G is defined as:

$$W_{ij} = \begin{cases} w(e) \text{ if } e = (i,j) \in E \\ 0 \text{ if } e = (i,j) \notin E \end{cases} \tag{1}$$

The diagonal matrix called the degree matrix D_{ii} is defined as $D_{ii} = \sum_j W_{ij}$. To define the graph Laplacian two well-known ways are to use:

- normalized graph Laplacian $\mathcal{L} = I - D^{-1/2} W D^{-1/2}$,
- unnormalized graph Laplacian $L = D - W$.

We use graph-based learning technique called label spreading that is based on normalized graph Laplacian. In this algorithm, node's labels propagate to neighbor nodes according to their proximity. Since the edges between the nodes have certain weights, some labels propagate easier. Consequently, nodes that are close (in the Euclidean distance) are more likely to have the same labels.

3 Experimental Setting

3.1 Classification Algorithms

In supervised learning, we use template attack (TA) and its pooled version (TA_p), random forest (RF), multilayer perceptron (MLP), and Naive Bayes (NB) algorithms. In the graph-based SSL, we use k-nearest neighbors (k-NN) (i.e., the method to assign labels) since it produces a sparse matrix that can be calculated very quickly. For self-training, we use Naive Bayes. In all the experiments, we use Python [14].

Template Attack The template attack (TA) relies on the Bayes theorem such that the posterior probability of each class value y, given the vector of N observed attribute values x:

$$p(Y = y | \boldsymbol{X} = \boldsymbol{x}) = \frac{p(Y = y) p(\boldsymbol{X} = \boldsymbol{x} | Y = y)}{p(\boldsymbol{X} = \boldsymbol{x})}, \tag{2}$$

where $\boldsymbol{X} = \boldsymbol{x}$ represents the event that $\boldsymbol{X}_1 = \boldsymbol{x}_1 \wedge \boldsymbol{X}_2 = \boldsymbol{x}2 \wedge \ldots \wedge \boldsymbol{X}_N = \boldsymbol{x}_N$. When used as a classifier, $p(\boldsymbol{X} = \boldsymbol{x})$ in Eq. (2) can be dropped as it does not depend on the class y. Accordingly, the attacker estimates in the profiling phase $p(Y = y)$ and $p(\boldsymbol{X} = \boldsymbol{x} | Y = y)$ which are used in the attacking phase to predict $p(Y = y | \boldsymbol{X} = \boldsymbol{x})$ [15]. Note that the class variable Y is discrete while the measurement X is continuous. So, the discrete probability $p(Y = y)$ is equal to its sample frequency where $p(X_i = x_i | Y = y)$ displays a density function. Mostly in the state of the art, TA is based on a multivariate normal distribution of the noise and thus the probability density function used to compute $p(\boldsymbol{X} = \boldsymbol{x} | Y = y)$

equals:

$$p(\boldsymbol{X} = \boldsymbol{x}|Y = y) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_y|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_y)^T \Sigma_y^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_y)}, \tag{3}$$

where $\boldsymbol{\mu}_y$ is the mean over $\boldsymbol{X}$ for $1, \ldots, D$ and Σ_y the covariance matrix for each class y. The authors of [16] propose to use only one pooled covariance matrix to cope with statistical difficulties that result into low efficiency. We will use both versions of the template attack, where we denote pooled TA attack as TA_p.

Naive Bayes The Naive Bayes (NB) classifier [17] is also based on the Bayesian rule but is labeled "Naive" as it works under a simplifying assumption that the predictor features (measurements) are mutually independent among the D features, given the class value. The existence of highly-correlated features in a dataset can influence the learning process and reduce the number of successful predictions. Also, NB assumes a normal distribution for predictor features. NB classifier outputs posterior probabilities as a result of the classification procedure [17]. The Bayes' formula is used to compute the posterior probability of each class value y given the vector of N observed feature values x.

Multilayer Perceptron The multilayer perceptron (MLP) classifier is a feed-forward artificial neural network. MLP consists of multiple layers (at least three) of nodes in a directed graph, where each layer is fully connected to the next one and training of the network is done with the backpropagation algorithm [18].

Random Forest Random forest (RF) is a well-known ensemble decision tree learner [19]. Decision trees choose their splitting attributes from a random subset of k attributes at each internal node. The best split is taken among these randomly chosen attributes and the trees are built without pruning. RF is a parametric algorithm with respect to the number of trees in the forest. It is also a stochastic algorithm, because of its two sources of randomness: bootstrap sampling and attribute selection at node splitting.

k-NN k-nearest neighbors is the basic non-parametric instance-based learning method. The classifier has no training phase; it just stores the training set samples. In the test phase, the classifier assigns a class to an instance by determining the k instances that are the closest to it, with respect to Euclidean distance metric: $d(x_i, x_j) = \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2}$. Here, a_r is the r-th attribute of an instance x. The class is assigned as the most commonly occurring one among the k-nearest neighbors of the test instance. This procedure is repeated for all test set instances.

3.2 Datasets

We use three datasets in our experiments. To test across various settings, we target 1) high-SNR unprotected implementation on a smartcard [20], 2) low-SNR unprotected implementation on FPGA, and 3) low-SNR implementation on a smartcard protected with the randomized delay countermeasure.

We do not consider the variations in the number of available points of interest (features) since in such a case, the number of scenarios would become quite large. We select 50 points of interests with the highest correlation between the class value and data set for all the analyzed data sets and investigate scenarios with a different number of classes – 9 classes and 256 classes.

Calligraphic letters (e.g., $\mathcal{X}$) denote sets, capital letters (e.g., X) denote random variables taking values in these sets, and the corresponding lowercase letters (e.g., x) denote their realizations. Let k^* be the fixed secret cryptographic key (byte) and the random variable T the plaintext or ciphertext of the cryptographic algorithm which is uniformly chosen. The measured leakage is denoted as X and we are particularly interested in multivariate leakage $\boldsymbol{X} = X_1, \ldots, X_D$, where D is the number of time samples or features (attributes) in ML terminology.

Considering a powerful attacker who has a device with knowledge about the secret key implemented, a set of N profiling traces $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is used in order to estimate the leakage model beforehand. Note that this set is multi-dimensional (i.e., it has a dimension equal to $D \times N$). In the attack phase, the attacker then measures additional traces $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_Q$ from the device under attack in order to break the unknown secret key k^*.

DPAcontest v4 The dataset provides measurements of a masked AES software implementation [20]. As the mask is known, one can easily turn it into an unprotected scenario. Though, as it is a software implementation, the most leaking operation is not the register writing, but the processing of the S-box operation and we attack the first round. Accordingly, the leakage model changes to

$$Y(k^*) = \texttt{Sbox}[P_{b_1} \oplus k^*] \oplus \underbrace{M}_{\text{known mask}} , \tag{4}$$

where P_{b_1} is a plaintext byte and we choose $b_1 = 1$. Again we consider the scenario of 256 classes and 9 classes (considering $HW(Y(k^*))$). Compared to the measurements from version 2, the model-based SNR is much higher and lies between 0.1188 and 5.8577.

Unprotected AES-128 on FPGA This dataset targets an unprotected implementation of AES-128. AES-128 core was written in VHDL in a round based architecture, which takes 11 clock cycles for each encryption. The AES-128 core is wrapped around by a UART module to enable external communication. It is designed to allow accelerated measurements to avoid any DC shift due to environmental variation over prolonged measurements. The design was implemented on Xilinx Virtex-5 FPGA of a SASEBO GII evaluation board. Side-channel traces were measured using a high sensitivity near-field EM probe, placed over a decoupling capacitor on the power line. Measurements were sampled on the Teledyne LeCroy Waverunner 610zi oscilloscope and the trace set is publicly available at https://github.com/AESHD/AES_HD_Dataset. Although the full dataset consists of 1 250 features, here we use only the 50 most important features, as selected with the Pearson correlation. A suitable and commonly used

(HD) leakage model when attacking the last round of an unprotected hardware implementation is the register writing in the last round [20] These measurements are relatively noisy and the resulting model-based SNR (signal-to-noise ratio) has a maximum value of 0.0096. As this implementation leaks in HD model, we denote this implementation as AES_HD.

Random Delay Countermeasure Dataset The third dataset uses a protected (i.e., with a countermeasure) software implementation of AES. The target smartcard is an 8-bit Atmel AVR microcontroller. The protection uses random delay countermeasure as described by Coron and Kizhvatov. The trace set is publicly available at https://github.com/ikizhvatov/randomdelays-traces [21]. Adding random delays to the normal operation of a cryptographic algorithm has an effect on the misalignment of important features, which in turns makes the attack more difficult. As a result, the overall SNR is reduced. We mounted our attacks in the Hamming weight power consumption model against the first AES key byte, targeting the first S-box operation. The dataset consists of 50 000 traces of 3 500 features each. The best 50 features were selected using Pearson correlation. For this dataset, the SNR has a maximum value of 0.0556. In the rest of the paper, we denote this dataset as the AES_RD.

3.3 Dataset Preparation

We experiment with randomly selected 50 000 measurements (profiled traces) from all three datasets. The datasets are standardized by removing the mean and scaling to unit variance. For supervised learning scenarios, the measurements are randomly divided into 1:1 ratio for training and test sets (e.g., 25 000 for training and 25 000 for testing). The training datasets are divided into 5 stratified folds and evaluated by 5-fold cross-validation procedure for appropriate parameter tuning. For semi-supervised learning scenarios, we divide the training dataset into a labeled set of size l and unlabeled set of size u, as follows:

- (100+24.9k): $l = 100$, $u = 24900 \rightarrow 0.4\%$ vs 99.6%
- (500+24.5k): $l = 500$, $u = 24500 \rightarrow 2\%$ vs 98%
- (1k+24k): $l = 1000$, $u = 24000 \rightarrow 4\%$ vs 96%
- (10k+15k): $l = 10000$, $u = 15000 \rightarrow 40\%$ vs 60%
- (20k+5k): $l = 20000$, $u = 5000 \rightarrow 80\%$ vs 20%

4 Experimental Results

In side-channel analysis, an adversary is not only interested in predicting the labels $y(\cdot, k_a^*)$ in the attacking phase but he also aims at revealing the secret key k_a^*. A common measure in SCA is the guessing entropy (GE) metric. In particular, let us assume, given Q amount of samples in the attacking phase, an attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in a decreasing order of probability with $|\mathcal{K}|$ being the size of the keyspace. So, g_1 is the most likely and $g_{|\mathcal{K}|}$ the least likely key candidate. The GE is the average position of k_a^* in

g. As SCA metric, we report the number of traces needed to reach a guessing entropy of 5. We use '–' in case this threshold is not reached within the test set.

In supervised learning, the classifiers are built on the labeled training sets and estimated on the unlabeled test set. When considering SSL, we first learn the classifiers on the labeled sets. Then, we learn with the labeled set and unlabeled set in a number of steps, where in each step, we augment the labeled set with the most confident predictions from the unlabeled set. Finally, we conduct the estimation phase on a different unlabeled set (test set).

For machine learning techniques that have parameters to be tuned, we conducted a tuning phase on the labeled sets and use such tuned parameters in consequent experimental phases. The best obtained tuning parameters, with respect to the accuracy of the classifiers, are: for k-NN with label spreading, we select k to be equal to 7, for random forest we use 200 trees, while for MLP, we use 4 hidden layers where the number of neurons per layer is $50, 30, 20, 50$, the 'adam' solver and the 'relu' activation function. For Naive Bayes, template attack, and its pooled version, there are no parameters to tune.

As already mentioned, for self-training, we use an adaptive threshold denoted σ. In the beginning, σ is set to the value of 0.99. The threshold value remains unchanged as long as there exists any instance in the unlabeled examples for which the probability of assignment to any class is higher than 0.99. When there are no such instances left, the σ value is decreased in the next iteration by 20% (i.e., to 79.2%) and the procedure is repeated for the remaining unlabeled examples for which a successful classification has not yet been made. The whole process is repeated 5 times, each time reducing the parameter by 20%. Thereafter, all the remaining instances are attributed to the class having the highest probability.

In Tables 1– 3, we give results on the number of traces needed to reach guessing entropy of 5 for the DPAcontest v4, AES_HD, and AES_RD datasets, respectively, for all methods and dataset sizes. In all scenarios where SSL gives better results than the supervised approach, we denote such results in bold formatting. Due to the lack of space, we do not give accuracy results but we note that, where SSL shows improvements, the accuracy increases up to 15%.

4.1 DPAcontest v4 Dataset Results

The results for DPAcontest v4, HW model, in Table 1 and in Figure 2a clearly show the superiority of SSL approaches compared to supervised learning for a small number of traces (i.e., 100 and 500) in the training set. The only classifier not showing improvement with the introduction of SSL approaches is RF. The best results in terms of lowest GE are achieved with the MLP classifier. Another interesting observation is that TA does not reach GE = 5 for the majority of training dataset sizes for supervised learning (see also Figure 2b), while it always reaches the goal with label spreading (LS) approach. Both for the HW model (9 classes) and for the intermediate value model (256 classes), LS method appears to provide better results in the majority of cases when compared to the self-training (ST) method. The NB classifier gives stable and favorable results both for HW and value models, comparable to MLP and TA_p.

4.2 AES_HD Dataset Results

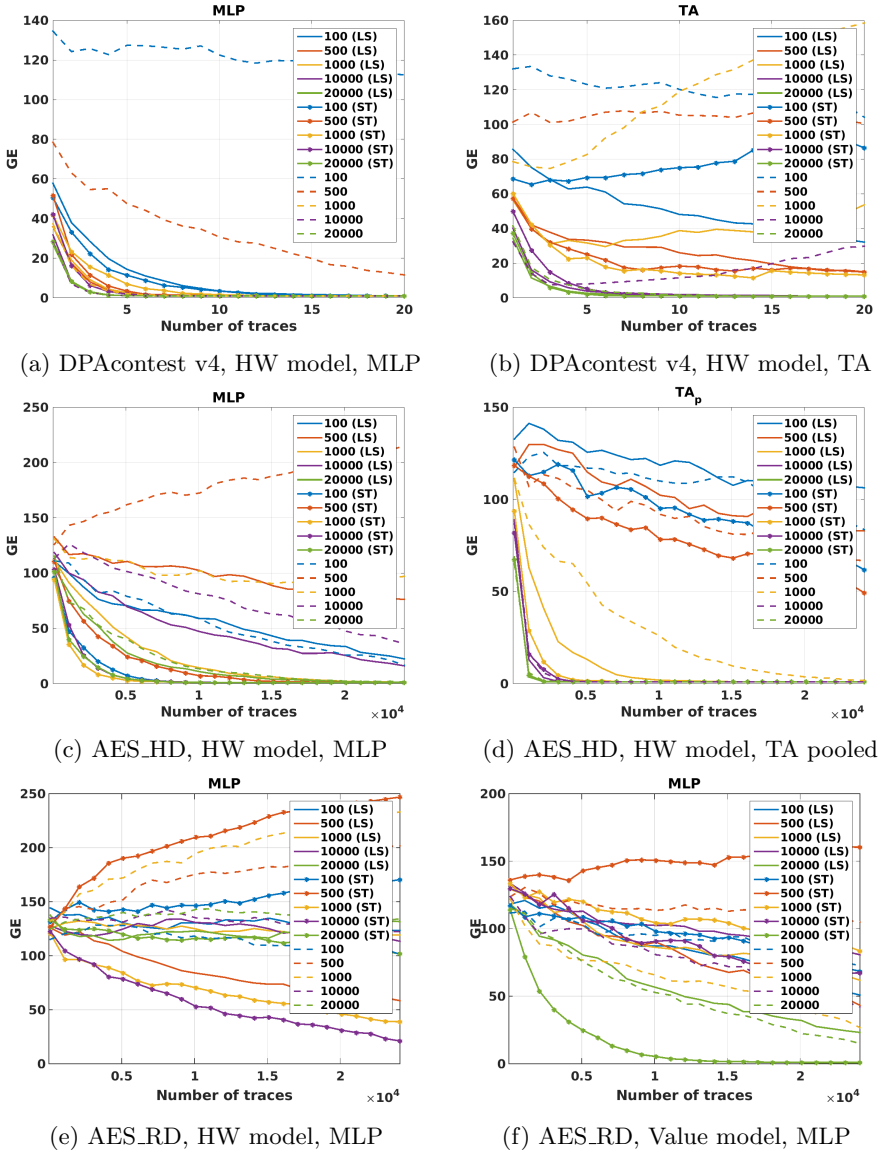
AES_HD dataset results, given in Table 2, demonstrate a highly increased number of traces needed to reach GE = 5, when compared to DPAcontest v4 dataset from Table 1. This is expected, since AES_HD contains more noise. Still, MLP, NB, and TA_p reach the designated threshold in many cases, both for HW and value models. Even when GE = 5 is not reached, from Figure 2c, it can be seen that SSL methods are quite superior to supervised learning for the majority of input dataset sizes. This is especially pronounced for the ST approach, for which all input dataset sizes surpass even the best results for supervised learning. The results for the pooled version of TA are interesting (Figure 2d), as they show a clear superiority of ST semi-supervised approach in all cases, and superiority of LS for larger dataset sizes in HW model. Intermediate value model results, given in the lower part of Table 2 show that it is difficult to reach the GE threshold in most cases and for most classifiers. Still, SSL methods give better results for the MLP and TA_p classifiers and slightly worse results on larger dataset sizes for the NB classifier.

4.3 AES_RD Dataset Results

AES_RD dataset is the most difficult dataset, considering its low signal-to-noise ratio and presence of a countermeasure. The results depicted in Table 3 point to an even higher number of traces needed to reach GE = 5, when compared to the AES_HD dataset. The benefit of using SSL methods for this dataset still exists, but less so when compared to the other two datasets. For example, NB classifier reaches the threshold for the dataset size of 1000 instances only for ST, then, for 10k instances, the best results are achieved with supervised learning, while for 20k instances, again ST is superior to both LS and supervised learning. From Figure 2e, it can be seen that, in most cases, SSL methods using MLP are better than their supervised counterparts. Also, ST method for the larger number of traces appears to be superior to the other approaches. For the intermediate value model on this dataset (Table 3 below and Figure 2f), the only clear benefit of using SSL approaches is for large dataset sizes (especially 20k for ST approach), which suggests that too much noise does not allow for efficient modeling (either for supervised or for SSL approaches), when the sample sizes are low.

5 Conclusions and Future Work

Previously, in the SCA community, profiled side-channel analysis has been considered as a strict two-step process, where only the profiled model is transferred between the two phases. Here, we explore the scenario where the attacker is more restricted in the profiling phase but can use additional available information given from the attacking measurements to build the profiled model. Two approaches to SSL have been studied in scenarios with low noise/high noise/high noise with countermeasures, 9/256 classes for prediction, and a different number



(a) DPAcontest v4, HW model, MLP

(b) DPAcontest v4, HW model, TA

(c) AES_HD, HW model, MLP

(d) AES_HD, HW model, TA pooled

(e) AES_RD, HW model, MLP

(f) AES_RD, Value model, MLP

Fig. 2: Guessing entropy results.

Table 1: Test set results, supervised learning vs. semi-supervised learning approaches, DPAcontest v4, number of traces to reach GE = 5 (– if not reached).

Size	TA	TA_p	MLP	NB	RF
	9 classes, supervised learning/SSL:self-training/SSL:label spreading				
100+24.9k	–/–/**72**	9/9/**7**	–/**9**/**9**	14/**8**/**7**	17/936/47
500+24.5k	–/**86**/**40**	4/4/4	28/**5**/**4**	5/6/**4**	11/207/16
1k+24k	–/**37**/–	4/5/4	4/6/4	4/7/5	13/375/13
10k+15k	–/**6**/**5**	3/4/3	3/4/3	5/5/**4**	13/37/**12**
20k+5k	5/**4**/**4**	3/3/3	3/3/3	4/4/4	12/17/12
25k	5	3	3	5	11
	256 classes, supervised learning/SSL:self-training/SSL:label spreading				
100+24.9k	–/–/–	–/–/–	–/**152**/**75**	–/**313**/**72**	–/–/–
500+24.5k	–/–/–	28/–/55	–/**10**/**17**	143/**76**/**30**	–/–/–
1k+24k	–/–/–	5/–/11	–/**4**/**7**	30/**21**/**10**	–/–/–
10k+15k	–/–/**973**	2/–/2	2/2/2	2/3/2	–/–/–
20k+5k	–/–/**8**	2/–/2	1/1/1	2/2/2	–/–/–
25k	5	2	1	2	–

Table 2: Test set results, supervised learning vs. semi-supervised learning approaches, AES_HD, number of traces to reach GE = 5 (– if not reached).

Size	TA	TA_p	MLP	NB	RF
	9 classes, supervised learning/SSL:self-training/SSL:label spreading				
100+24.9k	–/–/–	–/–/–	–/**5756**/–	12777/**6091**/–	23077/–/–
500+24.5k	–/–/–	–/–/–	–/**12286**/–	5126/8710/22868	–/–/–
1k+24k	–/–/–	18568/**2952**/**7015**	–/**4207**/**16070**	1913/3437/15308	15896/21812/–
10k+15k	–/**6242**/–	2148/2397/**1615**	–/**4918**/–	1111/3010/1315	–/**16705**/–
20k+5k	–/**5688**/–	1183/**962**/**963**	15775/**4947**/**14094**	893/1953/1034	–/**10689**/–
25k	–	1099	14693	952	–
	256 classes, supervised learning/SSL:self-training/SSL:label spreading				
100+24.9k	–/–/–	–/–/–	–/–/–	–/–/–	–/–/–
500+24.5k	–/–/–	–/–/–	–/–/–	–/–/–	–/–/–
1k+24k	–/–/–	–/–/–	–/–/–	–/–/–	–/–/–
10k+15k	–/–/–	–/**21470**/**17896**	–/**12385**/**14937**	6872/9270/7844	–/–/–
20k+5k	–/–/–	–/–/**19951**	19887/**9860**/**8745**	4330/5663/4601	–/–/–
25k	–	–	–	18104	4001

of measurements in the profiling phase. As side-channel attack techniques, we use three ML methods (multilayer perceptron with four hidden layers, Naive Bayes, and random forest), template attack, and its pooled version. The obtained results show that SSL is able to help in many scenarios. Significant improvements are achieved for almost all classifiers, including template attack in the low noise scenario for the small number of samples in the learning dataset. Also, template attack was improved for the majority of dataset sizes using SSL methods. It is shown that the higher the number of samples in the profiling phase, the less influential are the added unlabeled samples from the attacking phase. When the noise level is higher, SSL methods still show superiority over supervised learning approaches for the majority of dataset sizes and when using most classifiers. The improvements are smaller since those scenarios are, in general, much more difficult to attack. For the AES_RD dataset, which has a significant amount of noise and a random delay countermeasure, a clear benefit of using SSL methods may be established only for 9-classes HW model, while for 256 classes model, both supervised learning and SSL methods perform similarly. In general, when averaged over all considered scenarios, MLP classifier demonstrates the best results, followed by TA_p, and NB. Regarding the SSL method of choice, it appears that

Table 3: Test set results, supervised learning vs. semi-supervised learning approaches, AES_RD, number of traces to reach GE = 5 (– if not reached).

Size	TA	TA_p	MLP	NB	RF
	9 classes, supervised learning/SSL:self-training/SSL:label spreading				
100+24.9k	–/–/–	–/–/–	–/–/–	–/–/–	–/–/–
500+24.5k	–/–/–	–/–/–	–/–/–	–/–/–	–/–/–
1k+24k	–/–/–	–/–/–	–/–/–	–/**18484**/–	–/–/–
10k+15k	–/–/–	16918/–/20325	–/–/–	14329/–/21356	–/–/–
20k+5k	–/–/–	11735/**10846**/**11475**	–/–/–	15266/**12785**/15504	15539/20943/17944
25k	–	11139	—	15231	19734
	256 classes, supervised learning/SSL:self-training/SSL:label spreading				
100+24.9k	–/–/–	–/–/–	–/–/–	–/–/–	–/–/–
500+24.5k	–/–/–	–/–/–	–/–/–	–/–/–	–/–/–
1k+24k	–/–/–	–/–/–	–/–/–	–/–/–	–/–/–
10k+15k	–/–/–	–/–/–	–/–/–	–/–/–	–/–/–
20k+5k	–/–/–	–/–/–	–/**10210**/–	15560/**13387**/18230	–/–/–
25k	–	–	–	14860	–

self-training is better in the majority of cases when compared to label spreading. Still, for the low noise dataset scenario, label spreading may be used instead.

As a future work, we will concentrate on datasets with countermeasures since that setting seems to be the most problematic for SSL. A second research direction would be to consider not only those measurements with the highest probabilities but also to use the distribution of probabilities from the SSL learning. Finally, in a real-world scenario, two different devices should be considered, which may result in (slightly) different distributions (see e.g., [22, 23]).

References

1. Heuser, A., Rioul, O., Guilley, S.: Good is Not Good Enough — Deriving Optimal Distinguishers from Communication Theory. In Batina, L., Robshaw, M., eds.: CHES. Volume 8731 of Lecture Notes in Computer Science., Springer (2014)
2. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.: Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In: COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers. (2015) 20–33
3. Heuser, A., Zohner, M.: Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Schindler, W., Huss, S.A., eds.: COSADE. Volume 7275 of LNCS., Springer (2012) 249–264
4. Picek, S., Heuser, A., Guilley, S.: Template attack versus bayes classifier. Journal of Cryptographic Engineering **7**(4) (Nov 2017) 343–351
5. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Preprocessing. In Fischer, W., Homma, N., eds.: CHES 2017, Taipei, Taiwan, 2017, Proceedings. Volume 10529 of LNCS., Springer (2017) 45–68
6. Heyszl, J., Ibing, A., Mangard, S., Santis, F.D., Sigl, G.: Clustering Algorithms for Non-Profiled Single-Execution Attacks on Exponentiations. In: CARDIS. Lecture Notes in Computer Science, Springer (November 2013) Berlin, Germany.
7. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking Cryptographic Implementations Using Deep Learning Techniques. In Carlet, C., Hasan, M.A., Saraswat, V., eds.: SPACE 2016, Hyderabad, India, 2016, Proceedings. Volume 10076 of Lecture Notes in Computer Science., Springer (2016) 3–26

8. Picek, S., Samiotis, I.P., Kim, J., Heuser, A., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. In Chattopadhyay, A., Rebeiro, C., Yarom, Y., eds.: Security, Privacy, and Applied Cryptography Engineering, Cham, Springer International Publishing (2018) 157–176
9. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1) (Nov. 2018) 209–237
10. Lerman, L., Medeiros, S.F., Veshchikov, N., Meuter, C., Bontempi, G., Markowitch, O.: Semi-supervised template attack. In Prouff, E., ed.: COSADE 2013, Paris, France, 2013, Revised Selected Papers, Springer Berlin Heidelberg (2013) 184–199
11. Schwenker, F., Trentin, E.: Pattern classification and clustering: A review of partially supervised learning approaches. Pattern Recognition Letters **37** (2014) 4–14
12. Chapelle, O., Schlkopf, B., Zien, A.: Semi-Supervised Learning. 1st edn. The MIT Press (2010)
13. Bengio, Y., Delalleau, O., Le Roux, N.: Efficient Non-Parametric Function Induction in Semi-Supervised Learning. Technical Report 1247, Département d'informatique et recherche opérationnelle, Université de Montréal (2004)
14. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research **12** (2011) 2825–2830
15. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: CHES. Volume 2523 of LNCS., Springer (August 2002) 13–28 San Francisco Bay (Redwood City), USA.
16. Choudary, O., Kuhn, M.G.: Efficient template attacks. In Francillon, A., Rohatgi, P., eds.: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, 2013. Revised Selected Papers. Volume 8419 of LNCS., Springer (2013) 253–270
17. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An Algorithm for Finding Best Matches in Logarithmic Expected Time. ACM Trans. Math. Softw. **3**(3) (September 1977) 209–226
18. Haykin, S.: Neural Networks: A Comprehensive Foundation. 2nd edn. Prentice Hall PTR, Upper Saddle River, NJ, USA (1998)
19. Breiman, L.: Random forests. Machine Learning **45**(1) (2001) 5–32
20. TELECOM ParisTech SEN research group: DPA Contest (4^{th} edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
21. Coron, J., Kizhvatov, I.: An Efficient Method for Random Delay Generation in Embedded Software. In: Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings. (2009) 156–170
22. Renauld, M., Standaert, F., Veyrat-Charvillon, N., Kamel, D., Flandre, D.: A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices. In Paterson, K.G., ed.: Advances in Cryptology - EUROCRYPT 2011, Tallinn, Estonia, 2011. Proceedings. Volume 6632 of Lecture Notes in Computer Science., Springer (2011) 109–128
23. Choudary, O., Kuhn, M.G.: Template Attacks on Different Devices. In Prouff, E., ed.: Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers, Cham, Springer International Publishing (2014) 179–198