Side-Channel Analysis and Machine Learning: A Practical Perspective

Stjepan Picek*, Annelie Heuser†, Alan Jovic‡, Simone A. Ludwig§,
Sylvain Guilley¶, Domagoj Jakobovic‡ and Nele Mentens*,
*KU Leuven ESAT/COSIC and imec, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
†IRISA/CNRS, Rennes, France
‡University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
§Department of Computer Science, North Dakota State University, Fargo, ND, USA
¶TELECOM-ParisTech, Paris, France and Secure-IC S.A.S., Rennes, France

***Abstract*—The field of side-channel analysis has made significant progress over time. Side-channel analysis is now used in practice in design companies as well as in test laboratories, and the security of products against side-channel attacks has significantly improved. However, there are still some remaining issues to be solved for side-channel analysis to become more effective. Side-channel analysis consists of two steps, commonly referred to as identification and exploitation. The identification consists of understanding the leakage and building suitable models. The exploitation consists of using the identified leakage models to extract the secret key. In scenarios where the model is poorly known, it can be approximated in a profiling phase. There, machine learning techniques are gaining value. In this paper, we conduct extensive analysis of several machine learning techniques, showing the importance of proper parameter tuning and training. In contrast to what is perceived as common knowledge in unrestricted scenarios, we show that some machine learning techniques can significantly outperform template attacks when properly used. We therefore stress that the traditional worst case security assessment of cryptographic implementations, that mainly includes template attacks, might not be accurate enough. Besides that, we present a new measure called the Data Confusion Factor that can be used to assess how well machine learning techniques will perform on a certain dataset.**

I. Introduction

Any device that contains a secret, such as a cryptographic key, can be targeted by an adversary. One very powerful class of attacks are side-channel attacks (SCAs) which aim at breaking cryptographic secrets by exploiting physical information while the device is processing sensitive data [1]. For example, an adversary could monitor the running time [2], the cache behavior, the power consumption [3], and/or the electromagnetic radiation [4] of the device. The most powerful type of side-channel attacks, so-called profiled side-channel attacks, are even effective when only very few measurements are available, thanks to an additional profiling phase. Within this phase the adversary estimates advanced leakage models for targeted intermediate computations, which are then exploited to extract secret information from the device in the actual attack phase. Template attacks (TAs) [5], most of the time based on Gaussian noise assumption, are the most commonly used profiled attacks in practice; they are also known to be the most powerful from an information theoretic point of view. A variant of TAs is the stochastic attack (SA) that uses linear regression in the profiling phase [6].

Recently, another type of solutions that rely on machine learning (ML) has been investigated [7]–[10]. These related contributions highlight that ML-based side-channel attacks are effective in various experiments. However, most of the previous work has been conducted in scenarios that were more or less disadvantageous for template attacks. In particular, in [7], the authors showed that ML techniques can be advantageous when only a limited number of profiling traces are provided, rendering the estimation of probabilities in template attacks not accurate enough. This effect was also confirmed in [11]. Moreover, in [11] the authors addressed the problem of dimensionality of side-channel traces and showed that machine learning techniques are more efficient than template attacks when a higher number of useless samples are considered.

a) Our Contributions: We make several contributions in this paper. First, we show that even in unrestricted (but still practical) scenarios machine learning techniques can be more successful than template attacks. Accordingly, our work shows that the traditional template attack scenario is not enough when considering the worst case side-channel security threat. In contrast to most previous work, the attackers are provided with a sufficiently high number of traces in order to build precise leakage models. We particularly show the importance of a proper algorithm selection and parameter tuning phase for machine learning techniques. Naturally, the importance of tuning is well known, but in our opinion it has been slightly overlooked in the area of side-channel attacks. As presented here, our results show that proper parameter tuning can make a difference between an average efficient attack and attacks that even outperform template attacks.

All our experiments are conducted in practical scenarios with three levels of noise and two multi-class scenarios. We note that we use datasets that represent the de-facto standard in the side-channel community and therefore we believe they are representative for investigating the efficiency of ML methods in side-channel analysis. Next, we discuss how the number of features influences the success rate of the attack, where even a small number of features can be enough if the algorithm is properly tuned. Finally, we present a new measure that can be used to estimate the potential efficiency of machine learning approaches for SCA. This measure, the so-called Data Confusion Factor can be used to differentiate among various ML techniques but also to assess the influence of noise and the influence of class distance to the successfulness of the

algorithms. We note that this measure is designed with the specific goal of SCA analysis and is therefore not meant to be readily plugged into different domains.

b) Road Map: The paper is organized as follows. In Section II, we present basic notions on side-channel attacks. In Section III-A, we briefly discuss the datasets we use and in Section III-B we enumerate the machine learning techniques utilized in this research. The results from the tuning phase are given in Section III-C and from the testing phase in Section III-D. In Section III-E, we present the Data Confusion Factor measure and give results that support it as a relevant measure. Afterwards, we discuss the influence of the number of features in Section III-F. In Section III-G we apply our settings to a template attack. Section III-H presents a discussion about the results we obtained as well as some directions for future work. We conclude the paper in Section IV.

II. Profiled side-channel analysis

A. Background & Notations

Profiled side-channel analysis estimates the worst case security risk by considering the most powerful side-channel attacker. In particular, one assumes that an attacker is able to possess an additional device of which he has nearly full control. From this device he obtains leakage measurements and is able to control the used secret key or at least knows which one is used. Knowing the secret key enables him to calculate intermediate processed values that involves the secret key for which he is estimating models. These models can then be used in the attacking phase to predict which intermediate values are processed and therefore have conclusions about the secret key. In the following, we consider side-channel attacks on block ciphers for which a divide and conquer approach can be utilized. Note that, as there exist operations within the block cipher which manipulates each block/chunk (e.g., bytes in Advanced Encryption Standard (AES)) independently and most importantly involving only one block/chunk of the secret key, an attacker only needs to make hypotheses about the secret key block/chunk instead of the complete secret key at once.

More formally, let k^* denote (a chunk of) the fixed secret cryptographic key that is stored on the device and let t denote (a chunk of) the plaintext or ciphertext of the cryptographic algorithm. The mapping y maps the plaintext or the ciphertext $t \in \mathcal{T}$ and the key $k^* \in \mathcal{K}$ to a value that is assumed to relate to the deterministic part of the measured leakage x. For example,

$$y(t,k) = HW(\texttt{Sbox}[T \oplus k]), \tag{1}$$

where $\texttt{Sbox}[\cdot]$ is a substitution operation and HW the Hamming weight. Generally, it is assumed that f is known to the attacker. The measured leakage x can then be written as

$$x = \varphi(y(t,k^*)) + r, \tag{2}$$

where r denotes independent additive noise and where φ is a device-specific unknown deterministic function. In the sequel, we are particularly interested in multivariate leakage $\mathbf{x} = x_1, \ldots, x_D$, where D is the number of time samples, i.e., features (or attributes). Figure 1 displays one trace measurement over 600 time samples during substitution operations as in Eq. (1).

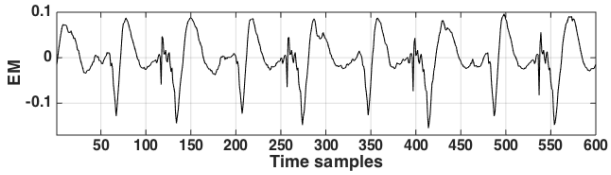


Fig. 1: Electromagnetic emanation over $D = 600$ time samples

Now, it is considered that the attacker has the following information at his disposal to conduct an attack:

- *profilng phase:* N traces $\mathbf{x}_{p_1}, \ldots, \mathbf{x}_{p_N}$, plaintexts/ciphertext $t_{p_1}, \ldots, t_{p_N}$ and the secret key k_p^*, such that he can calculate $y(t_{p_1}, k_p^*), \ldots, y(t_{p_N}, k_p^*)$.
- *attacking phase:* Q traces $\mathbf{x}_{a_1}, \ldots, \mathbf{x}_{a_Q}$ (independent from the profiling traces), plaintexts/ciphertext $t_{a_1}, \ldots, t_{a_Q}$

In the attacking phase the goal is to make predictions about $y(t_{a_1}, k_a^*), \ldots, y(t_{a_N}, k_a^*)$, where k_a^* is the secret key on the attacking device.

B. Template Attack

The template attack (TA) [5] is the most commonly used profiled side-channel attacks, which we detail in the following. The attack principle relies on the Bayes theorem, where the time sample are assumed to be not independent, we therefore deal with multivariate probability distributions. In the profiling phase the probabilities $\hat{P}(\mathbf{x}|y(t_p, k_p^*))$ for each value of $y(t_p, k)$ are estimated based on the profiling traces $\mathbf{x}_{p_1}, \ldots, \mathbf{x}_{p_N}$. In the attack phase, the attacker estimates the probability that $y(t_a, k_a^*)$ occured using the bayes theorem:

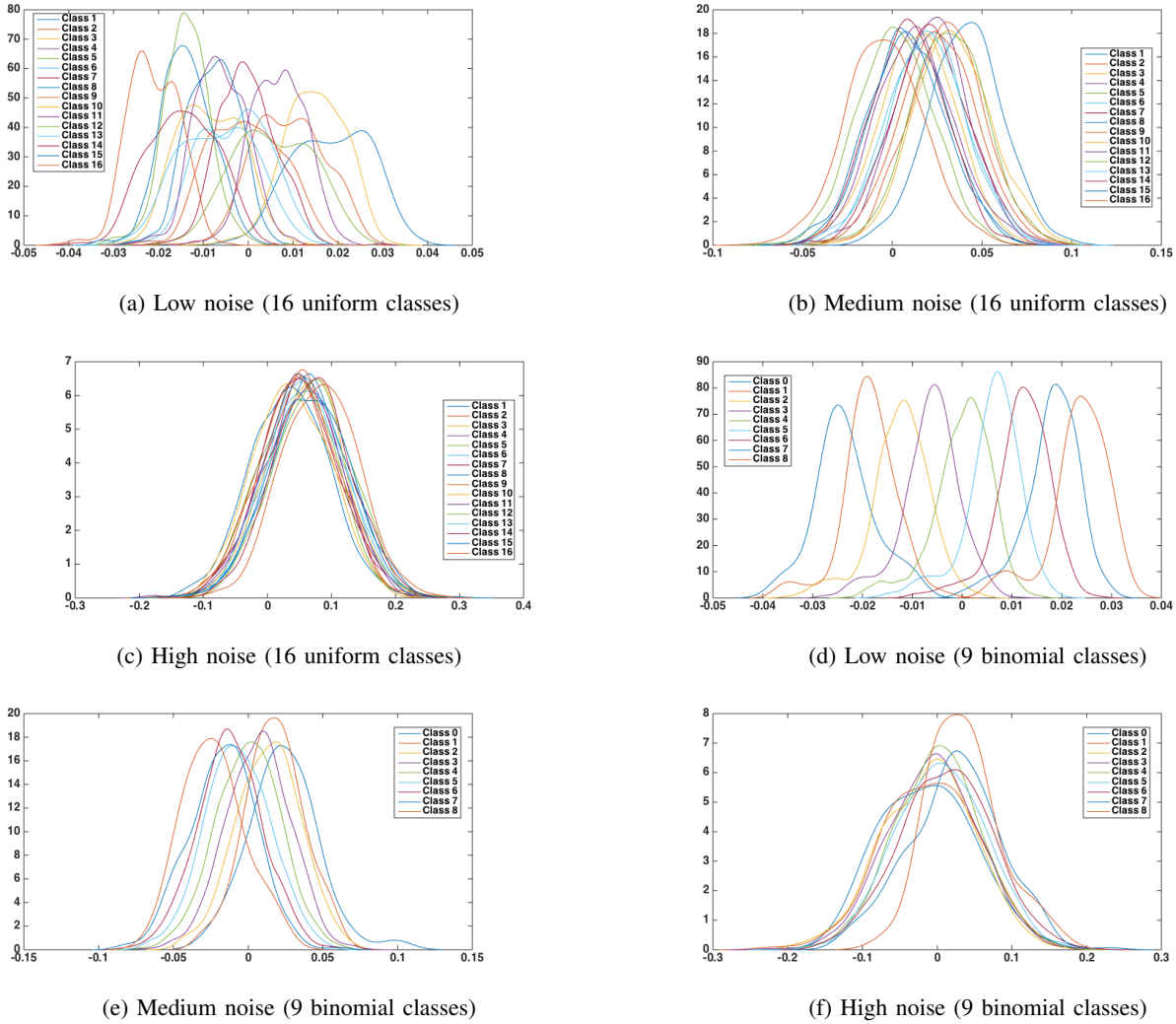
$$p(Y = y|\mathbf{X} = \mathbf{x}) = \frac{p(Y = y)p(\mathbf{X} = \mathbf{x}|Y = y)}{p(\mathbf{X} = \mathbf{x})}. \tag{3}$$

Moreover, as he knows t_a he can thus gain information about the secret key k_a^*.

A particular choice of the underlying distributions of probabilities has been made in the original TA paper and has been investigated in many follow-up papers (see e.g., [12]). In particular, it is assumed that each $P(\mathbf{x}|y(t,k))$ follows a (multivariate) Gaussian distribution and is thus parameterized by its mean and covariance matrix. The authors of [12] propose to use only one pooled covariance matrix to cope with statistical difficulties and thus a lower efficiency. Besides the standard approach, we will additionally utilize this version of the template attack in our experiments later on.

III. Experiments

In this section, we first provide an investigation of the experimental data. The second part presents results for supervised machine learning attacks in comparison to the template attack. In all experiments, we use 20 000 measurements for each scenario where those measurements are selected uniformly at random.



(a) Low noise (16 uniform classes)

(b) Medium noise (16 uniform classes)

(c) High noise (16 uniform classes)

(d) Low noise (9 binomial classes)

(e) Medium noise (9 binomial classes)

(f) High noise (9 binomial classes)

Fig. 2: Estimated densities for each class (first time instance)

A. *Experimental Data*

To test our methods, we use the publicly available traces of the DPA contest v4 [13]. The traces are captured while executing a low-cost masking protection applied on AES. However, as the mask is known, one can easily turn it into an unprotected scenario.

The original traces have a (relatively) high signal-to-noise ratio (SNR), which we label as the "low noise" scenario. Additionally, we add independent Gaussian noise to be able to investigate a medium and a high noise scenario. To calculate the SNR we use the following expression:

$$\frac{var(signal)}{var(noise)} = \frac{var(y(t,k^*))}{var(x - y(t,k^*))}. \tag{4}$$

Furthermore, as described in the next sections, we are targeting two different leakage models, which result in different class properties. For case study with 9 classes we use 50 features, while for case study with 16 classes we use 100 features. In both cases we sort them in descending order of correlation with the leakage model.

1) Uniformly distributed classes: The first case study targets a 4-bit intermediate variable that is loaded/manipulated on the device. Accordingly, we classify it into 16 different classes. Figures 2a-2c show the estimated density distributions of the first most leaking time instance, i.e., the one with the highest absolute correlation between x and $y(k^*)$. For low noise (see Figure 2a), most classes are distinguishable, whereas for high noise, most of the density distributions are completely overlapping (see Figure 2c). As expected, one can observe that the higher the added Gaussian noise, the closer the distributions become to Gaussian. The absolute value of correlation between the time instances and the intermediate value is displayed in Figure 3. For low noise, the correlation ranges between 0.65 – 0.8, whereas we have 0.2 – 0.52 for medium noise and 0.06 – 0.24 for high noise, which covers nearly the complete range. For comparison, we also calculated the SNR, which is between 12 – 2, 1 – 0.1, and 0.12 – 0.02 for low, medium, and high noise, respectively. Figure 4 shows the correlation between the

TABLE I: Parameter tuning results (accuracy) for SMO

C \ γ	0.01	0.02	0.05	0.1	0.2	0.3	0.4	0.5	0.6
1.0	69.5	72.5	77.7	81.3	85.3	86.4	88.1	88.6	89.4
2.0	72.2	75.7	79.8	83.8	87.4	87.9	88.3	88.6	89.5
5.0	76.3	78.6	82.6	86.5	89.5	89.6	90	90.1	90.2
10.0	78.2	80.1	84.8	88.3	90.2	90.2	90.4	90.4	90.6
20.0	79.3	81.7	86.6	89.4	90.8	91	91.1	91.2	91.3
30.0	80.1	82.8	87.7	89.8	91.1	91.4	91.5	91.5	91.5
40.0	82.2	85.1	86.2	90.1	90.5	91.3	91.6	91.5	91.3
50.0	87.2	88.9	89.8	90.2	91.1	91.4	91.6	91.3	91.1
60.0	88.1	90.1	90.2	90.4	91.2	91.3	91.3	91.3	91.4

first and the remaining time instances. Naturally, we have a high correlation between the time instances in the case of low noise, while the traces are nearly uncorrelated in the case of high noise.

2) Binomial distributed Hamming Weight classes: The second case study targets the Hamming Weight (HW) of an 8-bit intermediate variable resulting in 9 binomially distributed classes. In particular, we target the processing of the AES S-box in the first round, i.e., $HW(\texttt{Sbox}[t \oplus k^*])$ where t is the first plaintext byte. As before, we show the densities of each class (see Figures 2d to 2f), the correlation between the traces and the leakage model (see Figure 5), and the correlation between the first and the remaining time instances (see Figure 6). For low noise, the classes are more separable as the uniformly distributed classes. The correlation is slightly higher for low noise and lower for medium and high noise, with a SNR between 9 – 0.01, 0.9 – 0.01, and 0.12 – 0.01, respectively.

B. Supervised Machine Learning

In all machine learning experiments, we use the Weka suite [14], v3.8, with default parameters set for all the methods, unless mentioned otherwise later on. Since there are a number of machine learning approaches one could use, we opted to include the classifiers that are shown to be highly accurate on a variety of datasets [15]. Hence, in our experiments, we use Support Vector Machines (SVM, or more precisely Sequential Minimal Optimization – SMO) [16], Random Forest (RF) [17], Rotation Forest (RTF) [18], and MultiBoost (MB) algorithms [19]. We omit the details about those algorithms due to the lack of space.

C. Parameter Tuning

We divide our datasets in a ratio of 2:1 where we take the bigger set as the training set (2/3 of the data) and the smaller set for testing (1/3 of the data). On the training set, we conduct a 10-fold cross-validation with all the considered parameters and report the averaged results of individual folds. All the results in this section are presented as the accuracy (%) of the classifier. In Tables I, IIa, and IIb, we present the results for the tuning of the SVM, MultiBoost, and Rotation Forest algorithms. For Random Forest we do not conduct a parameter

it. \ subc	3	4	5
10	83.5	83.6	83.6
50	87.2	87.3	87.3
100	88.1	88.2	88
200	88.4	88.5	88.5
300	88.6	88.7	88.6

(a) Parameter tuning results (accuracy) for MultiBoost

it. \ mpg	3	4	5
50	90.4	90.7	90.4
100	90.2	90.4	90.4
150	90.3	90.4	90.6
200	90.4	90.7	90.7

(b) Parameter tuning results (accuracy) for Rotation Forest

TABLE II: Parameter tuning results for MultiBoost and Rotation Forest

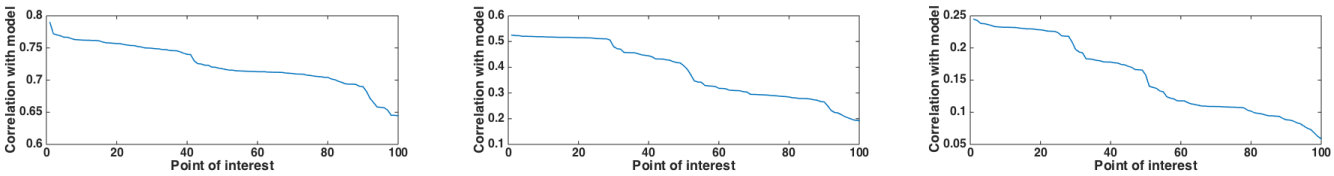
tuning, but we use a setting with 300 trees, for which we obtain a success rate of 86.5%. For each table, the best solution is highlighted using a gray background in the corresponding cell.

For the MultiBoost algorithm, we tune the iterations (it) and sub-committees ($subc$) parameters. For Rotation Forest, we tune the number of iterations (it) and the maximal size of the group of features (mpg). We use here only the radial-based SVM where the most significant parameters are the cost of the margin C and the radial kernel parameter γ. We opted to use radial-based SVM on the basis of our previous experience as well as the fact that it has only two important parameters that need to be tuned.

The best results are obtained for the SMO algorithm, but we note that this algorithm also shows significant variation in the accuracy when using different parameters highlighting the importance of a proper tuning phase On the other hand, Rotation Forest gives only slightly worse results, but we can observe that its behavior is quite robust when changing parameter values. Therefore, if one has sufficient time to conduct extensive tuning, SMO seems to be the best choice, but if there is no time for a longer tuning phase, Rotation Forest presents a viable choice. We note that these results could possibly be even further improved by a more exhaustive parameter tuning phase, where one would investigate larger sizes, but that would come with the price of significantly longer tuning for only slightly better results.

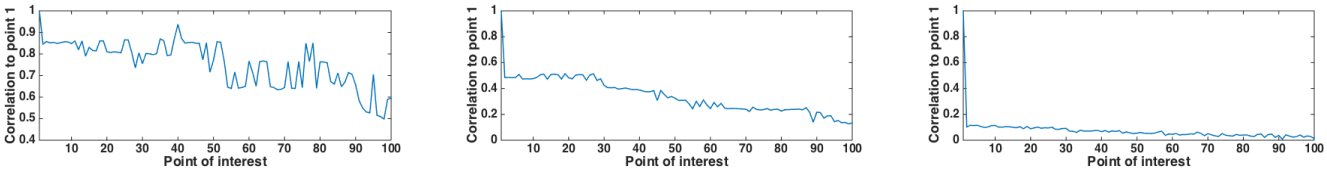
D. Testing

After we obtained the best parameter combinations in the tuning phase, we use the best settings to conduct the testing phase, with the results given in Table III. Note that we abbreviate the dataset by combining the amount of noise and the number of classes, e.g. Medium/9 denotes the dataset with 9 classes and traces with a medium level of noise.



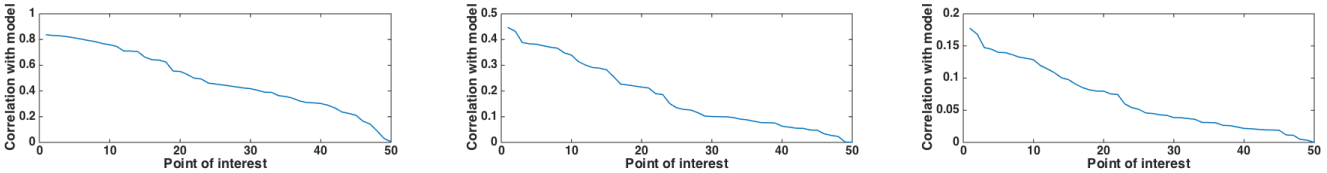
(a) Low noise (b) Medium noise (c) High noise

Fig. 3: Correlation between the traces and the leakage model (16 uniform classes)



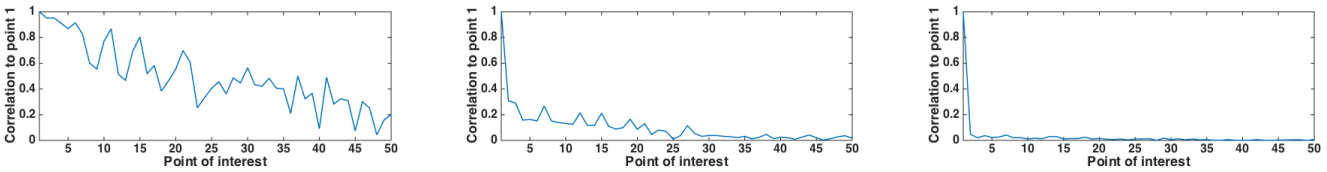
(a) Low noise (b) Medium noise (c) High noise

Fig. 4: Correlation between the first and all instances (16 uniform classes)



(a) Low noise (b) Medium noise (c) High noise

Fig. 5: Correlation between the traces and the leakage model (9 binomial classes)



(a) Low noise (b) Medium noise (c) High noise

Fig. 6: Correlation between the first and all instances (9 binomial classes)

We give the results in the form ACC/F-Measure/AUC in Table III. The first number represents the accuracy as it is the value that is most often reported. In this paper, we use the accuracy (success rate) as the primary criterion of the algorithms' successfulness, and only if such value is similar/the same for several algorithms, we consider the F-Measure and the AUC. We observe that the SMO algorithm is the best with an accuracy of 91.1% for a scenario with low noise and 9 classes. However, the difference between the SMO and the Rotation Forest algorithms is rather small and we can observe that when the level of noise is higher, e.g. medium or high, SMO even becomes the worst performing algorithm for the binomial distributed Hamming Weight classes. This behavior is also highly dependent on the distribution of the traces in the training phase, i.e., if certain classes are significantly under-represented, this can affect the final results. Note that one could add weight factors to such classes to compensate, but we leave that approach for future work.

E. Data Confusion Factor

In previous sections, we show the efficiency of ML techniques, but still that does not offer us any insight about the difficulty of those scenarios for ML nor what scenario is the most problematic for classification. Therefore, we develop a new measure that can help us to estimate the difficulty of such problems:

TABLE III: Testing results for the best algorithms' settings (ACC/F-Measure/AUC)

Algorithm	SMO	MB	RTF	RF
Low/9	91.1/91.1/97.7	88.3/88.3/97.6	90.5/90.5/98.9	86.5/86.4/98.2
Med/9	38.5/38.4/76.1	41.6/39.6/76.4	44.3/42.8/80.2	40.7/37.6/77.7
High/9	25.3/24.7/58.6	28.5/23.5/58.4	29.1/24.0/62.0	28.1/21.7/60.8
Low/16	98.9/98.9/99.6	97.2/97.2/99.6	97.1/97.1/99.7	96.2/96.2/99.6
Med/16	52.5/52.2/92.9	43.1/42.6/90.2	48.0/47.2/93	41.1/39.7/90.9
High/16	23.5/23.0/78.8	19.8/17.7/72.9	22.3/19.5/78.8	19.1/15.5/73.4

$$DCF = \frac{1}{n \times (n-1)} \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} (1 - |\frac{Avg - \#class_i}{\#all}|) \times (\frac{\#wrong_{j \in i}}{M})^{n - HD(i,j)}, \quad (5)$$

where M:

$$M = \begin{cases} all_{i+j} & \text{if } all_{i+j} \neq 0 \\ \#class_i & \text{otherwise.} \end{cases} \quad (6)$$

Here, n is the number of classes, Avg is the average number of representatives per class, $\#class_i$ is the number of instances belonging to class i, and $\#all$ is the total number of instances. The first part of the expression above serves as a corrective factor for classes for which the number of instances significantly deviates from the average. Next, $\#wrong_{j \in i}$ represents the number of instances of class j misclassified to class i and all_{i+j} is the number of instances belonging to classes i and j. $HD(i,j)$ represents the Hamming distance between classes i and j and as we see, the bigger is that difference, the larger is the influence of a certain summand. This is because the misclassified instances belonging to a class relatively far from the correct class (with respect to HD) introduce more confusion than misclassified instances belonging to neighbor classes. The DCF value assumes values in the range $[0, 1]$. In general, the higher the value of DCF, the more difficult the separation between classes and the more difficult the problem.

Note that DCF can be calculated both for supervised and unsupervised machine learning techniques. The only difference is whether the data will be calculated on the basis of testing results (predictions) for supervised learning or on the basis of clusters for unsupervised learning. There are two ways to calculate DCF; either over all pairwise comparisons for a dataset, or only between classes. In the former case, the comparison can be made only between datasets having the same number of classes, while in the latter case the comparison is always possible. The results for the former case and all scenarios we investigate here are given in Table IV. We can see that as one would expect, the higher the level of noise, the more difficult the problem for ML.

However, from these data, we can observe that for the binomial distributed Hamming Weight classes, the difference between the low noise scenario and the medium noise scenario is much more apparent than the difference between medium and high noise scenarios. This is to be expected, as the low noise scenario for Hamming Weigh classes is indeed much easier to classify when compared to medium and high noise scenarios. On the other hand, when working with uniformly distributed classes, the difference in the confusion levels for all three noise levels is much less obvious while still giving advantage to the low noise setting. The reason for the low difference is because the number of samples per class is almost equal and the only factor contributing to the difference lies in incorrect classification of the samples. Note that these findings are also confirmed in Table III. Our new metric provides interesting feedback already in the profiling phase to rate the quality of ML performance, especially with regards to the distribution of instances and their classes.

TABLE IV: Data Confusion Factors

Algorithm	SMO	MB	RTF	RF
Low/9	$6.52 \cdot 10^{-7}$	$2.58 \cdot 10^{-6}$	$4.89 \cdot 10^{-6}$	$5.79 \cdot 10^{-6}$
Medium/9	$1.59 \cdot 10^{-2}$	$1.03 \cdot 10^{-1}$	$9.58 \cdot 10^{-2}$	$1.3 \cdot 10^{-1}$
High/9	$1.01 \cdot 10^{-1}$	$2.42 \cdot 10^{-1}$	$2.28 \cdot 10^{-1}$	$2.44 \cdot 10^{-1}$
Low/16	$9.99 \cdot 10^{-6}$	$2.37 \cdot 10^{-8}$	$2.37 \cdot 10^{-8}$	$2.37 \cdot 10^{-8}$
Medium/16	$1.05 \cdot 10^{-6}$	$1.45 \cdot 10^{-5}$	$1.5 \cdot 10^{-5}$	$5.3 \cdot 10^{-5}$
High/16	$1.16 \cdot 10^{-4}$	$8.89 \cdot 10^{-4}$	$1.17 \cdot 10^{-3}$	$3.86 \cdot 10^{-3}$

F. On the Influence of the Number of Features.

Next, we take the best performing algorithm from the previous test (SMO with $C = 40$ and $\gamma = 0.4$) and we investigate how the algorithm behaves when the number of features changes. In order to do so, we use a feature selection mechanism that enables us to sort the features in accordance to their relevance. For the feature selection, we use the InfoGain method. InfoGain is a commonly used filter-based feature selection method that evaluates the worth of an attribute by measuring its information gain with respect to the class attribute [20]. In this experiment, we use 5%, 10%, 20%, 30%, 40%, 50%, and 100% of the highest ranking attributes for further analysis, and the results are shown in Table V.

The results show us that SMO is able to reach a relatively high accuracy even if it uses only 20% of the features. We note that when the level of noise is higher, the difference in results for various numbers of features that are used diminishes. This shows us that in many real-world settings where the level of noise is high, we do not need to use a high number of features to obtain a good success rate. A reason could be that simple models tend to be more successful in cases where accuracy is low or even just slightly better than guessing the majority class. Naturally, this comes under the assumption that we are indeed using the most relevant features.

TABLE V: Feature selection results (accuracy) for SMO

% of features	5	10	20	30	40	50	100
Low/9	53.4	55.3	74.7	77	80.1	83.5	91.1
Medium/9	32.9	35.5	40.6	42.1	43.6	42	38.5
High/9	27	27.7	28.4	29.2	29.1	29.3	25.3
Low/16	51.4	54.3	71.8	83.4	93.6	96.3	98.9
Medium/16	25	30.1	35.1	37.4	37.4	40.1	52.5
High/16	12.8	15.4	17.5	18.5	18.9	18.3	23.5

TABLE VI: Accuracy for the template attack (standard and pooled)

% of features	5	10	20	30	40	50	100
Low/9	16.42	31.15	64.61	67.78	74.60	73.85	70.08
Low/9 pooled	12.89	26.39	60.77	65.70	72.23	71.28	71.00
Medium/9	9.16	14.55	27.13	30.05	35.12	39.23	38.78
Medium/9 pooled	6.41	11.94	25.89	27.32	33.57	37.29	39.19
High/9	5.21	11.97	15.05	17.51	20.19	21.39	24.99
High/9 pooled	2.36	5.11	8.80	9.25	11.76	13.98	14.76
Low/16	42.01	52.72	59.19	67.00	71.55	78.12	93.47
Low/16 pooled	42.05	51.41	57.86	66.84	72.03	79.12	93.39
Medium/16	15.41	21.46	27.14	29.27	31.95	33.24	36.36
Medium/16 pooled	14.96	22.14	28.85	31.48	34.30	36.93	45.13
High/16	8.80	9.27	11.32	12.20	13.22	14.49	17.74
High/16 pooled	9.06	9.78	11.94	13.10	14.43	16.18	22.74

G. Template Attack

To be able to directly compare our results, we used the same settings as in the previous subsection, i.e., the profiling set contains $2/3$ of the data and the rest is used for classification. Table VI lists the accuracy of the standard template attack and the pooled version (see Sect. II) using 5%, 10%, 20%, 30%, 40%, 50%, and 100% of features. Interestingly, and which has not been noted yet, using 9 binomial distributed classes we observe that the pooled version is not as efficient as the standard approach, whereas for the uniformly distributed 16 classes the accuracy is increased. Compared to SMO, the accuracy of the template attack is worse for all levels of noise and number of classes. The highest difference occurs in the case of low noise with 9 classes where we have an accuracy of 91.1% for SMO and 73.44% for the template attack.

H. Discussion

In this paper, we give an extensive analysis of machine learning techniques used in side-channel analysis. Our results confirm previous work that reports the applicability of machine learning techniques, but we observe that such techniques are much more powerful than usually considered. In fact, for a number of scenarios, the machine learning approach yields the best results even compared to template attacks even when provided with a relatively high (but still practical) number of profiling traces.

Naturally, such high success rate comes with the price that one needs to investigate several ML techniques as well as to conduct a rather long parameter tuning phase. In Table I, we see how the success rate can go from 69% to more than 91%. This shows that a proper tuning phase, which has previously been omitted in the state-of-the-art, is mandatory in order to give a fair and realistic comparison between ML techniques and template attacks. Furthermore, Rotation Forest algorithm performs only slightly worse when compared to SVM, but we see that its success rate is quite consistent with different parameter values. We believe that Rotation Forest actually should be considered particularly interesting to the SCA community since it uses PCA on each subset of features. As far as we are aware, Rotation Forest was not previously investigated in SCA.

One can ask how many features are usually needed to obtain a certain success rate for ML techniques. In Section III-F, we use the best performing ML algorithm and investigate the success rate for several datasets. We note that features need to be sorted based on the InfoGain criterion before being used since otherwise it could happen that we try to conduct a classification with features that have the least information. As an interesting example, we mention the case where ML uses 5% of the features, which amounts to only 3 features. We see that even for such an extreme case, ML is able to reach a good success rate, e.g. more than 53% for dataset with low noise and 9 classes.

Another interesting fact can be observed thanks to the different investigated scenarios: (1) uniform vs (2) binomial distributed. First, we show that ML techniques are efficient in both settings. But actually what is interesting is that for all techniques (all ML, TA, and SA), the accuracy of 16 uniformly distributed classes is higher than for 9 binomial ones, even though (2) has a higher SNR (and correlation) than (1) (see Figures 3 and 5). We believe that this is mainly due to the distributions of the values. Looking at the accuracy for each class separately we observe that for (2) we have a lower accuracy for the less populated HW classes (for all techniques). To cope with this circumstance, ML even offers techniques to tune the training set further to compensate for those kind of effects, which should be even more beneficial compared to TA and SA, which is an interesting aspect for future work.

The results obtained with the Data Confusion Factor measure can offer additional level of confidence in the performance of ML techniques. Moreover, this measure fills the gap since it clearly quantifies the difficulty of the problem with regards to the level of noise and the number of classes. The measure has two contributing parts. The first part penalizes the data sets that have classes with a larger number of instances than the average class in the set as these are more difficult for the classifiers to handle, while the second part deals with instances that are wrongly classified. As a future work, we plan to explore the direct connection between DCF and the success rate of side-channel attacks. Furthermore, we plan to extend the measure

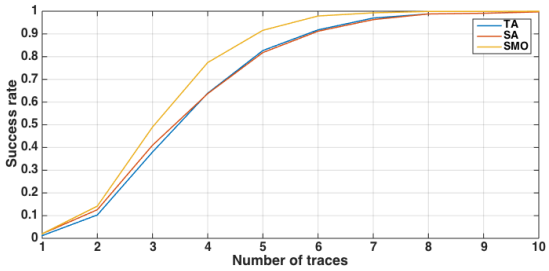


Fig. 7: Success rate for the low noise scenario using 9 HW classes

to be able to compare the results among different number of classes scenarios.

The next goal will be to link the accuracy we achieved in Table V to the rate of success of finding the correct secret key. Similar to the template attack, the presented ML techniques can be adapted to output not only the most likely class, but also the probabilities for each class. Accordingly, using this feature and the maximum-likelihood approach, one can easily extend the ML techniques to efficiently guess the secret key. Figure 7 plots the success rate of the low noise scenario for 9 HW classes using 50 points in time resulting in over 10 000 experiments. One can see that SMO is more efficient than TA and SA, where the difference between TA and SA is rather small. Recall that we achieved an accuracy of 91.1%, 73.44%, and 72.48% for SMO, TA, and SA, respectively. Needless to say, with low noise all three methods reveal the secret key using a low number of measurements. This gives a motivation that the difference in accuracy found in our studies directly impacts the success rate of side-channel analysis.

One interesting future field of research would be to investigate the behavior of various deep learning algorithms for this problem.

IV. Conclusions

This paper demonstrates the application of several state-of-the-art machine learning methods for side-channel analysis. Given a proper tuning, we show that even in an unrestricted scenario, i.e., using a sufficient number of profiling traces, ML techniques are able to be more efficient than template attacks. In case of limited resources (e.g., computing power, storage), we furthermore investigate the performance of all techniques when only a subset of features or points in time are used.

Finally, we present a novel measure called the Data Confusion Factor that provides a measure of successfulness for machine learning techniques. This measure is a preliminary but promising step to link the outcome of ML techniques with the success rate of side-channel analysis within the profiling phase, which opens a completely new research topic for SCA.

Acknowledgment

This work was supported in part by the Research Council KU Leuven (C16/15/058), by the Flemish Government through G.0130.13N and FWO G.0876.14N, and by the Hercules Foundation AKUL/11/19. In addition, this work was supported in part by Croatian Science Foundation under the project IP-2014-09-4882.

References

[1] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[2] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Proceedings of CRYPTO'96*, ser. LNCS, vol. 1109. Springer-Verlag, 1996, pp. 104–113.

[3] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Proceedings of CRYPTO'99*, ser. LNCS, vol. 1666. Springer-Verlag, 1999, pp. 388–397.

[4] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES '01. London, UK: Springer-Verlag, 2001, pp. 251–261. [Online]. Available: http://dl.acm.org/citation.cfm?id=648254.752700

[5] S. Chari, J. R. Rao, and P. Rohatgi, "Template Attacks," in *CHES*, ser. LNCS, vol. 2523. Springer, August 2002, pp. 13–28, San Francisco Bay (Redwood City), USA.

[6] W. Schindler, K. Lemke, and C. Paar, "A Stochastic Model for Differential Side Channel Cryptanalysis," in *CHES*, ser. LNCS, LNCS, Ed., vol. 3659. Springer, Sept 2005, pp. 30–46, Edinburgh, Scotland, UK.

[7] A. Heuser and M. Zohner, "Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines," in *COSADE*, ser. LNCS, W. Schindler and S. A. Huss, Eds., vol. 7275. Springer, 2012, pp. 249–264.

[8] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: a first study," *Journal of Cryptographic Engineering*, vol. 1, pp. 293–302, 2011.

[9] L. Lerman, G. Bontempi, and O. Markowitch, "Power Analysis Attack: An Approach Based on Machine Learning," *Int. J. Appl. Cryptol.*, vol. 3, no. 2, pp. 97–115, Jun. 2014. [Online]. Available: http://dx.doi.org/10.1504/IJACT.2014.062722

[10] L. Lerman, S. F. Medeiros, G. Bontempi, and O. Markowitch, "A Machine Learning Approach Against a Masked AES," in *CARDIS*, ser. Lecture Notes in Computer Science. Springer, November 2013, berlin, Germany.

[11] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F. Standaert, "Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis)," in *Constructive Side-Channel Analysis and Secure Design COSADE 2015, Berlin, Germany, 2015. Revised Selected Papers*, 2015, pp. 20–33.

[12] O. Choudary and M. G. Kuhn, "Efficient Template Attacks," in *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, 2013, pp. 253–270.

[13] TELECOM ParisTech SEN research group, "DPA Contest (4^{th} edition)," 2013–2014, http://www.DPAcontest.org/v4/.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[15] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *Journal of Machine Learning Research*, vol. 15, pp. 3133–3181, 2014. [Online]. Available: http://jmlr.org/papers/v15/delgado14a.html

[16] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[17] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[18] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 10, pp. 1619–1630, Oct. 2006.

[19] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997. [Online]. Available: http://dx.doi.org/10.1006/jcss.1997.1504

[20] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.