Climbing Down the Hierarchy: Hierarchical Classification for Machine Learning Side-channel Attacks

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

[1] KU Leuven ESAT/COSIC and imec, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
[2] IRISA/CNRS, Rennes, France
[3] University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
[4] IRISA/Inria, Rennes, France

Abstract. Machine learning techniques represent a powerful paradigm in side-channel analysis, but they come with a price. Selecting the appropriate algorithm as well as the parameters can sometimes be a difficult task. Nevertheless, the results obtained usually justify such an effort. However, a large part of those results use simplification of the data relation and in fact do not consider all the available information. In this paper, we analyze the hierarchical relation between the data and propose a novel hierarchical classification approach for side-channel analysis. With this technique, we are able to introduce two new attacks for machine learning side-channel analysis: Hierarchical attack and Structured attack. Our results show that both attacks can outperform machine learning techniques using the traditional approach as well as the template attack regarding accuracy. To support our claims, we give extensive experimental results and discuss the necessary conditions to conduct such attacks.

Keywords: Side-channel attacks, profiled scenario, machine learning techniques, hierarchical classification, hierarchical attack, structured attack

1 Introduction

Side-channel attacks (SCAs) are capable of revealing secret keys on cryptographic devices from unintentionally emitted information during computation, like power consumption or electromagnetic emmanation. To evaluate the worst case security threat, so-called profiled attacks have to be conducted, which consist of an additional profiling phase. In this phase, one has the full control over the device to build additional advanced models, which are then exploited to extract key-dependent information in the attacking phase.

Profiled attacks can be divided into two main approaches. First, there exist the traditional methods like the template attack [1] and stochastic approach [2], relying on the maximum-likelihood estimation. Recently, machine learning (ML) techniques, adapted as side-channel attacks, have been proposed, which can be beneficial in many scenarios. In particular, ML techniques have originally been introduced to classify between two classes (i.e., a bit is set (= 1) or not (= 0)) [3, 4]. The first extension to 9 Hamming weight (HW) classes for Support Vector Machines (SVMs) has been given in [5]. Following this approach, recent works studied ML techniques mostly using 9 (e.g., [6, 7]) or up to 16 classes [8].

However, in most real-world applications, the measured leakage does not follow the Hamming weight (HW) or the Hamming distance (HD) model. For instance, the authors in [9] showed that the leakage of the DPAcontest v2 traces [10] using a Xilinx FPGA VirtexTM-5 [11] is highly "non-linear", i.e., the HD does not apply. Instead, one should consider the output of the S-Box itself as a sensitive variable, resulting in 256 classes (considering AES with 8-bit words). Similarly, even when using an Atmel ATMega-163 smart-card as in the DPAcontest v4 [12], the leakage model does not exactly follow the Hamming weight. Thus, especially in a profiled scenario, to be able to capture more knowledge on the leakage model, it is more advisable to use the S-Box output with 256 classes as a sensitive variable directly. Even more, considering 256 classes yields direct information on the key as each class is related to only one key guess. This is naturally not the case when considering 9 Hamming weight classes. For instance, in the worst case when classifying into the Hamming weight class 4 we have 70 corresponding key guesses. Therefore, in a low noise scenario an attacker is clearly able to reveal the secret key within only one trace using 256 classes, whereas this is not possible (on average[5]) when using HW classes.

1.1 Idea & Contributions

The more classes we have, the more instances (measurements) are necessary to obtain high accuracies, i.e., high probability to classify each element to its class correctly. Generally speaking one has either $n+1$ HW classes or 2^n classes relating directly to the key guess where n is the number of bits of intermediate states and a key chunk[6]. Up to now in side-channel analysis, the class variables have been seen in a flat hierarchy, where each label directly results in the classified variables (see Fig. 1). Naturally, estimating 2^n instead of $n+1$ classes may bring statistical difficulties. When considering random classification, with 9 classes there is 1/9 chance of a successful guess, while in the 256 classes scenario there is 1/256 chance for a random hit. However, only considering the Hamming weight classes instead of the value itself brings two drawbacks as discussed before: first, it lowers the information about the secret key and second, it yields an imprecise estimation of the leakage model. On the other hand, with the increase in the number of classes, the computational complexity also rises. In general, for most

[5] Note that, an attacker could reveal the secret key with only one trace if it corresponds to HW 0 or 8, which occurs with a probability of $\frac{2}{256}$.

[6] For simplicity we assume that one key chunk is of the same size as one intermediate state chunk, however, this study can easily be extended for other scenarios as given e.g. in DES.

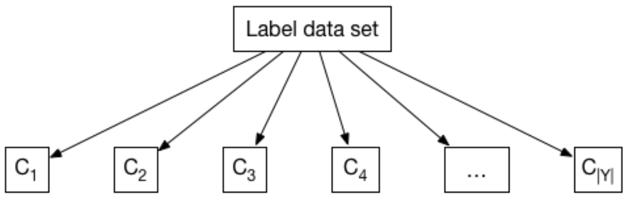


Fig. 1: Flat (standard) approach

ML techniques (with the exception of decision tree based techniques), the complexity for multi-class classification rises with $\mathrm{O}\big(|\mathcal{Y}|\big)$, where $|\mathcal{Y}|$ is the number of classes. This is particularly critical when performing a proper tuning within the profiling phase, which may become very complex in terms of computation resources.

In this paper, in order to circumvent the problems arising when classifying $n+1$ or 2^n directly, we propose to adapt a divide-and-conquer strategy, which enables us to use as many classes as required from a side-channel leakage point of view with a much lower complexity of resources and with a higher accuracy compared to the standard approach. More precisely, our idea is to view the class variables in a tree structure with additional intermediate nodes which are given due to the natural clustering of the measured leakage.

A general illustration is given in Fig. 2, where we first divide into M nodes and then directly into the leafs. Note that compared to the flat approach in Fig. 1, the number of leaves does not change, but its depth does. Certainly, in some scenarios due to the given structure of the leakage, it may be suitable to include several layers of nodes resulting in a higher depth.

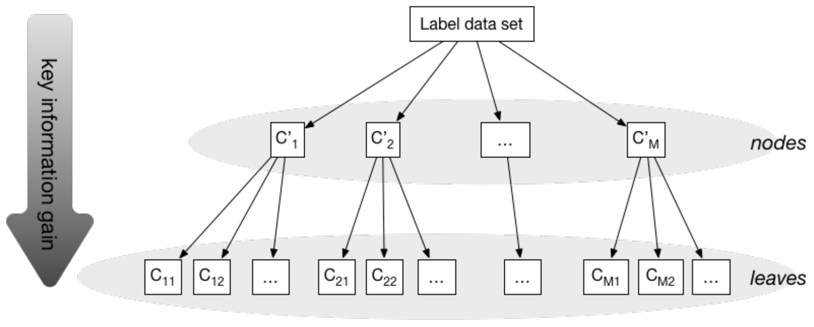


Fig. 2: Hierarchical approach

In the context of side-channel analysis, there is a simple hierarchy that one could follow, which we detail next. However, we stress that the hierarchical approach is not restricted to this specific scenario, but is rather a general concept. In particular, one could adapt the approach of clustering side-channel measurements by their similarity as introduced in [13] to build an appropriate tree structure. Note that here we use a priori knowledge to build more accurate classifiers, i.e., use the fact that we know (or can assume) the semantic hierarchy of the data.

First, we divide our measurements regarding the HW of the sensitive variable resulting in 9 classes and then each of those classes is further divided into the sensitive variable itself (leaf classes). Note that, since the HW of a 8-bit word forms a binomial distribution, two classes (HW 0 and HW 8) cannot be divided any further (i.e., these are already leaf classes), while the rest of the classes (HW 1 to HW 7) have various number of subclasses. In this case, using hierarchical approach, the largest number of classes we need to handle at once is for the class with the Hamming weight equal to 4, where there are 70 possible values. We give a depiction of hierarchical model with the Hamming weight/sensitive variable model in Fig. 3.

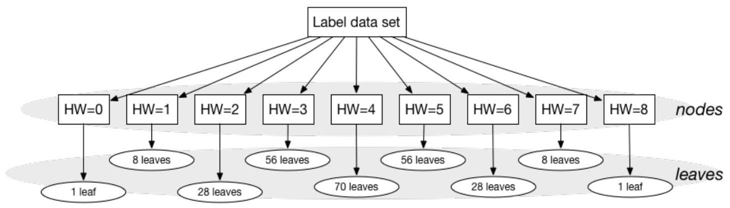


Fig. 3: Hierarchical approach considering HW classes as intermediate nodes

1.2 Road Map

This paper is organized as follows. In Section 2, we present basic information about machine learning and the algorithms we use. Section 3 presents our underlying setup and experimental results for tuning and testing phase for two powerful ML techniques. Next, in Section 4, we give results for a realistic testing scenario as well as a comparison with the template attack. In Section 5, we give a short discussion on the performance of ML as well as some possible future research directions. Finally, in Section 6, we conclude the paper.

2 Machine Learning Techniques

In this section, we briefly present machine learning techniques used in the paper. In our experimental setup, we use four algorithms, where two are relatively simple techniques, i.e., the Naive Bayes and the C4.5 decision tree, while the other two, Support Vector Machines and Rotation Forest, are more complex.

Remark 1. Note that we do not claim that these techniques are optimal, however, preliminary tests showed that both complex techniques had the highest accuracy out of a large pool of tested ML techniques. The reason why we additionally included two simple techniques will be discussed in Section 4.

2.1 Naive Bayes

The Naive Bayes classifier is a method based on the Bayesian rule which works under a simplifying assumption: it assumes that the predictor features (points in the measurement traces) are mutually independent given the target class. Existence of highly-correlated features in a dataset can thus influence the learning process and reduce the number of successful predictions. Additionally, Naive Bayes assumes normal distribution for predictor features. A Naive Bayes classifier outputs posterior probabilities as a result of the classification procedure [14]. Note that the space complexity for Naive Bayes algorithm 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 features. On the other hand, time complexity for the training phase equals $\mathrm{O}\big(ND\big)$ and for the testing phase is equal to $\mathrm{O}\big(|\mathcal{Y}|D\big)$. Here, N is the number of training examples.

2.2 Decision Tree - C4.5

C4.5 is the landmark decision tree algorithm developed by R. Quinlan [15]. It is a divide-and-conquer algorithm that splits features at tree nodes using the information-based gain ratio criterion. The node splits on further branches if more information is gained (as measured by gain ratio) by the split than by keeping all the instances at the node. The runtime of the algorithm is $\mathrm{O}\big(D\big)\times N\times \log N$ where D is the number of features and N is the number of instances [16]. The trees are first grown to full length and pruned afterwards in order to avoid data overfitting.

2.3 Rotation Forest

Rotation Forest (RF) is a more recent decision tree ensemble method proposed by Rodriguez et al [17]. The ensemble is capable of both classification and regression, depending on the base classifier where in most applications, C4.5 algorithm is used as the base learner [15]. The algorithm focuses on presenting transformed data to the classifier by using a projection filter. The most common projection filter and the one that has been shown to be the main factor for the success of the ensemble is the principal component analysis (PCA) [18]. The running time is the same as for C4.5 multiplied with the number of iterations.

2.4 Support Vector Machines

Support Vector Machine (SVM) is a kernel based machine learning family of methods that are used to accurately classify both linearly separable and linearly inseparable data [19]. The basic idea when the data is 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. The computational complexity of SVM with radial kernel is between linear and quadratic in the number of instances. In this work, we investigate only the radial-based SVM. The most significant parameters are the cost of the margin C and the radial kernel parameter γ. As a learning method for SVM, sequential minimal optimization (SMO) type algorithm is used [20]. Because SMO is a binary classification algorithm, for multi-class classification purposes it is adapted to perform $N \times (N-1)/2$ binary classifications, where N denotes the number of classes.

3 The Hierarchical Approach under Test

3.1 Experimental Data

In order to ensure the reproducibility of our results, we use two publicly available data sets for our study.

DPAcontest v2 [10]. This version of the contest provides 1 000 000 measurements (in the template base) 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.,

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

where k^* denotes the secret key, $\texttt{Sbox}^{-1}[\cdot]$ denotes the inverse Sbox operation, 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[7]. For our study, we selected 50 points of interest with the highest correlation between $Y(k^*)$ and data set. Furthermore, we select 100 000 measurements randomly to conduct the subsequent experiments. Figure 4a 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) lies between 0.0069 and 0.0096. To calculate the SNR, we use the model-based approach where we assume a leakage model and X is the measurement we calculate:

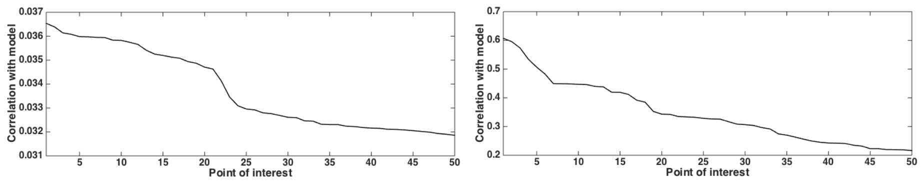
$$\frac{var(signal)}{var(noise)} = \frac{var(Y(k^*))}{var(X - Y(k^*))}. \tag{2}$$

[7] see e.g., in the hall of fame on [10]

DPAcontest v4 [12]. The 4th version provides 100 000 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

$$Y(k^*) = \texttt{Sbox}[P_{b_1} \oplus k^*] \oplus \underbrace{M}_{\text{known mask}} , \tag{3}$$

where P_{b_1} is a plaintext byte and we choose $b_1 = 1$. Figure 4b 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.



(a) DPAcontest v2 (b) DPAcontest v4

Fig. 4: Correlation between our model and the measurements

3.2 Training Phase and Parameter Tuning

Tuning represents an important phase in order to properly use ML techniques, but unfortunately has often been ignored or underestimated in previous works on machine learning side-channel attacks. For parameter tuning, we first randomly selected the instances in a ratio of 2:1, where the total number of instances equals 20 000 instances. Those instances are randomly selected out of the datasets of 100 000 and 1 000 000 for DPAcontest v4 and DPAcontest v2, respectively. 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. All the results in this section are presented as the percentage precision (accuracy) of the classifier. Here, accuracy is defined as the ratio between the sum of true positive and true negative records and sum of all records.

In our experiments we use Weka as the framework for conducting the ML analysis [21]. We do not give details about the tuning phase but we note that we made a grid search where for the Rotation Forest algorithm we investigated *Iteration* parameter in the range $[10, 60]$ with a step of 10. For SVM, we experimented with γ and C parameters, where we tested γ values in the range $[10, 70]$ with a step of 10 and C values in the range $[0.1, 0.5]$ with a step of 0.1. Based on our experiments, we select as the best performing combinations Rotation Forest with 60 iterations and SVM with $C = 70, \gamma = 0.5$ for DPAcontest v4 scenario. For DPAcontest v2, we select Rotation Forest with 60 iterations and SVM with $C = 50, \gamma = 0.4$.

Considering our hierarchical approach and dividing first the measurements into 9 HW classes, we conduct a tuning phase with Rotation Forest and SVM for HW classes 1 to 7 (recall that it is not possible to divide HW classes 0 and 8 further into subclasses). We investigate the same parameter ranges as before, but again omit tuning details, since they are straightforward to obtain (the best obtained algorithms and parameters are given in the next section).

3.3 Testing Results

In this section we perform the testing on an independent set of traces to verify the performances for classifying into 9 and 256 classes (see Table 1) and the performance within each meta-class for the hierarchical approach (see Table 2). The results are given in Accuracy/F-measure/AUC form. 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]. All the values are given in percentages and in parenthesis, we give the parameter combinations reaching those values.

Note that this represents an ideal test scenario, where each meta-class only contains measurements from the correct HW class. Therefore, all the instances in a meta-class really belong to that meta-class. The next section discusses a more realistic attacking scenario where errors are propagated through the tree. For the testing results for the hierarchical approach (i.e., looking at each subclass), we only give the best obtained values. In addition to the best values, we give the parameter combinations used to obtain those values.

4 Realistic Testing

The goal in this section is to attack the implementation with our new approaches and assess their performance when compared to attacking immediately 256 classes. Here, we use 10 000 and 25 000 random measurements for all tests in order to have a fair comparison. The traces are divided uniformly at random in

Table 1: Testing results for 9 and 256 classes (Accuracy/F-measure/AUC)

	DPAcontest v4		DPAcontest v2	
Algorithm	Rotation Forest	SVM	Rotation Forest	SVM
9 classes				
Value	94.1/94.1/99.6	95.5/95.5/98.9 ($C = 70, \gamma = 0.5$)	25/19.8/50.2 ($Iter. = 60$)	23.67/19/50.1
256 classes				
Value	26.7/24.3/50.9	27.8/28/96.9 ($C = 70, \gamma = 0.5$)	0.36/0.4/50.4	0.45/0.4/51 ($C = 50, \gamma = 0.4$)

Table 2: Testing results for the hierarchical approach (Accuracy/F-measure/AUC)

Set	DPAcontest v4	DPAcontest v2
HW1	69.6/68.9/91.1 (SVM, $C = 4, \gamma = 0.6$)	15.2/15.2/5.3 (RF, $Iter. = 50$)
HW2	67.8/57.5/96.3 (SVM, $C = 10, \gamma = 0.7$)	4.0/2.2/51.7 (SVM, $C = 1, \gamma = 0.1$)
HW3	49.5/49.4/97.3 (SVM, $C = 10, \gamma = 0.9$)	2.0/4/52.2 (SVM, $C = 1, \gamma = 0.1$)
HW4	46.4/46.4/97.5 (SVM, $C = 20, \gamma = 0.8$)	1.7/7/51.8 (SVM, $C = 1, \gamma = 0.4$)
HW5	49.9/50.1/97.3 (SVM, $C = 10, \gamma = 1$)	2.0/0.9/50.4 (SVM, $C = 1, \gamma = 0.4$)
HW6	57.7/58.0/96.3 (SVM, $C = 10, \gamma = 0.7$)	3.8/1.1/50.1 (SVM, $C = 1, \gamma = 0.1$)
HW7	74.5/74.6/92.2 (SVM, $C = 4, \gamma = 0.7$)	13.2/12.1/4.9 (SVM, $C = 1, \gamma = 0.2$)

2:1 ratio where we use 2/3 of measurements for profiling and 1/3 for the testing. The best results in all tables are highlighted with gray background color of a cell.

Therefore, we investigate a number of cases here that all fall within three categories:

1. Attacking directly all 256 classes.
2. Attacking 9 classes (i.e., the Hamming weight classes).
3. New attacks - Hierarchical attack and Structured attack.

4.1 Hierarchical Attack

In the Hierarchical attack, one first investigates how to classify measurements into a (relatively) small number of classes, i.e., into subclasses (which can be repeated several times) and then, in the second step, the obtained classification results are further classified into leaves. However, since not all the measurements are correctly classified in the first step, they need to be discarded (since they belong to subclasses) and the total number of available measurements will be consequently lower than the number of instances we begin with. Note that the number of leafs is the same as if one considers the flat approach, but the classification method in each independent step considers a smaller number of classes. With this approach, we are able not only to improve the accuracy, but also to lower the computational and space complexity for the classification process. Finally, since we are running independent experiments on each of the subclasses, it is also easy to parallelize the attack, which may not be an option when considering the flat approach. We give more algorithmic description of the hierarchical attack in the following listing:

1. Find a hierarchical relation to explore.
2. Run a classifier for each level of subclasses.
3. Consider all instances classified above some threshold value as correctly classified (e.g., all instances that have a probability of more than 90% to be correctly classified into certain subclass), otherwise discard.
4. For all instances kept in a subclass, run new classifier in order to find in which subclass they belong (repeat until leaf class is reached).

In our case study we exploit the HW of an intermediate value (see Eqs. (1) and (3)) and thus first divide into 9 HW classes. Then, we use the measurement predictions from that phase to conduct an attack on the intermediate value itself (leaf class).

4.2 Structured Attack

In addition to the Hierarchical attack, we introduce an attack combination of the standard flat approach and our hierarchical approach, which we denote as the Structured attack. Accordingly, we merge the information from both approaches and even further improve the accuracy. This is due to a fact that, when attacking with the flat approach, we expect that the final accuracy will be lower than for the hierarchical attack, but there will be instances where the flat classification classifies correctly, while hierarchical classification makes a wrong prediction. This combination is of particular interest when the computing power and the runtime complexity is of no importance, but only the accuracy of the attack. We give a more precise listing for the Structured attack in the following:

1. Run classifier with the flat approach.
2. Run the Hierarchical attack.
3. Assign weight factors for each of the two aforesaid steps.
4. Combine results from flat and hierarchical approach. Similarly as in the Hierarchical approach, set a threshold value that signifies which classification guesses to take as true.

4.3 Attack Results and Comparison with Template Attack

In order to facilitate a better understanding of the obtained results, we also use two simpler machine learning techniques - Naive Bayes and Decision Tree (C4.5). The Naive Bayes algorithm does not have parameters one could tune and the tuning phase for Decision Tree highlights that the default values of parameters are the best. Therefore, we use a minimum amount of two instances per leaf and a confidence factor for pruning equal to $C = 0.25$. It is important to state that these two methods are extremely fast (especially the Naive Bayes) and we consider it to be beneficially to run them always as a first indicator of what can be expected from an ML approach. Recall, the parameter combinations for RF and SVM algorithms used in this section are obtained in Sections 3.2 and 3.3.

In Table 3, we present the results when working with 10 000 instances from the DPAcontest v2 and the DPAcontest v4 using 9 classes. Next, in Table 4, we present results for 10 000 instances using 256 classes. Note that the parameters for ML techniques are as presented in Table 1. As in the previous scenario, we can observe that SVM has the highest accuracy. Interestingly, Naive Bayes is more efficient than Decision Tree when working with 256 classes, but for 9 classes the situation is opposite. This is due to a jump of complexity appearing in the Decision Tree when the number of classes is increased. However, here we observe that Naive Bayes has the best accuracy when considering all 256 classes, which is a somewhat surprising result. We believe that a more extensive tuning phase for Rotation Forest and SVM would change this. However, we do not consider this completely justified when considering the huge difference in the runtime complexity between the Naive Bayes and these methods.

To rate the goodness of our achieved results, we additionally applied the template attack (TA) [1] to the same set of traces as used in the realistic attacking scenario and tested it for the standard approach of classifying 9 (see Tables 3 and 6) and 256 classes (see Tables 4 and 7) directly. TA is the most common and well-studied profiled side-channel attack and it is considered as the most powerful one from an information theoretic perspective given an infinite amount of measurements. However, compared to ML techniques, recent works showed its inferiority when, for example, the profiling set is not large enough or the attack is provided with too many useless (without information) points of interest (features) [23]. Actually, the attack principle is very close to the one of the Naive Bayes, where the main difference is the consideration of the features along with the measurements to be dependent. In particular, the Naive Bayes assumes independence and thus considers a univariate normal distribution. On the contrary, for TA the noise is considered dependent and thus a multivariate distribution is taken. For more details, we refer interested readers to [1]. To be more efficient and numerically stable, we applied the adaptation of using only one covariance matrix instead of 9 or 256 as described in [24]. Note that, for all the algorithms, we use the same datasets with the same feature selection process in order to make it as fair as possible (and to avoid seeing differences in results stemming from other than hierarchy causes, e.g. better feature selection).

The attack results of our new Hierarchical and Structured attack using 10 000 instances are given in Table 5. As both new attacks are considering the same leaf classes as in Table 4, we can directly compare their results. We note that in this set of experiments, we use the threshold value equal to 0.9, which represents that only the measurements with a high output probability of belonging to a certain class are taken as correctly classified. We can see that the Hierarchical SVM outperforms regular SVM, but the Structured SVM is by far the most efficient method for both the DPAcontest v4 and DPAcontest v2 scenario. We note that

Table 3: Attack scenario with 9 classes, 10 000 instances

Algorithm	# classes	Parameters	Training	Testing
		DPAcontest v4		
Naive Bayes	9	-	66.8	65.9/66.0/91.2
Decision Tree	9	$c = 0.25, M = 2$	70.1	71.8/71.8/85.2
SVM	9	$C = 70, \gamma = 0.5$	90.9	91.39/91.4/98
TA	9	-	-	76.71
		DPAcontest v2		
Naive Bayes	9	-	11.78	11/10.2/50.1
Decision Tree	9	$c = 0.25, M = 2$	19.36	20.58/20.4/50.7
Rotation Forest	9	$Iter = 60$	24.69	25.12/19.6/51.2
TA	9	-	-	8.31

Table 4: Attack scenario with 256 classes, 10 000 instances

Algorithm	# classes	Parameters	Training	Testing
		DPAcontest v4		
Naive Bayes	256	-	18.5	17.0/16.3/93
Decision Tree	256	$c = 0.25, M = 2$	13.4	13.2/13.2/58.5
SVM	256	$C = 70, \gamma = 0.5$	30.4	27.8/28/96.9
TA	256	-	-	20.19
		DPAcontest v2		
Naive Bayes	256	-	0.42	0.58/0.1/51.3
Decision Tree	256	$c = 0.25, M = 2$	0.28	0.36/0.3/49.9
SVM	256	$C = 50, \gamma = 0.4$	0.43	0.45/0.4/51
TA	256	-	-	0.39

the training phases for the Hierarchical and Structured attacks consist of training phases of the whole hierarchy (i.e., classes HW1 up to HW7) and therefore we give here the median value of the training accuracies. Note that we had only 6 700 instances to train for the Hierarchical attack, which gives on average 26 traces per class. When conducting the Hierarchical attack, we cannot use the whole test set (i.e., 3 300 instances), since some of them are wrongly classified when classifying into 9 classes. We give details about the available number of instances in notes below the table. For instance, the remark "From 3 016 correctly classified instances for 9 classes" means that after classifying into 9 classes, we have 3 016 correctly classified instances. Then, when classifying into subclasses we have only 3 016 instances in total and we see that in total 33.7% of those instances are correctly classified into intermediate values. In Tables 6, 7, and 8, we present results for 25 000 instances. When comparing the results achieved using 10 000 instances, we observe that the results of all attacks for the DPAcontest v4 are improving, whereas for the higher noise scenario from DPAcontest v2, the results are not changing noticeably when considering 9 and 256 classes (i.e., when working with standard ML techniques). Note that here we have on average 65 instances per class when training for hierarchical attack.

Table 5: Hierarchical and Structured attack, 10 000 instances

Algorithm	# classes	Parameters	Training	Testing
		DPAcontest v4		
Hierarchical attack	9/256	Table 2	38.23	31.36*
Structured attack	9/256	Tables 1 and 2	-	33.7
		DPAcontest v2		
Hierarchical ML	9/256	Table 2	2.95	1.32**
Structured ML	9/256	Tables 1 and 2	-	0.91

* From 3 016 correctly classified instances for 9 classes.

** From 829 correctly classified instances for 9 classes.

5 Discussion

In our examples, when conducting the hierarchical approach, we consider an extreme case: first dividing into 9 classes in accordance with the HW, and then dividing into all values for the corresponding HW. For some classes (HW 0 and 8) the hierarchical approach does not make a difference, since it is not possible to divide them any further. Contrary, for the Hamming weight class 4 contains 70 leaves, which is again a complex scenario. Therefore, one could instead use sets of two (or any other number) values that are mapped to the same class. For instance, in the Hamming weight class 4, values 23 and 27 can be grouped into a subclass. Then, in the next step, one uses a binary classification for those two values.

Table 6: Attack scenario with 9 classes, 25 000 instances

Algorithm	# classes	Parameters	Training	Testing (Acc./F-measure/AUC)
		DPAcontest v4		
Naive Bayes	9	-	70.01	67.85/67.9/91.7
Decision Tree	9	$c = 0.25, M = 2$	74.39	74.75/74.7/86.7
SVM	9	$C = 70, \gamma = 0.5$	93.83	94.32/94.3/98.6
TA	9	-	-	77.85
		DPAcontest v2		
Naive Bayes	9	-	8.21	8.1/10.2/50.3
Decision Tree	9	$c = 0.25, M = 2$	19.43	20.26/20.2/50.7
Rotation Forest	9	$Iter = 60$	25.15	24.71/19.5/50.4
TA	9	-	-	6.47

Table 7: Attack scenario with 256 classes, 25 000 instances

Algorithm	# classes	Parameters	Training (Acc.)	Testing (Acc./F-measure/AUC)
		DPAcontest v4		
Naive Bayes	256	-	20.44	20.27/18.4/94.5
Decision Tree	256	$c = 0.25, M = 2$	15.38	16.23/16.2/60.5
SVM	256	$C = 70, \gamma = 0.5$	35.54	35.02/35.1/98.1
TA	256	-	-	25.07
		DPAcontest v2		
Naive Bayes	256	-	0.65	0.5/0.1/50.8
Decision Tree	256	$c = 0.25, M = 2$	0.42	0.39/0.4/50
TA	256	-	-	0.4

Table 8: Hierarchical and structured attack, 25 000 instances

Algorithm	# classes	Parameters	Training	Testing
		DPAcontest v4		
Hierarchical attack	9/256	Table 2	44.01	40.74*
Structured attack	9/256	Tables 1 and 2	-	44.43
		DPAcontest v2		
Hierarchical ML	9/256	Table 2	2.92	1.69**
Structured ML	9/256	Tables 1 and 2	-	0.92

* From 7 844 correctly classified instances for 9 classes.

** From 2 066 correctly classified instances for 9 classes.

Using SVM with a flat classification, 256 classes, and measurements from the DPAcontest v4 (10 000 instances), we correctly classify 917 out of 3 300 instances (27.8%). When classifying into 9 classes, SVM reaches an accuracy of 91.4%, which translates into 3 016 correctly classified instances. The Hierarchical attack has an accuracy of 31.3%, which amounts to 945 instances that are correctly classified. Although the difference between the flat and hierarchical approach is small in this example, we note that we still improve the accuracy without using any extra information and with only a small overhead from the computational side. Even more so, when considering the Structured attack, the accuracy equals 33.7%, which is in total equal to 1 112 correctly classified instances. When compared to the flat approach, this amounts to 21% more instances that are correctly classified, which represents a significant improvement.

When considering the measurements of the DPAcontest v2, the classification is much more difficult. Indeed, when classifying into 9 classes, the accuracy is around 25% which results into 5 out of 9 classes having correctly classified instances. Therefore, to significantly improve the accuracy of the Hierarchical attack, we would need to use much more measurements. On the basis of the results from Table 5, it could seem that the Hierarchical attack has better accuracy (1.32%) than the Structured attack (0.91%) but that is actually not true. Indeed, when considering the Hierarchical attack, the accuracy can be calculated only from the number of instances that are correctly classified into subclasses (829 instances), which results in around 11 correctly classified instances. On the other hand, the accuracy of 0.91% for the Structured attack must be taken on the whole test set (3 300) which equals 30 correctly classified instances. Therefore, we see that the Structured attack offers a significant improvement over all other considered methods.

We emphasize that in order to obtain a fair comparison the Hierarchical and Structured attack must be compared with the 256 classes scenario, and not with the results from 9 classes. With the increase in the number of instances to 25 000, the superior performance of the Hierarchical and Structured attacks becomes even more apparent. For instance, when considering the measurements of the DPAcontest v4, SVM with the flat approach and classifying 256 classes has the accuracy of 35%, TA of 25%, and Structured attack of 44%.

On a more general level, we present here two novel attacks that are able to significantly increase the efficiency of ML techniques when compared to related work. Naturally, conducting the Hierarchical attack is more computationally expensive than just attacking the Hamming weight classes, and the Structured attack is even more expensive since one needs to use flat approach on all 256 classes as well as the Hierarchical attack. However, the increase in the number of experiments is well compensated with the increased accuracy of those methods. On the other hand, the hierarchical approach for ML techniques is beneficial from the runtime complexity side, since using smaller number of classes decreases the runtime of ML, while dividing experiments enables one to easily use parallel computing. The process of making a hierarchy is here considered to be simple and therefore its complexity is negligible. Naturally, this does not need always to happen, which would make our attack more complex in accordance with the process of finding the hierarchy.

When considering realistic settings, one does not know whether a classifier correctly classified certain instances into subclasses. Therefore, it is necessary to use a threshold which serves as a cut-off for all measurements below it. Naturally, the value of such a threshold is a parameter that can be tuned and that differs with respect to the underlying setting. For instance, measurements with smaller levels of noise can have higher threshold values since it is expected that the classifier will be able to classify certain instance with high probability of success. However, we note that the threshold level is in the end to be set by the attacker, with regards to how reliable he considers the classifier to be.

In this paper, we considered the HW of the intermediate value as a first level in the hierarchy. However, we do not claim that this choice is optimal or should be generally taken. Another approach would be to use the values of each bit of the intermediate value as a level of hierarchy, e.g., all the measurements where the first bit equals to 1 goes into one class and where the first bit equal to 0 into other class. Then, each of those subclasses has 128 subclasses. A more general approach would be, if some hierarchical structure is not readily observable, to build a hierarchy with the automatic generation of subclasses, where algorithm groups leaf classes by their similarity [13].

Finally, we give several observations why the hierarchical attack might improve the accuracy in some cases. The first reason is because we use a priori knowledge about the dataset (i.e., we know the semantic hierarchy). Naturally, this can also be a source of mistake, where the question is how severe would a (slightly) wrong hierarchy influence the results. Since in our experiments, we use only two levels of hierarchy, then consequently, the propagation of error in the classification cannot go far. The second reason why hierarchical attacks improve the accuracy over flat approach is that they can limit the model complexity and constrain the expressiveness of a hypothetical class. We leave for future research the experiments showing which reason has more influence on success in these scenarios. Moreover, it would be interesting to explore how robust is the hierarchical classification when the hierarchy does not model the data completely. Still, we emphasize that the complexity of classification for each subclass and the corresponding subclasses is lower than in the case of the flat classification (since most of the algorithms have complexity increasing linearly with the number of classes).

As future work, we are interested in exploring how hierarchical and structured approaches behave when using a larger number of instances. Moreover, we observe that in the hierarchical approach, wrongly classified measurements often exhibit some structure (e.g., the measurements belonging to one class are dominantly classified as belonging to some other class) and we would like to investigate the automatic generation of classes (similar to [25]). With such an approach, we expect to find some new subclasses that can be used in the hierarchical approach.

6 Conclusions

In this paper, we introduced the concept of hierarchical machine learning classification for side-channel analysis. Instead of attacking immediately the sensitive variable or just the Hamming weight of it, we propose to use a divide-and-conquer approach in a form of class hierarchy. To show the practicability of our new approach, we conducted our analysis on two publicly available data sets from the DPAcontest with different SNRs and made a comparisons to machine learning techniques and the template attack using the standard (flat) approach. Our results show that, for both data sets, the Hierarchical and Structured attacks outperform other ML approaches as well as the template attack. Aside from the better accuracy with our hierarchical approach, an additional advantage is also the lower computational complexity for ML techniques, which renders more plausible such attacks when using realistic data sets with large number of measurements and points in time.

Acknowledgments

S. Picek was supported in part by Croatian Science Foundation under the project IP-2014-09-4882.

References

1. 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.
2. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In LNCS, ed.: CHES. Volume 3659 of LNCS., Springer (Sept 2005) 30–46 Edinburgh, Scotland, UK.
3. Lerman, L., Bontempi, G., Markowitch, O.: Side Channel Attack: an Approach Based on Machine Learning. In: Second International Workshop on Constructive SideChannel Analysis and Secure Design, Center for Advanced Security Research Darmstadt (2011) 29–41
4. 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 10.1007/s13389-011-0023-x.
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. Lerman, L., Bontempi, G., Markowitch, O.: The bias-variance decomposition in profiled attacks. J. Cryptographic Engineering **5**(4) (2015) 255–267
7. Lerman, L., Bontempi, G., Markowitch, O.: Power analysis attack: an approach based on machine learning. IJACT **3**(2) (2014) 97–115
8. 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.

9. Heuser, A., Kasper, M., Schindler, W., Stöttinger, M.: A New Difference Method for Side-Channel Analysis with High-Dimensional Leakage Models. In Dunkelman, O., ed.: CT-RSA. Volume 7178 of Lecture Notes in Computer Science., Springer (2012) 365–382
10. TELECOM ParisTech SEN research group: DPA Contest (2$^{\text{nd}}$ edition) (2009–2010) `http://www.DPAcontest.org/v2/`.
11. Xilinx: Virtex-5 libraries guide for hdl designs http://www.xilinx.com/support/documentation/sw_manuals/ xilinx14_4/virtex5_hdl.pdf.
12. TELECOM ParisTech SEN research group: DPA Contest (4$^{\text{th}}$ edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
13. de Almendra Freitas, C.O., Oliveira, L.S., Aires, S.B.K., Bortolozzi, F.: Metaclasses and Zoning Mechanism Applied to Handwriting Recognition. J. UCS **14**(2) (2008) 211–223
14. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning **29**(2) (1997) 131–163
15. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
16. Frank, E., Witten, I.H.: Generating Accurate Rule Sets Without Global Optimization. In Shavlik, J., ed.: Fifteenth International Conference on Machine Learning, Morgan Kaufmann (1998) 144–151
17. Rodriguez, J.J., Kuncheva, L.I., Alonso, C.J.: Rotation forest: A new classifier ensemble method. IEEE Trans. Pattern Anal. Mach. Intell. **28**(10) (October 2006) 1619–1630
18. Kuncheva, L.I., Rodríguez, J.J.: An experimental study on rotation forest ensembles. In: Proceedings of the 7th International Conference on Multiple Classifier Systems. MCS'07, Berlin, Heidelberg, Springer-Verlag (2007) 459–468
19. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer-Verlag New York, Inc., New York, NY, USA (1995)
20. Platt, J.: Fast Training of Support Vector Machines using Sequential Minimal Optimization. In Schoelkopf, B., Burges, C., Smola, A., eds.: Advances in Kernel Methods - Support Vector Learning. MIT Press (1998)
21. 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
22. Powers, D.M.W.: Evaluation: from precision, recall and F-factor to ROC, informedness, markedness and correlation (2007)
23. 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
24. 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
25. Whitnall, C., Oswald, E.: Robust Profiling for DPA-Style Attacks. In Güneysu, T., Handschuh, H., eds.: Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Proceedings. Volume 9293 of Lecture Notes in Computer Science., Springer (2015) 3–21