Template Attack vs. Bayes Classifier*

Stjepan Picek[1], Annelie Heuser[2], and Sylvain Guilley[3,4]

[1] KU Leuven, ESAT/COSIC and iMinds, Kasteelpark Arenberg 10, bus 2452, B-3001 Leuven-Heverlee, Belgium
[2] CNRS, IRISA, Rennes, France
[3] LTCI, Télécom ParisTech, Université Paris-Saclay, 75 013 Paris, France
[4] Secure-IC S.A.S., Threat Analysis Business Line, 35 510 Cesson-Sévigné, France

Abstract. Side-channel attacks represent one of the most powerful category of attacks on cryptographic devices with profiled attacks in a prominent place as the most powerful among them. Indeed, for instance, template attack is a well-known real-world attack that is also the most powerful attack from the information theoretic perspective. On the other hand, machine learning techniques have proven their quality in a numerous applications where one is definitely side-channel analysis. As one could expect, most of the research concerning supervised machine learning and side-channel analysis concentrated on more powerful machine learning techniques. Although valid from the practical perspective, such attacks often remain lacking from the more theoretical side. In this paper, we investigate several Bayes classifiers, which present simple supervised techniques that have significant similarities with the template attack. More specifically, our analysis aims to investigate what is the influence of the feature (in)dependence in datasets with different amount of noise and to offer further insight into the efficiency of machine learning for side-channel analysis.

Keywords: Template attack, Supervised machine learning, Bayes classifier, Feature dependence

1 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 which aim at breaking cryptographic secrets by exploiting physical information while the device is processing sensitive data. For example, an adversary could monitor the running time [1], the cache behavior, the power consumption [2], and/or the electromagnetic radiation [3] 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

* This work has been supported in part by Croatian Science Foundation under the project IP-2014-09-4882. In addition, this work was supported in part by the Research Council KU Leuven (C16/15/058) and IOF project EDA-DSE (HB/13/020)

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), most of the time based on Gaussian assumption [4], 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. However, their efficiency can only be guaranteed when the template estimates are provided with an reasonable amount of traces in the profiling phase.

Recently, another type of solutions that rely on machine learning (ML) has been investigated [5–7, 7–12]. These related contributions highlight that ML-based side-channel attacks are effective in various experiments. However, ML techniques also come with their own drawbacks. First, today there exists a plenitude of different algorithms one can choose and that choice is not always an easy one. For side-channel evaluation recent works have mainly investigated Support Vector Machines (SVMs) as well as Random Forest. Indeed, here only previous experience can help since there are no general guidelines how to select a subset of algorithms to conduct an analysis.

Because of the "No Free Lunch" theorem, a researcher cannot be sure whether he selected the best algorithm for the task [13]. Moreover, usually more powerful ML methods come with a number of parameters one needs to tune. Furthermore, the higher complexity in the attack phase makes the investigated ML techniques unattractive for some security evaluation scenarios. Finally, even when good results are obtained, the complexity of ML methods makes it difficult to offer some theoretical consideration about the algorithms' performance.

In this paper, we employ several simpler ML techniques that have either no parameters or only a few parameters where all of those techniques share commonalities with the template attack.

Our Contributions There are several contributions we make in this paper. The most important one is the systematic study of the successfulness of the Bayes approach. Here, we investigate the efficiency of the method starting with an assumption that all attributes (features) are independent, following with one level of dependency and finishing with the scenario where all features are dependent.

Moreover, we test the performance of such classifiers in several different scenarios with respect to the number of classes and the level of noise. Finally, we use the best performing algorithms in a setting where each of the classes is equally distributed.

Road Map This paper is organized as follows. Section 2 provides preliminaries about profiled side-channel analysis and the studied algorithms. Our experimental setup is given in Subsection 3.1 and Subsection 3.2. The results of the tuning phase are given in Subsection 3.3 and in Subsection 3.4 for testing. We provide a discussion in Section 4 and conclude afterwards.

2 Background & Algorithms

In this section we cover basic information about profiled side-channel attacks, more specifically template attack (TA) and Bayes machine learning family of techniques.

2.1 Profiled side-channel analysis

Let k^* denote the secret cryptographic key, k any possible key hypothesis from the keyspace $\mathcal{K}$, and T be the input or ciphertext of the cryptographic algorithm. The mapping $f : (\mathcal{T}, \mathcal{K}) \rightarrow \mathbb{F}_2^n$ maps the input or ciphertext $t \in \mathcal{T}$ and a key hypothesis $k \in \mathcal{K}$ to an internally processed variable in some space $\mathbb{F}_2^n$ that is assumed to relate to the measured leakage X, where n is the number of bits. Generally it is assumed that f is known to the attacker. The measured leakage X can then be written as

$$X = \varphi(f(T, k^*)) + N, \tag{1}$$

where N denotes an independent additive noise and where φ is a device-specific deterministic function. In the sequel, we are particularly interested in multivariate leakage $\boldsymbol{X} = X_1, \ldots, X_D$, where D is the data dimensionality (i.e., the number of time samples per measurement trace) or features (attributes) in machine learning terminology.

In order to guess the secret key an attacker chooses a model class $Y \in \mathcal{Y}$ depending on a key guess k and on some known random text T. Considering a powerful attacker, a set of N profiling traces $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is used in order to estimate the leakage model beforehand, which can then be used in the attacking phase with $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_Q$ traces (the general scenario is displayed in Fig. 1).

2.2 Gaussian Naive Bayes

The Naive Bayes classifier is based on the Bayesian rule and works under the simplifying assumption that the measurements are mutually independent among the D features given the target class. More precisely, given the vector of N observed attribute values x the posterior probability for each class value y is computed as:

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

where $\boldsymbol{X} = \boldsymbol{x}$ represents the event that $\boldsymbol{X}_1 = \boldsymbol{x}_1 \wedge \boldsymbol{X}_2 = \boldsymbol{x}2 \wedge \ldots \wedge \boldsymbol{X}_N = \boldsymbol{x}_N$. Because this event is a conjunction of conditionally D independent events, individual probabilities can be multiplied. Moreover, these probabilities are simple to estimate from the training data:

$$p(\boldsymbol{X} = \boldsymbol{X}|Y = y) = \prod_{i=1}^{D} p(X_i = x_i|Y = y). \tag{3}$$

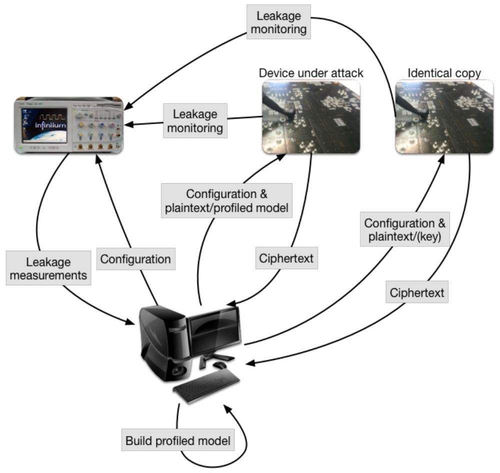


Fig. 1: General scenario of profiled side-channel analysis

As $p(\boldsymbol{X} = \boldsymbol{x})$ in Eq. (2) does not depend on y and this is not key dependent it can be droppped, therefore, the Naive Bayes classifies as:

$$p(Y = y|X = x) = p(Y = y)\prod_{i=1}^{D} p(X_i = x_i|Y = y). \tag{4}$$

Note that the class variable Y and the measurement X are not of the same type: Y is discrete while 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.

Moreover, the Gaussian Naive Bayes classifiers assumes that the predictor attributes are following a normal distribution and thus $p(X_i = x_i|Y = y)$ in Eq. (4) can be calculated as:

$$p(X_i = x_i|Y = y) = \frac{1}{\sqrt{2\pi\sigma_y}} e^{-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}}. \tag{5}$$

Note that the (Gaussian) Naive Bayes method only compiles a one dimensional table of class probability estimates and a two dimensional table of conditional attribute/value probability estimates during the training phase. The space complexity for both training and testing phase is $\mathrm{O}\big(|\mathcal{Y}|Dv\big)$, where $|\mathcal{Y}|$ is the number of classes, D is the number of features, and v is the average number of values for a feature. The time complexity for the training phase equals $\mathrm{O}\big(ND\big)$ and for the testing phase is equal to $\mathrm{O}\big(|\mathcal{Y}|D\big)$. Further information about Naive Bayes algorithm can be found in [14].

2.3 Template Attack

Similar to the Gaussian Naive Bayes classifier the template attack relies on the Bayes theorem and mostly in the state-of-the art relies on a normal distribution. However, it is not "Naive" and considers the features as dependent (i.e. Eq. (3) does not apply). Accordingly, template attack assumed that each $P(\boldsymbol{X} = \boldsymbol{x}|Y = y)$ follows a (multivariate) Gaussian distribution and is thus parameterized by its mean and covariance matrix:

$$p(X = x|Y = y) = \frac{1}{\sqrt{(2\pi)^D|\Sigma_y|}} e^{-\frac{1}{2}(\boldsymbol{x}-\mu_y)^T \Sigma_y^{-1}(\boldsymbol{x}-\mu_y)}. \tag{6}$$

The authors of [15] propose to use only one pooled covariance matrix Σ 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. The time complexity for TA is $\mathrm{O}\big(ND^2\big)$ in the training phase and $\mathrm{O}\big(|\mathcal{Y}|D^2\big)$ in the testing phase and space complexity $\mathrm{O}\big(|\mathcal{Y}|D^2v\big)$.

2.4 Averaged n-Dependence Estimators - AnDE

As discussed in the previous sections, TA assumes dependence among all features which may result in statistical difficulties (and higher computation costs). Contrary, Naive Bayes considers the features independently. However, if this assumption of independence is violated, Naive Bayes may result in high precision loss.

To deal with these two extreme, there are many possible approaches where the best results in the field of machine learning were obtained with algorithms that relax the independence assumption, as with LBR [16] and SP-TAN [17] classifiers (to name a few prominent ones). However, their improved performance comes with a price of considerably higher computation cost. The effort that aimed at relaxing the independence assumption, while keeping the computation cost low, led to the Averaged One-Dependence Estimators (*AODE*) strategy [18].

In *AODE*, there is a Super-Parent One-Dependence Estimate that relaxes the assumption of independence by making all other attributes independent given the class and one privileged attribute called the super-parent x_α. This represents a weaker conditional independence assumption than the one present in Naive Bayes since it must be true if Naive Bayes is true, but it also may be true when Naive Bayes is not true [18]:

$$\begin{aligned} &p(Y = y|X = x) = \\ &p(Y = y, x_\alpha) \prod_{i=1}^{D} p(X_i = x_i|Y_i = y_i, x_\alpha). \end{aligned} \tag{7}$$

Since this is a weaker assumption, the bias of this model should be lower, while the variance should be higher since it is derived from higher-order probability estimates. It is possible to further generalize to higher-order probabilities and use Averaged n-Dependence Estimators (*AnDE*) [19]:

$$p(Y = y|X = x) = \tag{8}$$
$$\sum_{s \in S^n} p(Y = y, x_s) \prod_{i=1}^{D} p(X_i = x_i | Y_i = y_i, x_s) / \binom{D}{n},$$

where S^n indicates all subsets of size n of the set $\{1, \ldots, D\}$.

$AnDE$ algorithm works by learning an ensemble of n-dependence classifiers where the prediction is obtained by aggregating the predictions of all classifiers. Note that the n-dependence estimator means that the probability of an attribute is conditioned by the class variable and at most n other attributes. In $AnDE$ algorithm, an n-dependence classifier is constructed for every combination of n attributes where those n attributes are set as parents to all other attributes. We note that Naive Bayes is the same as the $A0DE$ while $AODE$ is the same as $A1DE$.

In this paper we use $A1DE$ algorithm from the $AnDE$ family of algorithms. Note that $AnDE$ when $n > 2$ is rarely used since both space and time complexity is very high [19]. Space complexity for both training and testing phases for $AnDE$ equals $\mathrm{O}\big(|\mathcal{Y}|\binom{D}{n+1}v^{n+1}\big)$ where n is the number of parent nodes (except class). Time complexity for the training phase is $\mathrm{O}\big(N\binom{D}{n+1}\big)$ and for testing phase for classifying a single example is $\mathrm{O}\big(|\mathcal{Y}|D\binom{D}{n}\big)$ [19].

3 Experimental Evaluation

Before presenting the results of the experiments, we briefly discuss how to conduct proper (ML) analysis that is reproducible and statistically relevant. One needs to present the number of instances, the number of features, and the number of classes (if known). Additionally, if the data comes from different distributions, one needs to discuss those.

Furthermore, if not all data from datasets are used, it is necessary to write how the samples are chosen and how many are used in the experiments. Finally, one needs to define the level of noise appearing in the data in a clearly reproducible way, e.g., with the signal-to-noise ratio (SNR).

3.1 Data Setup

To ensure the reproducibility of presented results, we use two publicly available data sets for our study where they differ in the amount of noise. In our study we work with datasets with 5 000, 10 000, 20 000, 30 000, 50 000, and 100 000 measurements which are randomly selected from the whole data sets. The number of features equals 50 and the model consists either of 256 uniformly distributed classes (S-box output, see Eq. (9) and Eq. (11)) or 9 binomial distributed classes (the Hamming weight of the S-box output).

DPAcontest v2 [20]. 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.,

$$Y(k^*) = \underbrace{\texttt{Sbox}[T_{b_1} \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{C_{b_2}}_{\text{ciphertext byte}} , \tag{9}$$

where k^* denotes the secret key, T_{b_1} and T_{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. (9) $Y(k^*)$ consists in 256 values, as an additional model we applied the HW on this value resulting in 9 classes. For our study, we selected 50 points of interests with the highest correlation between $Y(k^*)$ and data set. Figure 2 shows the absolute correlation between $Y(k^*)$ and the measurements for our selected points. One can see that the measurements are relatively noisy and the resulting SNR (signal-to-noise ratio) is between 0.0069 and 0.0096. We calculate the SNR as:

$$\frac{var(signal)}{var(noise)} = \frac{var(Y(k^*))}{var(X - Y(k^*))}. \tag{10}$$

Figure 3 show the correlation between the first point of interest and all points. Interestingly, one can observe a dependency for roughly the first 50 points that decreases for the second half.

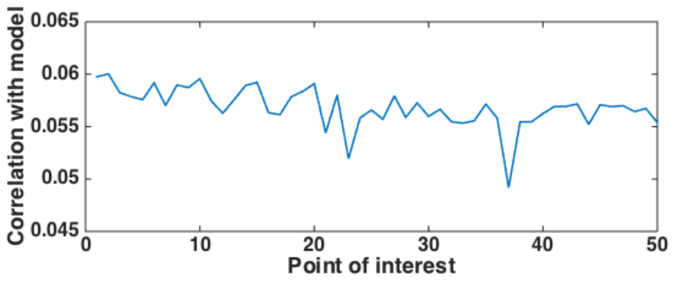
DPAcontest v4 [21]. The 4th version provides measurements of a masked AES software implementation. However, as the mask is known, one can easily turn it into an unprotected scenario. Though, as it is a software implementation, the most leaking operation is not the register writing, but the processing of the S-box operation and we attack the first round. Accordingly, the leakage model changes to

$$Y(k^*) = \texttt{Sbox}[P_{b_1} \oplus k^*] \oplus \underbrace{M}_{\text{known mask}} , \tag{11}$$

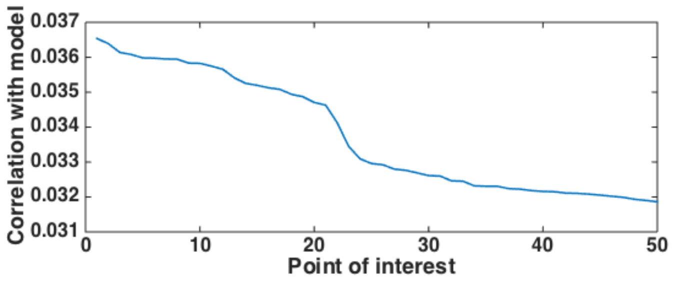
where P_{b_1} is a plaintext byte and we choose $b_1 = 1$. Figure 4 shows the absolute correlation between $Y(k^*)$ and the measurements for our selected points. Compared to the measurements from version 2, there is much higher correlation and naturally also SNR, which is between 0.1188 and 5.8577.

The correlation between the first point of interest and all points can be seen in Figure 5. In this data set again the first points are correlated, where the correlation is slightly decreasing over the features.

[5] see e.g., in the hall of fame on [20]



(a) Using 9 classes (HW)



(b) Using 256 classes

Fig. 2: Correlation between our model and the measurements (v2)

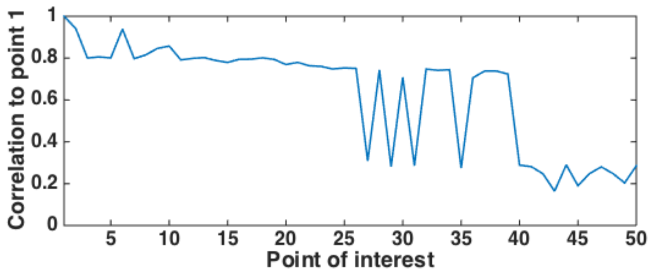
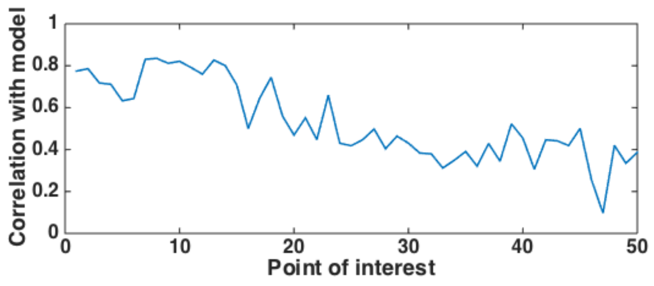


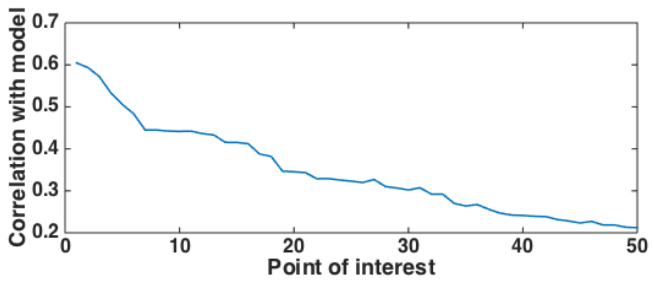
Fig. 3: Correlation between the first point of interest and all points (v2)

3.2 Practical Algorithm Setting

When discussing the experiments, first it is necessary to discuss how the data is divided into training and testing sets. Then, for the training phase one needs to define the test options (e.g. whether to use percentage split, the whole dataset, cross-validation, etc.) After that, for each algorithm one needs to define a set of parameter values to conduct the tuning phase. There are different options how to conduct tuning, but we consider as a reasonable approach to start with the



(a) Using 9 classes (HW)



(b) Using 256 classes

Fig. 4: Correlation between our model and the measurements (v4)

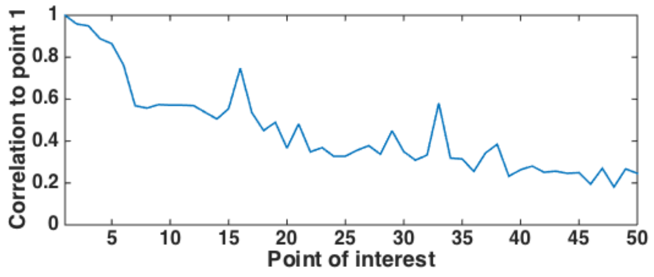


Fig. 5: Correlation between the first point of interest and all points (v4)

default parameters and continue changing them until there is no more improvement.

We divide data into training and testing set in a ratio of 2:1. Then, we take the bigger set as the training set (the set with the 2/3 of the data) and the smaller set for testing (1/3 of the data). On the training set, we conduct 10-fold cross-validation with all parameters considered. In the 10-fold cross-validation, the original training sample is first randomly partitioned into 10 equal sized subsets. Then, a single subsample is selected to validate the data, while the remaining 9 subsets are used for training. The cross-validation process is repeated 10 times, where each of 10 subsamples is used once for validation. The obtained results are then averaged to produce an estimate.

For the tuning phase, we consider sufficient to report the accuracy (ACC) value. Here, accuracy represents the ratio between the sum of true positive and true negative measurements divided by the total number of measurements. However, for the testing results, one should report the accuracy, the area under the ROC curve (AUC), and the F-measure. Here, the area under the ROC curve is used to measure the accuracy and ROC curve is the ratio between true positive rate and false positive rate. AUC close to 1 represents a good test, while value close to 0.5 represents a random guessing. F-measure is the harmonic mean of the precision and recall, where precision is the ratio between true positive (TP - the number of examples predicted positive that are actually positive) and predicted positive, while recall is the ratio between true positives and actual positives [22]. Both the F-Measure and the AUC can help in situations where accuracy can be misleading, i.e., where we are also interested in the number of false positive and false negative values.

With regards to the choice of algorithms, it is necessary to specify which framework and algorithms are investigated. Moreover, all settings that uniquely define the algorithm need to be enumerated. Here, we use Weka as the framework for conducting the ML analysis [23] and we use Naive Bayes and A1DE algorithms. Template attack and pooled template attack are implemented in MATLAB. Since the Naive Bayes and TA does not have parameters to tune, we only need to conduct tuning for the A1DE algorithm. We conduct a parameter sweep where we investigate the frequency limit and the weight parameters. The frequency limit $freq$ parameter denotes that all features with a frequency in the train set below this value are not used as parents. The weight parameter m sets the base probabilities with m-estimation [24].

Table 1: Parameter tuning for $A1DE$ and 9 classes

freq. \ m	0.1	0.2	0.5	0.8	1	2	3	4	5
					DPAcontest v4				
1	83.22	83.33	83.35	83.34	83.36	**83.39**	83.3	83.3	83.29
					DPAcontest v2				
1	27.86	27.86	27.86	27.86	**27.86**	27.86	27.86	27.86	27.86

Table 2: Parameter tuning for $A1DE$ and 256 classes

freq. \ m	0.1	0.2	0.5	0.8	1	2	3	4	5
					DPAcontest v4				
1	22.68	22.67	22.76	**22.77**	22.67	22.22	22.02	21.85	21.78
					DPAcontest v2				
1	0.54	0.54	0.54	0.54	**0.54**	0.54	0.54	0.54	0.54

3.3 Parameter Tuning Phase

Note that due to a large number of datasets, we conduct parameter tuning only for the middle-sized dataset (20 000 measurements) and we use the best parameter combinations on all datasets. In all tables we highlight the best results with text in bold style. All the results in this section are presented as the accuracy of the classifier. In Tables 1 and 2, we give the best parameter combinations for 9 and 256 classes scenario with A1DE algorithm, respectively. In all the cases where multiple parameter combination return the same accuracy value, we use the default one ($m = 1$) as the best one. We note that we also experimented with the frequency limit parameter $freq$ where we set it in range $[1, 5]$, but it did not have any influence on the results so we omitted it from the tables. On the basis of the tuning phase, one can observe that the parameters have very small influence on the result: in all the experiments we use frequency $freq$ equal to 1 while the weight parameter m changes slightly.

Table 3: Testing results for 9 classes (ACC/F-Measure/AUC)

Size	Naive Bayes	$A1DE$	TA	TA (pooled)
		DPAcontest v4		
5 000	65.52/65.5/91.3	**78.12/78.1/96.3**	19.49	62.07
10 000	67.01/67.1/91.5	**81.26/81.3/97.2**	52.14	76.54
20 000	68.25/66.7/91.3	**83.39/83.4/97.7**	75.43	77.78
30 000	67.66/67.7/91.7	**84.25/84.3/97.9**	77.45	78.09
50 000	67.19/67.2/91.5	**84.93/84.9/98**	78.71	77.85
100 000	67.29/67.3/91.7	**85.55/85.6/98.1**	79.91	77.83
		DPAcontest v2		
5 000	10.06/10.5/50.1	25.76/10.6/50	1.29	**10.07**
10 000	**10.94/9.9/50.1**	26.06/10.8/50	1.73	8.74
20 000	7.88/9.2/50.5	27.1/11.6/50	**15.48**	7.64
30 000	8.81/10.4/50.3	**25.6/15.5/51.7**	17.66	6.66
50 000	10.21/11.6/50.4	**24.3/15.8/51.2**	15.99	5.88
100 000	12.44/14.1/50.6	**23.79/16.3/50.5**	13.20	5.98

Table 4: Testing results for 256 classes (ACC/F-Measure/AUC)

Size	Naive Bayes	*A1DE*	TA	TA (pooled)
		DPAcontest v4		
5 000	**15.29/14.7/91.6**	10.29/8/93.7	0.23	14.89
10 000	18.26/17.1/93.4	15.65/13.7/95.5	0.32	**19.68**
20 000	20.21/18.3/94.5	22.56/21.2/96.9	0.52	**23.65**
30 000	20.88/19/94.7	**28.19/27.4/97.7**	9.44	25.53
50 000	21.22/19.1/95	**32.06/31.5/98.2**	15.63	27.47
100 000	22.25/20.1/95.3	**37.63/37.1/98.7**	21.66	29.14
		DPAcontest v2		
5 000	**0.59/0.1/51**	0.06/0/50	0.53	0.11
10 000	**0.56/0.2/51.3**	0.38/0/50	0.52	0.32
20 000	**0.6/0.1/51.2**	0.34/0/50	0.55	0.32
30 000	**0.63/0.1/50.8**	0.29/0/50	0.30	0.40
50 000	**0.51/0.1/51.1**	0.41/0/50	0.36	0.50
100 000	**0.54/0.1/50.9**	0.39/0/50	0.46	0.45

3.4 Verification Phase

In this section we perform the testing on an independent set of traces to verify the performances for classifying into 9 and 256 classes. We present results in Table 3 for 9 classes and in Table 4 for 256 classes in a form ACC/F-Measure/AUC for the Naive Bayes and A1DE classifiers while for the template attack (Eq. (**??**)) and template attack with pooled covariance matrix (Eq. (**??**)) we give only the accuracy value. Accuracy is used as the primary criterion of the algorithms' successfulness to predict the correct class y using $\boldsymbol{X}_1, \dots, \boldsymbol{X}_Q$ measurements, whereas the F-Measure and AUC gives more details about the classification and can additionally be used in case the accuracy of several algorithms is similar or even the same. We note that the size parameter represents the whole dataset size and in the testing phase only one third of that size is actually used (while the other two thirds are used in the training phase).

4 Discussion

From our results we can confirm that the pooled TA has a higher accuracy than TA when the profiling set is rather small (see e.g. when using 5 000 and 10 000 measurements for 9 classes and nearly for all profiling sets using 256 values). However, when the profiling set seems sufficient enough to achieve reasonable estimate of each Σ_y TA is more accurate than its pooled version.

Naturally, as the data shows dependence between features (cf. Figures 3 and 5), the accuracy of the approach to assume independence among all features as done by Naive Bayes is lower than that of the pooled version of TA and TA when using a higher amount of measurements. But we can observe it may be an alternative even for the pooled version when the profiling set is rather small (see e.g. when using 5 000 measurement DPAcontest v4 and for 5 000, 10 000, and 20 000 for DPAcontest v2).

On the other hand, the $A1DE$ algorithm achieves in most of the scenarios a higher accuracy than TA and the pooled version of TA for both 9 and 256 classes using the data of the DPAcontest v4. Therefore, $A1DE$ can be considered as a valid alternative for TA (standard and pooled). For example, looking at the results using 9 classes for the DPAcontest v4 one can observe that $A1DE$ is more efficient regardless on the number of profiling traces used (even when a rather large amount of 100 000 measurements in a rather high SNR scenario is used). However, when considering 256 classes and DPAcontest v4, we actually observer that there are several scenarios where the pooled version of TA achieves better results than $A1DE$. We want to stress that this ML technique does not come with a high computation burden when compared to SVM, Random Forest, and decision trees which have been often found superior to TA in this context in previous works. Even more, our experiments showed that the computation of TA using 256 classes was much more expensive in terms of time and resources than $A1DE$. To conclude, when dealing with measurements that do not have too much noise (as in DPAcontest v4) and where the number of classes is small (e.g. the Hamming weight model) the results suggest that $A1DE$ should represent a viable option for profiled SCA.

Note that, $A1DE$ results for the DPAcontest v2 that are marked with gray color deserve some extra explanations. Although the accuracy rate gives the impression that the results are good, we see that the AUC equals 50%, which means that the classifiers are making random choices. Indeed, by closer look on the confusion matrix (matrix where each column represents the instances in a predicted class while each row represents the instances in an actual class) we see that the algorithms classified all instances into a single class and the accuracy is simply the number of instances actually belonging to that class. This is a problem commonly known as overfitting. Naturally, classifying into only one class will not lead to a successful side-channel distinguisher as this reveals no information about the secret key.

Additionally, we want to discuss about the distribution of class values y in the profiling set. Naturally, when taking the Hamming weight of an 8-bit intermediate state we gain 9 binomial distributed classes. Therefore, the number of instances for observing class $y = 0$ or $y = 8$ is rather limited and the estimation for Σ_y is not provided with an equal number of instances for each y. In particular, this results in different estimation errors in Σ_y for all classes y. This is one explanation why the pooled variant is more efficient than the standard one when the number of measurements is not very large (i.e., in our experiments for DPAcontest v4 with 9 classes $\leq 30\,000$).

We therefore additionally test the accuracy when each class estimate of $\hat{P}(\boldsymbol{X}|Y = y)$ is provided with the same number of instances of $\boldsymbol{X}$. As the maximum number of instances for class $y = 8$ is around 350, we used 350 for each class resulting in 3 150 measurements in total. The results are given in Table 5.

Table 5: Testing results for 9 classes each with an equal number of measurements (350 for each class)

Dataset	**DPAcontest v4**	**DPAcontest v2**
Naive Bayes	73.76	**14.75**
*A*1*DE*	**80.67**	11.76
TA	63.61	12.53
TA (pooled)	77.82	13.00

Interestingly, we can observe that using 3 150 measurements with a equal number in each class results in a better accuracy than using 5 000 and 10 000 measurements with a binomial distribution. When comparing TA with the pooled version, the later one is still more accurate thus for our experiments 350 is not accurate enough for a well estimation of Σ_y. Again, $A1DE$ is superior to both TA versions and considering full independence with Naive Bayes is better than TA. Even more, when considering $A1DE$ for DPAcontest v2 with equal number of classes we do not run into the problem of one class prediction as in the previous section.

5 Conclusions

In this paper, we investigate the performance of template attack as a scenario where all features are dependent versus machine learning algorithm from the Bayes family. We start with the Naive Bayes where the assumption is that all features are independent and then we relax that assumption and go to the $A1DE$ algorithm that has a single dependency. Our results show that these alternatives are especially interesting when the amount of profiling traces is restricted and thus a proper estimation of the covariance matrix cannot be done.

Our results show that these ML algorithms may even have an advantage compared to the pooled version of the template attack in which only one covariance matrix has to be estimated. Concluding we propose new tools which, as the template attack, are based on the Bayes rule that might be relevant security evaluation methods when the profiling base is not extremely large and other ML techniques like Support Vector Machines and Random Forest might be too costly to tune and perform. Therefore, our results suggest one can often relax the constraint on the dependencies of features and consider features either independent or with minimal dependence and still achieve very good results.

References

1. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Proceedings of CRYPTO'96. Volume 1109 of LNCS., Springer-Verlag (1996) 104–113

2. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Proceedings of CRYPTO'99. Volume 1666 of LNCS., Springer-Verlag (1999) 388–397
3. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems. CHES '01, London, UK, UK, Springer-Verlag (2001) 251–261
4. 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.
5. 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
6. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. Journal of Cryptographic Engineering **1** (2011) 293–302
7. Lerman, L., Bontempi, G., Markowitch, O.: Power analysis attack: An approach based on machine learning. Int. J. Appl. Cryptol. **3**(2) (June 2014) 97–115
8. 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 Mangard, S., Poschmann, A.Y., eds.: Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers. Volume 9064 of Lecture Notes in Computer Science., Springer (2015) 20–33
9. Lerman, L., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES - Reaching the limit of side-channel attacks with a learning model. J. Cryptographic Engineering **5**(2) (2015) 123–139
10. Lerman, L., Medeiros, S.F., Bontempi, G., Markowitch, O.: A Machine Learning Approach Against a Masked AES. In: CARDIS. Lecture Notes in Computer Science, Springer (November 2013) Berlin, Germany.
11. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Side-channel Analysis of Lightweight Ciphers: Does Lightweight Equal Easy? In: RFIDSec. (2016)
12. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking Cryptographic Implementations Using Deep Learning Techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering, Springer International Publishing (2016) 3–26
13. Wolpert, D.H.: The Lack of a Priori Distinctions Between Learning Algorithms. Neural Comput. **8**(7) (October 1996) 1341–1390
14. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning **29**(2) (1997) 131–163
15. 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, November 27-29, 2013. Revised Selected Papers. Volume 8419 of LNCS., Springer (2013) 253–270
16. Zheng, Z., Webb, G.I.: Lazy Learning of Bayesian Rules. Machine Learning **41**(1) (2000) 53–84
17. Keogh, E.J., Pazzani, M.J.: Learning Augmented Bayesian Classifiers: A Comparison of Distribution-based and Classification-based Approaches (1999)
18. Webb, I.G., Boughton, R.J., Wang, Z.: Not So Naive Bayes: Aggregating One-Dependence Estimators. Machine Learning **58**(1) (2005) 5–24
19. Webb, G.I., Boughton, J.R., Zheng, F., Ting, K.M., Salem, H.: Learning by extrapolation from marginal to full-multivariate probability distributions: decreasingly naive Bayesian classification. Machine Learning **86**(2) (2012) 233–272

20. TELECOM ParisTech SEN research group: DPA Contest (2^{nd} edition) (2009–2010) `http://www.DPAcontest.org/v2/`.
21. TELECOM ParisTech SEN research group: DPA Contest (4^{th} edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
22. Powers, D.M.W.: Evaluation: from precision, recall and F-factor to ROC, informedness, markedness and correlation (2007)
23. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explor. Newsl. **11**(1) (November 2009) 10–18
24. Cestnik, B.: Estimating probabilities: A crucial task in machine learning. In: Proc. of the European Conference on Artificial Intelligenc. (1990)