Improving Side-channel Analysis through Semi-supervised Learning

Stjepan Picek[1], Annelie Heuser[2], Alan Jovic[3],
Karlo Knezevic[3], Tania Richmond[2], and Axel Legay[4]

[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
[4] UCLouvain, Belgium

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 of 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 to side-channel analysis, in which the attacker uses the 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. For machine learning techniques and pooled template attack, the results are often improved when only a smaller 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.

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 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 (significantly) 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 lying 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 beforehand 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.

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 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 the accuracy on the test set is significantly higher if semi-supervised learning is used (when compared to the "clas-

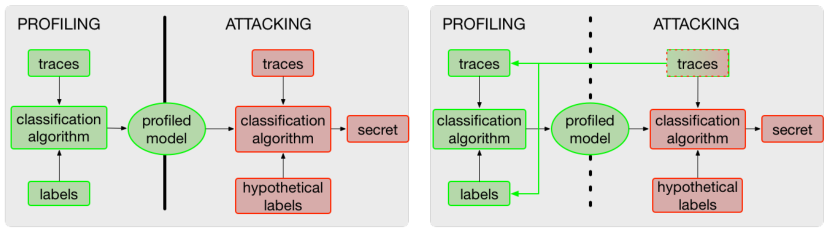

Fig. 1: Profiling side-channel scenario: traditional (left), semi-supervised (right)

sical" 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, our results show that using semi-supervised learning even in these settings is not deteriorating the efficiency of attacks.

As far as we are aware of, the only paper up till now implementing a semi-supervised analysis in SCA is [7], 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 with $n + m$ labeled traces for all classes with a semi-supervised attack with n labeled traces for one class and m unlabeled traces for the other unknown classes (i.e., in total $n + m$ traces). 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 with n traces and the semi-supervised attack with $n + m$ traces, where m is mostly smaller than n, which is the direction we take in this paper. Also, our analysis is not restricted to only one labeled class in the learning phase.

We primarily focus on improving the accuracy if the profiling phase is limited. Since we are considering extremely difficult scenarios, the improvements one can realistically expect are often not too big (i.e., in the range of only a few percents). 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 (SSL) 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 [8]. This approach assumes that the attacker is able to possess a device to conduct a profiling phase but have 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, the goal of semi-supervised learning is to predict a class for each sample in the test set $\boldsymbol{x}_t = (x_{l+u+1}, \ldots, x_n)$. For semi-supervised learning, two learning paradigms can be discussed: transductive and inductive learning [9]. In transductive learning (which is a natural setting for some semi-supervised 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 $f : \mathcal{X} \rightarrow \mathcal{Y}$. 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 whenever possible. From the algorithm class perspective, we will use two approaches in order to achieve successful semi-supervised learning, namely: self-training [9] (Section 2.1) and graph-based algorithms [9, 10] (Section 2.2).

Although, on an intuitive level, semi-supervised learning sounds like an extremely powerful paradigm (after all, humans learn through semi-supervised learning), the results show that it is not always the case. More precisely, when comparing semi-supervised learning with supervised learning, it is not always possible to obtain more accurate predictions. Consequently, we are interested in the cases where semi-supervised learning 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, semi-supervised learning 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 semi-supervised learning [9].

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 semi-supervised learning: 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.

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 [9]. 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. The assumption taken by the self-training algorithm is the same as the assumption taken by the underlying supervised classifier – i.e., when we use Support Vector Machines (SVM) as the classifier, then we work with the manifold assumption, while if we use Naive Bayes then we use the semi-supervised smoothness assumption (alongside the independence assumption, which is a standard for Naive Bayes).

In our experiments, we use Naive Bayes or SVM (with RBF kernel) as classifiers. The labeling threshold is set to the value obtained by cross-validation, where a ratio between training set classification accuracy and the size of the labeled samples from the unlabeled set is optimized. We repeat the labeling process as long as the classification accuracy on the testing set is increasing, or if the samples exist where the output probability of the classifier is higher than the threshold. The second readjustment is important, because we noticed that even a wrong labeling could improve the classifier generalization on the testing set.

Here, by classifier generalization, we consider how well will the classifier behave on a yet unseen dataset.

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 $\mathcal{L} = I - D^{-1/2}WD^{-1/2}$,
- 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. As the classifier within the label spreading, we use k-nearest neighbors (k-NN) (i.e., the technique how to assign labels) since it produces a sparse matrix that can be calculated very quickly. 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. We use label spreading as implemented in Python [11], but we wrote a custom wrapper around it in order to better suit our requirements. There, instead of using all measurements obtained from semi-supervised learning, we use only those samples that have the highest classification probabilities (similar to self-training).

3 Experimental Setting

3.1 Classification algorithms

We use template attack and its pooled version, Support Vector Machines (SVM), and Naive Bayes (NB) algorithms. Those algorithms are used both in supervised and in semi-supervised scenarios. Table 1 presents the time and space complexities for classification algorithms we use.

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 $p(Y = y|\boldsymbol{X} = \boldsymbol{x})$. 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 [12] 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 [13] 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 [13]. The Bayes’ formula is used to compute the posterior probability of each class value y given the vector of N observed feature values x.

Support Vector Machines Support Vector Machine (SVM) is a kernel based machine learning technique used to accurately classify both linearly separable and linearly inseparable data [14]. The SVM algorithm is parametric and deterministic. The basic idea when the data are not linearly separable is to transform them to a higher dimensional space by using a transformation kernel function. In this new space, the samples can usually be classified with a higher accuracy. Many types of kernel functions have been developed, with the most used ones being polynomial and radial-based.

Table 1: Time and space complexities. N is the number of samples in the training set, M is the number of samples in the test set, D is the number of attributes, $|\mathcal{Y}|$ is the number of classes of the target attribute, and v is the average number of values for a feature.

Alg.	Training		Testing	
	Time	Space	Time	Space
TA	$O(ND^2)$	$O(\|\mathcal{Y}\|D^2v)$	$O(\|\mathcal{Y}\|D^2)$	$O(\|\mathcal{Y}\|D^2v)$
k-NN	$O(1)$	$O(ND)$	$O(M(ND+kN))$	$O(ND+MD)$
NB	$O(ND)$	$O(ND)$	$O(\|\mathcal{Y}\|D)$	$O(MD)$
SVM	$O(N^3D)$	$O(N^2D)$	$O(MND)$	$O(N^2D)$

3.2 Datasets

We use two datasets that mainly differ in the amount of noise and the side-channel leakage distribution – DPAcontest v2 [15] and DPAcontest v4 [16]. 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 machine learning 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 v2 [15] DPAcontest v2 provides measurements of an AES hardware implementation. Previous works showed that the most suitable leakage model (when attacking the last round of an unprotected hardware implementation) is the register writing in the last round, i.e.:

$$Y(k^*) = \underbrace{\texttt{Sbox}^{-1}[C_{b_1} \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{C_{b_2}}_{\text{ciphertext byte}} , \tag{4}$$

where C_{b_1} and C_{b_2} are two ciphertext bytes, and the relation between b_1 and b_2 is given through the inverse ShiftRows operation of AES. In particular, we choose $b_1 = 12$ resulting in $b_2 = 8$ as it is one of the easiest bytes to attack[5]. In Eq. (4) $Y(k^*)$ consists in 256 values, as an additional model we applied the Hamming weight (HW) on this value resulting in 9 classes. These measurements are relatively noisy and the resulting model-based signal-to-noise ratio $SNR = \frac{var(signal)}{var(noise)} = \frac{var(y(t,k^*))}{var(x-y(t,k^*))}$, lies between 0.0069 and 0.0096. We use the measurements from the "template" part of the database.

DPAcontest v4 [16] The 4th version provides measurements of a masked AES software implementation. However, 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

$$Y(k^*) = \texttt{Sbox}[P_{b_1} \oplus k^*] \oplus \underbrace{M}_{\text{known mask}} , \quad (5)$$

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.

3.3 Dataset Preparation

We experiment with randomly selected 20 000 measurements (profiled traces) from DPAcontest v2 and DPAcontest v4 datasets. These measurements are divided into 2:1 ratio for training and testing sets (i.e., 13 000 in total for training with or without semi-supervised learning and 7 000 for testing). When using supervised learning, the training datasets are divided into 10 stratified folds and evaluated by 10-fold cross-validation procedure. For semi-supervised learning, we divide the training dataset into a labeled set of size l and unlabeled set of size u, as follows:

- (100+12.9k): $l = 100$, $u = 12\,900 \rightarrow 0.77\%$ vs 99.23%
- (250+12.75k): $l = 250$, $u = 12\,750 \rightarrow 1.93\%$ vs 98.07%
- (500+12.5k): $l = 500$, $u = 12\,500 \rightarrow 3.85\%$ vs 96.15%
- (1k+12k): $l = 1\,000$, $u = 12\,000 \rightarrow 7.69\%$ vs 92.31%
- (3k+10k): $l = 3\,000$, $u = 10\,000 \rightarrow 23.08\%$ vs 76.92%
- (5k+8k): $l = 5\,000$, $u = 8\,000 \rightarrow 38.46\%$ vs 61.54%
- (7k+6k): $l = 7\,000$, $u = 6\,000 \rightarrow 53.85\%$ vs 46.15%
- (10k+3k): $l = 10\,000$, $u = 3\,000 \rightarrow 76.92\%$ vs 23.08%

4 Experimental Results

As the main performance measure, we use the accuracy, i.e., the percentage of correctly classified instances: $ACC = \frac{TP+TN}{TP+FP+TN+FN}$.

[5] see e.g., in the hall of fame on [15]

In supervised learning, the classifiers are built on the labeled sets and estimated on the unlabeled sets. We give results here only for the results obtained from the testing phase. When discussing semi-supervised learning, 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. Once we cannot add any more measurements, we finish the learning phase. Finally, we conduct the estimation phase on a different unlabeled set.

For machine learning techniques that have parameters to be tuned, we conducted a tuning phase on labeled sets and use such tuned parameters in consequent experimental phases. For SVM with radial kernel, we select C equal to 10 and γ equal to 0.6 for DPAcontest v4 and C equal to 2 and γ equal to 0.05 for DPAcontest v2. A low cost of the margin parameter C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. The radial kernel parameter γ defines how much influence a single training example has, where the larger γ is, the closer other examples must be to be affected. When using k-NN with label spreading, we select k to be equal to 7. Naive Bayes and template attack do not have parameters to tune.

For semi-supervised learning, we tune the σ parameter. Table 2 states all the threshold levels σ for different classifiers for all different scenarios we consider, which was set at the labeled set training accuracy. In all experiments, when there is a single result with the best accuracy, we depict it in bold style.

Table 2: Threshold levels. When considering SVM threshold level σ for DPAcontest v4, both 9 and 256 classes scenarios use the same value. This is because the problem is "simple" for 9 classes and the threshold can be set to a higher value but we noticed no difference in performance. The same behavior is not observed for DPAcontest v2.

	DPAcontest v4		DPAcontest v2	
Classifier	9 classes	256 classes	9 classes	256 classes
NB, k-NN	0.99	0.99	0.99	0.99
SVM	0.22	0.22	0.01435	0.004

4.1 DPAcontest v2 Dataset Results

In Table 3, we give the results for the testing phase for supervised learning vs. semi-supervised learning methods for DPAcontest v2 dataset.

Overall, the accuracies are low for all scenarios, which is expected due to the high amount of noise. Supervised learning results serve as a baseline when comparing with the semi-supervised learning. Using only 100 traces in the profiling phase can be considered as the worst case scenario, while using all 13 000 in the profiling phase can be considered as the best case scenario. When having an extremely small number of traces in the profiling phase, a natural assumption is that adding more measurements in the semi-supervised phase helps. Still, as we assume there will be at least some portion of measurements incorrectly classified during semi-supervised learning, we cannot expect semi-supervised learning to be more successful than supervised learning with all traces.

For the 9 classes scenario, we can notice an interesting behavior for the smaller numbers of measurements, e.g., up to 1 000 measurements with Naive Bayes and TA$_p$ techniques. We see that the accuracies are actually higher than for the cases with more measurements. Namely, since there is a very limited number of traces in the profiling phase, some of the classes do not have any correctly trained representatives. As an example, for the scenario with 100 measurements, we actually see there are no instances of HW 0, HW 1, HW 7, and HW 8 classes present in the profiling phase. Consequently, the classifiers do not work anymore with 9 classes but only with 5 classes, which makes it a much simpler classification problem. Although such results look good, they are not very helpful in the SCA context to reveal the secret key.

Table 3: Testing results, supervised learning vs. semi-supervised learning approaches, DPAcontest v2, (ACC, %).

Size	NB	SVM	TA	TA$_p$
	9 classes, supervised learning / SSL:self-learning / SSL:label spreading			
100/+12.9k	**20.6**/14.5/11.8	**21.6**/18.7/18.1	0.4/**7.3**/5.9	**17.7**/17/15.6
250/+12.75k	10.4/**12.3**/11.3	**21**/20.8/19.8	**10.2**/0.5/0.4	15.8/**16.2**/15.1
500/+12.5k	10.8/**12.9**/11.8	**22**/21.4/21	**5.5**/1.4/1.1	15.3/**17.6**/16.7
1k/+12k	11.9/**12.7**/12.3	23.8/**25.2**/25.1	**3.8**/0.4/0.4	13.8/**14.9**/13.1
3k/+10k	**7.3**/7/6.8	24.5/**25**/24.9	**8.7**/1.5/1.4	10.4/**12.4**/12
5k/+8k	9.3/**10.3**/9.4	24.6/**25.5**/25.1	0.9/**15**/14.1	8.9/**11.8**/10.9
7k/+6k	8.8/**11**/10.4	25.5/**26**/26	**2.1**/1.7/1.3	8.4/**11.1**/10.9
10k/+3k	8.8/**10.2**/10	25.3/**26.2**/26	7.5/**8.3**/6.9	8.2/**8.6**/8
13k	8.3	26.2	15	7.6
	256 classes, supervised learning / SSL:self-learning / SSL:label spreading			
100/+12.9k	0.3/**0.6**/0.5	0.4/**0.5**/0.4	0.3/**0.5**/0.3	0.4/0.4/0.4
250/+12.75k	0.4/**0.7**/0.5	0.4/0.5/0.5	**0.6**/0.5/0.4	0.4/0.4/0.4
500/+12.5k	0.4/**0.6**/0.5	0.4/**0.5**/0.4	0.4/**0.6**/0.5	0.4/0.4/0.4
1k/+12k	0.5/0.5/0.5	0.4/0.4/0.3	0.4/**0.6**/0.5	0.5/0.5/0.5
3k/+10k	0.4/0.4/0.4	0.4/**0.5**/0.4	0.3/**0.4**/0.3	0.4/0.4/0.4
5k/+8k	**0.6**/0.5/0.4	**0.5**/0.4/0.4	0.4/0.4/0.4	**0.5**/0.4/0.4
7k/+6k	**0.7**/0.6/0.5	0.5/0.6/0.6	**0.5**/0.4/0.4	0.4/0.4/0.4
10k/+3k	**0.6**/0.5/0.5	0.5/0.5/0.5	0.4/0.4/0.4	0.4/0.4/0.4
13k	0.6	0.5	0.4	0.3

The most extreme cases behave worse than supervised learning when considering ML techniques. This is somewhat expected, since the level of noise is high and it is difficult to form “good” clusters with a very small number of labeled measurements. For the cases where the amount of labeled measurements is higher than 5 000, we see improvements with SSL, which is a clear indication that more measurements are necessary for SSL if the data is noisy. Unfortunately, not all cases for TA become stable. The analysis shows that some classes are underrepresented, which makes covariance matrices unstable.

The behavior for label spreading is quite similar to that of self-training, but we see somewhat worse accuracies throughout all scenarios. A smaller number of measurements for 9 classes have higher accuracies than using more measurements, but this is due to a lack of labeled examples of all classes which correspond to an easier problem since then the classification process has fewer classes to choose from. When considering 256 classes, interestingly, TA works better with SSL than with supervised learning for a number of cases. Still, the accuracies are very low, as expected, since we work with a highly noisy scenario and many classes.

4.2 DPAcontest v4 Dataset Results

In Table 4, we give the results for the testing phase for supervised learning vs. semi-supervised learning approaches for DPAcontest v4 dataset.

Here, we see far better results compared to DPAcontest v2. We observe that for both 9 and 256 classes, machine learning techniques work well. SVM performs better than Naive Bayes, which is expected since SVM is a more powerful classification technique. When considering template attacks, we see that the pooled version is significantly better, since it does not have the problem with covariance matrices instability. As it can be seen, for 13 000 measurements and 9 classes, TA and TA pooled perform similarly, which is a strong indication that the covariance matrices got stable and that if there would be further increase in the number of measurements, TA may outperform the pooled version.

As particularly interesting, we highlight the efficiency of ML techniques even in scenarios with only 100 to 500 measurements. This leads us to the conclusion that ML is an extremely powerful option to be used even in the most extreme profiling cases, provided that the level of noise is not too high. When considering 9 classes and SSL, only for the case with 100 measurements, the results for ML are slightly worse when compared with supervised learning. The other results are either better or comparable. What is the most interesting, TA and TA_p results are significantly better than those obtained with supervised learning. In particular, the accuracy for TA and TA_p increases for all scenarios regardless of the number of added unlabeled measurements. For TA, the explanation is simple but with profound consequences: by adding more measurements, we are able to resolve instabilities in the estimation of the covariance matrices and consequently, the accuracy of TA is significantly increasing. The highest increase (more than 73.3%) can be observed for TA when using 10k labeled measurements and 3k unlabeled ones. Interestingly, we see that for TA and TA_p using 10k+3k is approximately as efficient as using 13k labeled traces. The highest increases for TA_p can be observed in the first 4 scenarios (up to 12 000 of additional

Table 4: Testing results, supervised learning vs. semi-supervised learning approaches, DPAcontest v4, (ACC, %).

Size	NB	SVM	TA	TA$_p$
	9 classes, supervised learning / SSL:self-learning / SSL:label-spreading			
100/+12.9k	**61.5**/59/30	**69.1**/69/25	0.3/**58.9**/18.8	45.4/**67.6**/21.1
250/+12.75k	64.3/64.6/**65.3**	78.4/**78.2**/77.5	0.3/12.6/**61.4**	53/**75.2**/71.3
500/+12.5k	65.9/**66.2**/65.5	82.7/**82.8**/81.1	0.3/56.6/**58.8**	68.9/**76.9**/74.5
1k/+12k	64.8/**68.1**/67.7	86.6/**87.1**/84.1	1.3/**44.2**/7.1	73.1/**78.3**/76.6
3k/+10k	67.2/68.3/**68.7**	90.8/90.5/**91.8**	5.2/53/**66.6**	74.9/**78.1**/77.4
5k/+8k	67.9/68.1/**68.8**	92/**92.3**/91.8	2.8/**46.4**/3.2	75.8/**78.4**/78
7k/+6k	68/**68.4**/68.6	**92.8**/92.7/92.5	11.2/**75.6**/14.8	76.5/**78**/77.9
10k/+3k	68.1/68.7/68.7	93.3/**93.6**/93.5	0.4/**73.8**/49.6	77.2/77.9/**78**
13k	68.4	93.7	75.3	77.7
	256 classes, supervised learning / SSL:self-learning / SSL:label-spreading			
100/+12.9k	1.5/**2.7**/1.7	**5.1**/4.2/3.7	0.3/0.3/0.3	0.4/**3.4**/2.6
250/+12.75k	2.2/**3.1**/3	**6.8**/6.4/6.1	0.3/0.3/0.3	3.3/**3.7**/3.5
500/+12.5k	4.9/5.7/5.7	**10.3**/8.5/7.9	0.4/**0.5**/0.4	6.4/**7.1**/7
1k/+12k	**10.5**/9.3/8.5	**13.6**/12.8/11	0.4/**0.5**/0.4	**10.2**/9.5/9
3k/+10k	**16.5**/15.6/15	**22.4**/21.7/18.7	0.1/**0.4**/0.3	**16.3**/15.5/14.8
5k/+8k	**18**/17.3/16	**27.4**/25.7/24.8	**0.2**/0.1/0.1	**19.2**/18.7/17.2
7k/+6k	**19.5**/18.4/17	**30**/29/26.9	**0.3**/0.1/0.1	20.6/**21**/20.1
10k/+3k	**20.1**/19.6/18.1	**33.3**/32.8/28.8	0/0.2/0.2	**22.5**/22.4/21.9
13k	20.2	34.9	0.1	23.7

unlabeled measurements). Afterward, the accuracy is still higher when compared to the supervised scenario, but the margin is getting smaller. For 256 classes, the results for Naive Bayes are better for the most extreme cases (i.e., up to 500 labeled measurements) but slightly worse for the other scenarios. A similar behavior can be seen for TA$_p$. With label spreading, we see that the results are in general worse than for self-training. When considering 9 classes, the first case with 100 labeled measurements has a significant drop in accuracy when compared to supervised case or self-training. The rest of the results for 9 classes are comparable with the results obtained with self-training. What is important to notice are the cases $1k + 12k$ and $5k + 8k$, where TA is not stable, and, consequently, the results are much worse than for self-training. Scenarios with 256 classes are again similar, but we note that there are no cases where label spreading outperforms self-training.

On a more general level, one could ask if a small increase in the accuracy has a significance from a practical (attack) perspective. We believe it does, since 1) with SSL, it requires no additional knowledge except the one already used in profiling attack and 2) as shown in [17], even a small difference in accuracy can translate to a large difference is guessing entropy or success rate.

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/high noise, 9/256 classes for prediction, and a different number of measurements in the profiling phase. As side-channel attack techniques, we use two ML techniques (Naive Bayes and SVM), template attack, and its pooled version. The obtained results show that SSL is able to help in many scenarios. Significant improvements are achieved for the template attack and its pooled version in the low noise scenario. Particularly, we observed that using additional samples from the attacking phase improved the estimation of the covariance matrices which resulted in improvements of more than 70%. 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. In the scenarios with no increase in accuracy, the behavior is not significantly deteriorated compared to supervised learning (standard profiling), which makes SSL a general technique to be considered. When considering the scenario with high noise, the improvements are smaller, since those scenarios are, in general, much more difficult to attack.

As future work, we may consider the scenarios where the number of labeled traces is extremely small (100 labeled traces or less), while the number of unlabeled examples is much larger, e.g., 30 000, as we have shown that the greatest benefit of SSL is in these extreme cases. 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 SSL learning. Additionally, in the semi-supervised phase, we used ML classifiers to obtain new labeled measurements, but there is no reason why not try using TA_p attack. Consequently, we plan to investigate the scenario where TA_p attack is used as the classifier in self-training. Finally, in a real-world scenario, two different devices should be considered, which may result in (slightly) different distributions (see e.g., [18, 19]).

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. 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
5. 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
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. 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
8. Schwenker, F., Trentin, E.: Pattern classification and clustering: A review of partially supervised learning approaches. Pattern Recognition Letters **37** (2014) 4–14
9. Chapelle, O., Schlkopf, B., Zien, A.: Semi-Supervised Learning. 1st edn. The MIT Press (2010)
10. 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)
11. 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
12. 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
13. 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
14. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer-Verlag New York, Inc., New York, NY, USA (1995)
15. TELECOM ParisTech SEN research group: DPA Contest (2$^{\text{nd}}$ edition) (2009–2010) `http://www.DPAcontest.org/v2/`.
16. TELECOM ParisTech SEN research group: DPA Contest (4$^{\text{th}}$ edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
17. 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. Cryptology ePrint Archive, Report 2018/476 (2018) `https://eprint.iacr.org/2018/476`.
18. 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
19. 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