# Semi-Supervised Template Attack

Liran Lerman[1,2], Stephane Fernandes Medeiros[1], Nikita Veshchikov[1], Cédric Meuter[3], Gianluca Bontempi[2,⋆], and Olivier Markowitch[1]

[1] Quality and security of Information Systems, Département d'informatique, Université Libre de Bruxelles, Belgium
[2] Machine Learning Group, Département d'informatique, Université Libre de Bruxelles, Belgium
[3] Atos Worldline, Belgium

**Abstract.** Side channel attacks take advantage of information leakages in cryptographic devices. Template attacks form a family of side channel attacks which is reputed to be extremely effective. This kind of attacks assumes that the attacker fully controls a cryptographic device before attacking a similar one. In this paper, we propose to relax this assumption by generalizing the template attack using a method based on a semi-supervised learning strategy. The effectiveness of our proposal is confirmed by software simulations, by experiments on a 8-bit microcontroller and by a comparison to a template attack as well as to two supervised machine learning methods.

**Keywords:** Side channel attack, Template attack, Power analysis, Semi-supervised learning, Clustering, Hamming weight.

## 1 Introduction

Side Channel Attacks (SCA) take advantage of the fact that instantaneous power consumption [18], execution time [17] or/and electromagnetic emanations leakages [10] of a cryptographic device depend on the processed data and the performed operations. Power analysis attack is a type of SCA which assumes that the use of different keys implies differences in the power consumption. The evolution along the years of the power analysis attacks has been characterized by an increase in the complexity of the statistical analysis.

Simple Power Analysis (SPA) [18] was the first proposed approach to perform power attacks. It relies on an interpretation of the trace (of the power consumption) in order to retrieve information about the used key. In other words, the attacker tries to derive (a part of) the key directly from a given trace by observing patterns in it. For example, Hollestelle et al. [14] showed that such attack against RSA implemented with a square and multiply algorithm allows the recovery of the key. Differential Power Analysis (DPA) [18] uses more advanced statistical analysis than SPA by modeling the theoretical power consumption for each key.

⋆ Supported by the Belgian French Community ARC funding.

E. Prouff (Ed.): COSADE 2013, LNCS 7864, pp. 184–199, 2013.

The likelihood of the observed power consumption for each model is used to predict the key.

Template Attacks (TA) [4] make an additional step by estimating the conditional probability of the trace for each key during a profiling step. It extracts all available information from each trace and can be considered as the strongest form of side channel attack under the assumption that the distribution of traces is known [4]. This kind of attacks is feasible if the attacker can access the values of the keys during the profiling step.

Batina et al. [2] presented the Differential Cluster Analysis (DCA), a variant of DPA, against a cryptographic device using an unknown fixed subkey. This technique uses cluster analysis to detect internal collisions in the collected traces. It builds a cluster for each value of a target (i.e. a function of the cryptographic algorithm that handles the guessed key and a known value like the plaintext). In the second step it regroups traces of the target device having the same value. Finally it assesses the quality of the cluster separation by means of the "Sum-Of-Squared-Error" criterion. However, DCA requires that the key does not change during the attack.

Lerman et al. [19,20], Hospodar et al. [15,16], and Heuser et al. [13] discussed the role of machine learning in TA. They showed that a supervised machine learning procedure is able to outperform conventional TA. However their models suppose that in the profiling step the attacker can have a full control of a device identical to the attacked one. In other words, the attacker can choose both key and plaintext.

Dyrkolbotn et al. [8] targeted the precise Hamming weight of data with template attack. However, their research was based on a supervised method where they supposed that they can also fully control a clone device in order to build their classification model.

### 1.1 Our Contribution

This paper intends to make one further step on all of these previous works by generalizing the TA. This generalization is made by relaxing the hypothesis that the attacker has a full control of the device. It is performed thanks to a machine learning approach inferring the precise Hamming weight of the subkey. More precisely, in the machine learning domain, supervised learning is concerned with the task of inferring a stochastic dependency from observed data. In a SCA context, the purpose is to infer a secret information from a set of measured traces. Unsupervised learning is any kind of model which tries to find hidden structures in traces without knowing their respective keys. Our work introduces Semi-Supervised Template Attack (SSTA) which combines supervised and unsupervised learning [34]. In our approach the attacker needs to know the Hamming weights of a small subset of all the set of possible target keys and their related power consumption.

The results of this attack can be used as an input for other related attacks such as algebraic side channel attacks combined with a template attack approach [25].

We have analyzed the pertinence of our attack with simulations and real data experiments. Furthermore we compared SSTA to several profiling Side Channel Attacks: a template attack as well as to two supervised machine learning methods. Our evaluations show significant key-recovery success rates.

This paper is organized as follows: Section 2 introduces the notations and the basics of TA approach. Section 3 presents our machine learning approach applied to power analysis attack. A description of the experimental system and the results of an attack based on a machine learning technique are described in Section 4. Section 5 concludes the paper on a positive note and discusses future works.

## 2 Template Attack

Let us consider a crypto device executing a cryptographic algorithm with the binary key $O_k, k = 1, \ldots, K$, where $K = 2^B$ is the number of possible values of the key and $B$ the number of bits of the key. For each key we observe $N$ times over a time interval of length $n$ the power consumption of such device and we denote by *trace* the series of observations $T_{(i)}^{(k)} = \left\{ T_{(i)(t)}^{(k)} \in \mathbb{R} \mid t \in [1; n] \right\}$, $i = 1, \ldots, N$ associated to the $k$-th key. The state-of-the-art TA [4] modelizes the stochastic dependency between the key and a trace by means of a multivariate normal conditional density:

$$P(T_{(i)}^{(k)} | O_k; \mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_k|}} e^{-\frac{1}{2}(T_{(i)}^{(k)} - \mu_k)\Sigma_k^{-1}(T_{(i)}^{(k)} - \mu_k)^\top} \tag{1}$$

where $\mu_k \in \mathbb{R}^n$ and $\Sigma_k \in \mathbb{R}^{n \times n}$ are respectively the expected value and the covariance of the $n$ variate traces associated to the $k$-th key.

During the profiling step, a set of $N$ traces $T_{(i)}^{(k)}, i = 1, \ldots, N$, is collected for each key. TA estimates the expected value $\mu_k$ and the covariance $\Sigma_k$ by:

$$\hat{\mu}_k = \frac{1}{N} \sum_{i=1}^{N} T_{(i)}^{(k)} \tag{2}$$

and:

$$\hat{\Sigma}_k = \frac{1}{N-1} \sum_{i=1}^{N} (T_{(i)}^{(k)} - \hat{\mu}_k)^\top (T_{(i)}^{(k)} - \hat{\mu}_k) \tag{3}$$

During the attacking step, once a new trace $T$ is observed on the attacked device, TA returns the key which maximizes the likelihood:

$$\hat{k} = \arg\max_k \hat{P}(T|O_k) = \arg\max_k P(T|O_k; \hat{\mu}_k, \hat{\Sigma}_k) \tag{4}$$

This approach makes implicitly the assumption that the distribution of the traces for a given key follows a parametric Gaussian distribution, with $(n^2 + 3n)/2$ parameters.

We consider the power consumption $T^{(k)}_{(i)(t)}$ of the device at time $t$ depending on an internal value $f_k(x)$ where $x$ is (a part of) the plaintext and $k$ is (a part of) the key. In other words, we have that

$$T^{(k)}_{(i)(t)} = L(f_k(x)) + \epsilon \tag{5}$$

where $L$ is the data-dependent device leakage and $\epsilon$ is the independent random noise following a Gaussian distribution with zero mean.

A lot of power analysis found in literature are based on the Hamming weight (HW) model [8, 24, 27, 30, 32]. More precisely, that model assumes that $L$ is proportional to the Hamming weight of the internal value (i.e. $T^{(k)}_{(i)(t)} = \lambda \mathrm{HW}(f_k(x)) + \xi + \epsilon$ with $\lambda \in \mathbb{R}$ and $\xi \in \mathbb{R}$).

## 3 The SSTA Approach

As seen in the previous section, template attacks are performed in two steps: a profiling step (aka. learning step) and an attacking step (aka. validation or test step). The state-of-the-art approach of TA assumes that during the profiling step the attacker can fully control (choose the plaintext and the key) a copy of a device he wants to attack. By changing the key (or a part of it) the attacker can build a template for each value of the (sub)key.

In this paper we propose to relax this restrictive hypothesis (full control of a cloned device). The attacker only needs to control the attacked device with several known keys but he can still collect traces from the device with unknown keys.

This section discusses Semi-Supervised Template Attack (SSTA) approach.

### 3.1 The SSTA Approach

In SSTA the attacker collects traces measured when the device executes cryptographic operations. He can collect these traces while the device is used by different users having different keys. We suppose that the device is such that each user is linked to a particular key. In other words the attacker cannot change the key but he knows that the same fixed key is used when he manipulates the device. We suppose also that the keys are uniformly distributed which is relevant in case of symmetric ciphers (this is not necessarily the case in asymmetric algorithms). In SSTA, during the acquisition step, the attacker collects two sets of traces. The first set of traces $T^{(k)}_{(i)} \in \mathcal{T}^{\mathcal{K}}$ is collected using several known keys. The second set of traces $T_{(i)} \in \mathcal{T}^{\mathcal{U}}$ is acquired with unknown keys.

For the sake of simplicity we restrict to consider attacks on the Hamming weight of a single byte of the key. We assume that the power consumption is dependent on the Hamming weight of the manipulated data.

Suppose that the cryptographic device manipulates the $b$-th byte of the key $k$ (namely $k_b$) at time $t$ (i.e. $T^{(k)}_{(i)(t)} = \lambda \mathrm{HW}(k_b) + \xi + \epsilon$). At the moment $t$ when

the (sub)key is manipulated by the device, the traces linked to two (sub)keys having the same Hamming weight must be more similar than if (sub)keys had different Hamming weights. Therefore, in the first part of the profiling step, the attacker regroups traces from set $\mathcal{T}^{\mathcal{U}}$ that have similar power consumption when the cryptographic device manipulated the $b$-th byte of the key. It is done using the advantages offered by machine learning techniques [12]. More precisely, clustering techniques allow us to find all traces that have the same Hamming weight in order to group them into clusters.

In the second step, the attacker has to find the value of the Hamming weight of each cluster. For this, we have to consider the density distribution of the Hamming weights of a byte. There is asymptotically $\frac{1}{256}$ of the set of traces that is measured when the cryptographic device used a key with a Hamming weight of 0 or 8, $\frac{8}{256}$ of the set of traces that is measured when the cryptographic device used a key with a Hamming weight of 1 or 7, etc. So, the attacker can recover the value of the Hamming weight of each cluster by observing the relative number of traces and their energy consumption in each cluster (we suppose that the power consumption of the attacked device is a strictly increasing or a strictly decreasing function of Hamming weight of the key). For example, in order to distinguish the clusters 0 from 8, considering that the energy consumption is a strictly increasing function, the first cluster will have a lower power consumption than the second cluster.

Finally, in the attacking step, the attacker measures a trace $T$ on the attacked device. Afterwards the model returns the estimated Hamming weight $\hat{h}$ of $k_b$ which minimizes:

$$\hat{h}, \hat{i} = \arg\min_{h,i} d_t(T, T_{(i)}^{(h)}) \tag{6}$$

where $d_t$ is a distance measure (e.g Euclidean or Manhattan distance) between two traces on the instant $t$ (where $k_b$ is manipulated). In our case, the distance measure $d_t$ is the Euclidean distance.

Note that since the trace could be misclassified the model can give a wrong Hamming weight. In order to handle this issue the attacker can proceed by local search (e.g. trying $\hat{h}+1$ and $\hat{h}-1$ in the neighborhood of $\hat{h}$).

Once the Hamming weight is known, the attacker can find the value of the attacked byte by brute-force attack (i.e. try each key which has the Hamming weight $\hat{h}$). Brute-force enumeration, in case Hamming weight is known, would require less attempts than the classical brute-force which enumerates all 256 possible values. We can also use this Hamming weight as an input to other attacks such as an Algebraic Side Channel Attack [25] or a classical DPA combined with a template attack approach [26].

Furthermore after the classification of a trace from the attacked device, this trace can be added in the model in order to improve the accuracy of further executions. In other words, the model may improve its success rate at each execution of the attacked device by inserting the new trace in the set of traces used in the profiling step.

Up to now, we supposed that the instant $t$ when the device manipulates the attacked byte is known. In order to find this instant $t$, we suppose that the attacker has a small subset of keys of known Hamming weights. Thanks to methods of dependency (e.g. Pearson correlation, Kolmogorov-Smirnov) the attacker can find the instant where the cryptographic device manipulates the key. More precisely, SSTA choose the instant $t$ in the set of traces $\mathcal{T}^{\mathcal{K}}$ which has the highest dependence with the known Hamming weights of the keys. In our experiment we chose the instant $t$ that maximizes the mutual information between traces at instant $t$ and known Hamming weights.

Finally, Algorithm 1 (in appendix A) gives a pseudo-code of SSTA on a single byte of the key.

### 3.2 Assumptions

In this section we will discuss main issues which are important for the profiling step: the probability of having several keys of different Hamming weights and the required number of traces to collect in order to have at least one trace per Hamming weight.

**Probability of Having Several Keys of Different HW.** We suppose that the attacker has several keys of different Hamming weights. In order to find the best point in the collected traces, during the profiling step, the attacker has to

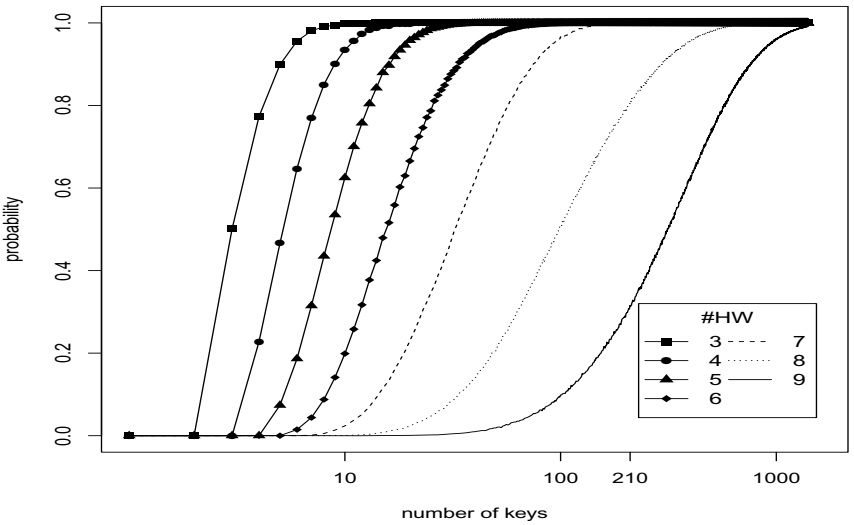



**Fig. 1.** Empirical probability of having several one byte sub-keys of different Hamming weights. It shows the probability of having 3 to 9 different Hamming weights when the attacker has 0 to 1500 different keys. The X axis is scaled logarithmically.

obtain as many keys of different Hamming weights as possible. We assume that the attacker cannot either set nor choose values of these keys. Figure 1 shows the probability that the attacker has several keys of different Hamming weights depending on the total number of keys he has. From Figure 1 we observe that an attacker needs few known keys if he wants 3 keys of different Hamming weights while a larger number of known keys is required if all the Hamming weights are needed. For example the attacker has 94% of chance to have 4 keys of different Hamming weights from a set of 10 randomly chosen known keys.

**Number of Traces to Collect in Order to Have at Least One Trace per HW.** Once the attacker knows when the key is manipulated he should collect traces in order to build the model. We estimate here how many traces should be collected. We must have at least one trace per value of Hamming weight in order to build each template.

The probability $p$ to obtain at least one trace per value of Hamming weight is:

$$p = P(G_0 > 0 \wedge G_1 > 0 \wedge G_3 > 0 \wedge ... \wedge G_8 > 0) \tag{7}$$

where $G_i$ is the number of traces linked to a key which has a Hamming weight of $i$.

Under the hypothesis of independency, the equation (7) can be rewritten as:

$$p = P(G_0 > 0) \times P(G_1 > 0) \times P(G_3 > 0) \times ... \times P(G_8 > 0) \tag{8}$$

where $P(G_i > j)$ follows a Binomial distribution with two parameters (the number of trials and the success probability in each trial): $n$ and $p_i$. Based on the equation (8) the Figure 1 shows the number of traces $n$ which should be collected depending on $p$. This figure shows that the attacker has to collect at least 1226 traces in order to have 99% of chance to have at least one trace per value of Hamming weight. Since the same traces can be used in order to attack any byte of the key, we need overall 1226 traces in order to find a key of any length.

## 4 Experiments

We validate our approach by conducting software simulations of our attack. During software simulations, physical leakages targeted by this attack (i.e. key manipulation) were simulated as the Hamming weight of the attacked part of the key.

In order to confirm the results of these simulations, we performed real data experiments: we attacked the initial AddRoundKey and the first round of AES [6] implemented on a microcontroller.

Afterwards we compared the SSTA approach with the TA approach and with two supervised machine learning methods based on the same datasets.

**Validation**
In order to assess the quality of SSTA, we adopt a holdout validation technique [1]. This technique needs two sets of traces. The first one (learning set) is used to build the model (i.e. find the best point in a trace and build each template). The second one (validation set) is used in order to assess the generalization accuracy. Typical values for a learning set are between 70% and 90% of all dataset and the rest is used for the validation set.

### 4.1 Software Simulations

In this first step of our experiment, we simulated the power consumption of an 8-bit microcontroller. Therefore we considered the case of an univariate problem where each trace has one point linked to the Hamming weight of an 8-bit key.

The estimation of the success rate[1] of the attack depending on the noise in the traces is shown in Figure 2. Note that the standard deviation is low compared to the average success rates. An `R` implementation of "Partitioning Around Medoids" [28][2] clustering technique was used. The complexity of this clustering algorithm is $O\big(i \times k \times (N-k)^2\big)$ [29, 31] where $i$ is the number of iterations, $k$ is the number of clusters and $N$ is the number of traces. The holdout technique used 3500 traces in the learning set and 1500 in the validation set. As expected we can see that the more noise we have, the more difficult it is to succeed the attack.

### 4.2 Real Data Experiments

After a simulation where each parameter can be controlled, we performed two real data experiments. The first one was meant to test SSTA on a simple case. The second experiment is a generalization of the first one. Table 1 summarizes both experiments.

**Measurement Setup**
We used an 8-bit microcontroller ATmega328P. The microcontroller was programmed using an Arduino Uno board [33]. In order to cut off any noise (generated by other parts of the Arduino Uno board's circuits) the microcontroller was removed from the board and placed on an external protoboard. The microcontroller was powered up using 5.3 V supply. It used an external 16 MHz clock. We measured the power consumption by inserting a 47 $\Omega$ resistor on the power pin $V_{CC}$ of ATmega328P. For all acquisitions we used an Agilent Infiniium 80000B Series oscilloscope 2 GHz 40 GSa/s (maximum capabilities). In practice we used 2 GHz and 250 MSa/s settings. See the acquisition scheme in Figure 4 in Appendix A.

[1] Which is the average of 10 success rates obtained in same conditions (i.e. with the same value of the standard deviation of the noise).
[2] Package "cluster" [23] available on CRAN.

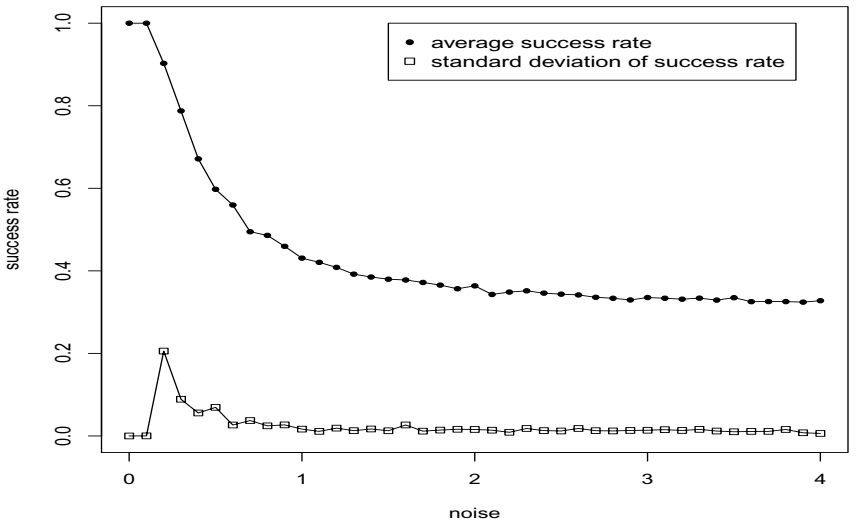



**Fig. 2.** Average success rates depending on the value of the noise. The noise is the value of standard deviation of $\epsilon$ in equation (5). It was generated using 10 experiments on a simulated device. "Partitioning Around Medoids" clustering technique was used.

**First Experiment**

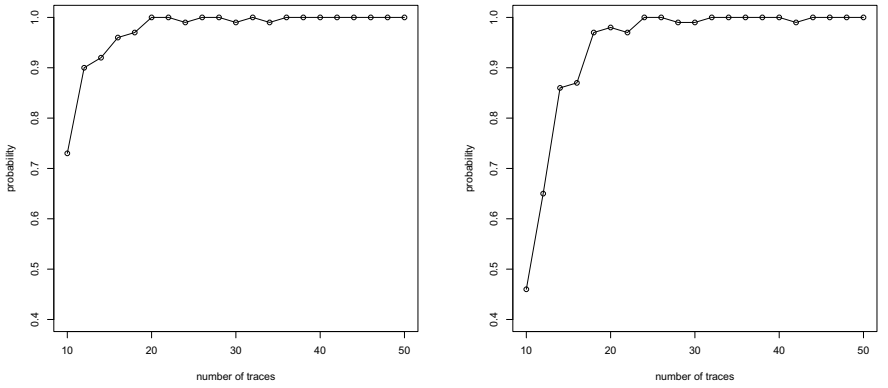
The attack was performed with a plaintext of 16 bytes (all set to zero). All 16 bytes of the key (except the attacked byte) were also set to zero. The reason of these fixed values is to decrease the complexity of subsequent steps of the analysis. We chose to attack the 13-th byte of the key[3] which value ranges between 0 to 255.

For each value of the attacked byte we performed 10 average acquisitions. By average acquisition we mean the average, performed on the oscilloscope, of 128 single acquisitions using the same key and plaintext. The parameters 10 and 128 were chosen on the basis of practical and logistical issues.

The result of a measure (an average of 128 acquisitions) is a trace such as the one plotted in Figure 5 in Appendix A. Traces were aligned using a trigger on the "encipher" signal sent to the device. We achieved a preprocessing step which selected each fifth point in each trace.

Two different signals (handled using interruptions) could be sent to the microcontroller, one to change the value of the attacked byte of the key (i.e. to increment its value), the other to order to encipher the plaintext using the current value of the key.

[3] This value has been randomly chosen and do not impact on the results because of the 8-bit architecture.




(a) First experiment (b) Second experiment

**Fig. 3.** Empirical probability to find the best point in a trace depending on the size of $\mathcal{T}^{\mathcal{K}}$. In each experiment, repeated 100 times, the set $\mathcal{T}^{\mathcal{K}}$ was randomly chosen.

We performed our experiment based on the "holdout" technique with a learning set of 8 traces per byte value and with 2 traces per byte value in the validation set[4]. The best point found by the feature selection model was located at the end of the initial AddRoundKey, which is the moment in time where the attacked key byte (i.e. the 13-th) is manipulated. The best point is the point which is the most correlated to the target value. In our case we used mutual information[5] between the energy consumption and the HW of the key.

Figure 3(a) shows the probability to find the best point in a trace depending on the number of traces in $\mathcal{T}^{\mathcal{K}}$. It is worth to note that a small subset of the set of traces is enough in order to find the best point in each trace. For instance there is more than 95% chances to find the best point using only 20 traces linked to known keys.

In this experiment we also used the "Partitioning Around Medoids" clustering technique. The result of our attack is shown in Table 1

The success rate is 61.5% which is higher than the success rate of the naive model which is based on the probability density function of Hamming weight. This model will always give the answer 4 for the Hamming weight of a byte, since it is the most probable value. Its success rate is $\frac{70}{256} \approx 27.34\%$ (see Table 2).

The success rate of our attack can be improved by allowing a misclassification. Our model approximates the Hamming weight 89.77% of the time with an error of at most 1 (i.e. an error of at most 1 is represented by $|h - \hat{h}| \leq 1$ where $h$ is the actual Hamming weight of a byte and $\hat{h}$ is the Hamming weight given by our model). Moreover, our model approximates the Hamming weight with a success of 100% with an error of at most 2.

[4] Standard values for theses parameters.
[5] Package "*sideChannelAttack*" [21] available on CRAN.

**Second Experiment**
The second experiment is more general than the first one and it is meant to confirm the results of the first one by using random values for keys, by using fewer keys and by allowing noisier traces.

We used the same settings and same devices as in the first experiment, apart from what discussed below.

All bytes of the key were randomly generated. We generated 210 different keys. Based on Figure 1, 31.33% bytes of the key should have all possible Hamming weight values (i.e. we should be able to attack 5 bytes). In practice we were able to attack 6 bytes of the key.

The attack was performed with fixed plaintexts of 16 bytes. This times we chose to attack the 15-th byte of the key.

In this experiment we performed 10 average acquisitions per key/plaintext pair. Each acquisition is the average of 100 single acquisitions using the same key and plaintext.

Figure 3(b) shows the probability to find the best point in a trace depending on the number of traces in $\mathcal{T}^{\mathcal{K}}$. The results are similar to the first experiment. However, in this new experiment, the attacker needs more traces compared to the previous experiment due to noisier traces.

The result of this experiment is shown in Table 1. Using 210 traces the model gives the correct Hamming weight in 53% of cases. Moreover, 85.31% of the time our model approximate the Hamming weight, with an error of at most 1.

### 4.3 Discussions

The experimental results of the previous section suggest that the SSTA is a practical attack. We compare the SSTA approach with several profiling Side

**Table 1.** Success rates of experiments of SSTA, TA, RF (Random Forest) [3] and SVM (Support Vector Machine) [5]. LS/VS is the number of traces in the learning set and in the validation set.

| Model | Experiment | LS/VS | #Traces for average | #Keys | Success rate (%) | | |
|---|---|---|---|---|---|---|---|
| | | | | | $\lvert h - \hat{h}\rvert = 0$ | $\lvert h - \hat{h}\rvert \leq 1$ | $\lvert h - \hat{h}\rvert \leq 2$ |
| SSTA | First | 8/2 | 128 | 256 | 61.5 | 89.77 | 100 |
| | Second | 8/2 | 100 | 210 | 53.0 | 85.31 | 100 |
| TA | First | 8/2 | 128 | 256 | 84.18 | 99.80 | 100 |
| | Second | 8/2 | 100 | 210 | 71.56 | 98.58 | 100 |
| RF | First | 8/2 | 128 | 256 | 77.54 | 99.61 | 100 |
| | Second | 8/2 | 100 | 210 | 73.46 | 99.53 | 100 |
| SVM | First | 8/2 | 128 | 256 | 83.98 | 100 | 100 |
| | Second | 8/2 | 100 | 210 | 78.91 | 100 | 100 |

**Table 2.** Success rate of a Naive model

| Naive model Success rate (%) | | | | |
|---|---|---|---|---|
| $\lvert h - \hat{h}\rvert = 0$ | $\lvert h - \hat{h}\rvert \leq 1$ | $\lvert h - \hat{h}\rvert \leq 2$ | $\lvert h - \hat{h}\rvert \leq 3$ | $\lvert h - \hat{h}\rvert \leq 4$ |
| 27.34 | 71.09 | 92.97 | 99.22 | 100 |

Channel Attacks. The first one is an univariate TA where the best point was also found thanks to a mutual information approach. Thus we performed a TA against the same datasets used in both experiments presented previously. The results are shown in Table 1. The performance of TA is better in each experiment. For example in the first experiment the success rates increase in average by 10.90 percentage points while in the second experiment the success rates increase in average by 10.61 percentage points.

Another comparison was performed against two state-of-the-art univariate supervised machine learning methods: the Random Forest[6] and the Support Vector Machine[7]. The results are shown in Table 1. The table shows that these models are as competitive as the TA.

Performances of TA, RF and SVM can be explained by their profiling step. Indeed these approaches assume that the attacker has a full control of (a copy of) the cryptographic device which is not the case of SSTA.

It is worth to note that the success rate of SSTA could be improved by using the knowledge of the set of traces $\mathcal{T}^{\mathcal{K}}$ during the clustering step. Indeed, this prior information can allow to improve the clustering method by the fact that we know in which cluster should be placed each trace $T_{(i)}^{(k)} \in \mathcal{T}^{\mathcal{K}}$.

## 5 Conclusions

We presented and assessed a generalization of the template attack, based on a semi-supervised technique of machine learning, able to infer a model from power consumption observations. This model predicts the Hamming weight of the attacked byte of a key. The contribution is done on the profiling step where the attacker needs to know only a subset of keys of different Hamming weights. We implemented our technique in a simulated and real data settings. In both cases we show that we can retrieve the Hamming weight of a byte of the key faster than the naive model.

The added value of SSTA compared to TA and to supervised machine learning approaches is that SSTA has less constraints. Indeed TA, RF and SVM suppose

[6] We used 500 trees in the RF which is the default value in the used implementation available on CRAN [22].

[7] We used the radial kernel in the SVM which is the default kernel in the implementation available on CRAN [7].

that the attacker has full control of the device during the profiling step which is not the case of SSTA. Therefore SSTA offers good performances, thus can be used in practice, while having less constraints.

We analyzed the limits of our attack by (1) estimating the probabilities that the attacker has as subset of keys with different Hamming weights and (2) by computing the number of traces to be collected in order to complete a real data attack.

Future works will focus on the generalization of these preliminary results: firstly by assessing the impact of the plaintext on the prediction accuracy, secondly by considering larger parts of the key, thirdly by considering all rounds of AES as well as other algorithms and by varying the cryptographic device and its architecture. Another idea is to build templates with different devices for each key. Interesting future research perspectives are also to consider other univariate clustering methods [9] the adaptation of clustering multivariate techniques [11] as well as specific feature selection techniques for the dimensionality reduction of traces. We would also like to relax the hypothesis of the fact that the attacker knowns a subset of keys during the profiling step. Another possibility is to use the SSTA as a preprocessing to other attacks such as an Algebraic Side Channel Attack [25] or a classical DPA [26]. Eventually, protected implementations against this kind of analysis will be investigated.

**Acknowledgements.** The authors would like to thank Atos Worldline and especially Filip Demaertelaere for his support.

## References

1. Alpaydin, E.: Introduction to Machine Learning, 2nd edn. The MIT Press (2010)
2. Batina, L., Gierlichs, B., Lemke-Rust, K.: Differential Cluster Analysis. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 112–127. Springer, Heidelberg (2009)
3. Breiman, L.: Random forests. Machine Learning 45, 5–32 (2001)
4. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
5. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning, 273–297 (1995)
6. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
7. Dimitriadou, E., Hornik, K., Leisch, F., Meyer, D., Weingessel, A.: e1071: Misc Functions of the Department of Statistics (e1071), TU Wien. R package version 1.6 (2011)
8. Dyrkolbotn, G.O., Snekkenes, E.: Modified Template Attack: Detecting Address Bus Signals of Equal Hamming Weight. In: Ntnu, S.F.M. (ed.) The 2nd Norwegian Information Security Conference (NISK 2009), pp. 43–56. Tapir Akademisk Forlag (November 2009)

9. Gan, G., Ma, C., Wu, J.: Data clustering - theory, algorithms, and applications. Series on Statistics and Applied Probability. SIAM, Society for Industrial and Applied Mathematics (2007)
10. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
11. Hardle, W., Hlavka, Z.: Multivariate Statistics: Exercises and Solutions, 1st edn. Springer Publishing Company, Incorporated (2007)
12. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd edn. Springer Series in Statistics. Springer (September 2009)
13. Heuser, A., Zohner, M.: Intelligent machine homicide. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 249–264. Springer, Heidelberg (2012)
14. Hollestelle, G., Burgers, W., Hartog, J.I.D.: Power Analysis on Smartcard Algorithms Using Simulation. Imported from DIES (2004)
15. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. J. Cryptographic Engineering 1(4), 293–302 (2011)
16. Hospodar, G., De Mulder, E., Gierlichs, B., Vandewalle, J., Verbauwhede, I.: Least Squares Support Vector Machines for Side-Channel Analysis. Center for Advanced Security Research Darmstadt, pp. 99–104 (2011)
17. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
18. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
19. Lerman, L., Bontempi, G., Markowitch, O.: Power analysis attack: an approach based on machine learning. International Journal of Applied Cryptography (to appear)
20. Lerman, L., Bontempi, G., Markowitch, O.: Side Channel Attack: an Approach Based on Machine Learning. Center for Advanced Security Research Darmstadt, pp. 29–41 (2011)
21. Lerman, L., Bontempi, G., Markowitch, O.: sideChannelAttack: Side Channel Attack, R package version 1.0-6 (2012)
22. Liaw, A., Wiener, M.: Classification and regression by randomforest. R. News 2(3), 18–22 (2002)
23. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M.: Cluster Analysis Basics and Extensions (2005)
24. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Investigations of Power Analysis Attacks on Smartcards. In: Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology, Berkeley, CA, USA, p. 17 (1999)
25. Oren, Y., Renauld, M., Standaert, F.-X., Wool, A.: Algebraic side-channel attacks beyond the hamming weight leakage model. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 140–154. Springer, Heidelberg (2012)
26. Oswald, E., Mangard, S.: Template Attacks on Masking—Resistance Is Futile. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 243–256. Springer, Heidelberg (2006)
27. Renauld, M., Standaert, F.-X.: Algebraic side-channel attacks. In: Bao, F., Yung, M., Lin, D., Jing, J. (eds.) Inscrypt 2009. LNCS, vol. 6151, pp. 393–410. Springer, Heidelberg (2010)

28. Sergios, T., Konstantinos, K.: Pattern Recognition, 3rd edn. Academic Press, Inc., Orlando (2006)
29. Singh, S.S., Chauhan, N.C.: K-means v/s k-medoids: A comparative study. In: National Conference on Recent Trends in Engineering and Technology (2011)
30. Standaert, F.-X., Peeters, E., Archambeau, C., Quisquater, J.-J.: Towards Security Limits in Side-Channel Attacks. IACR Cryptology ePrint Archive 2007, 222 (2007)
31. Velmurugan, T., Santhanam, T.: Computational complexity between k-means and k-medoids clustering algorithms for normal and uniform distributions of data points. Journal of Computer Science 6(3), 363–368 (2010)
32. Wang, S., Le, T.-H., Berthier, M.: When CPA and MIA go hand in hand. In: Center for Advanced Security Research Darmstadt, pp. 82–98 (2011)
33. Arduino website, `http://www.arduino.cc` (consulted December 8, 2011)
34. Zhu, X.: Semi-Supervised Learning Literature Survey Contents. SciencesNew York 10(1530), 10 (2008)

# A Appendix

## A.1 Algorithm

**Algorithm 1.** SSTA Algorithm: pseudo-code

**Require:** $attackedTrace, tracesUnknownKeys, tracesKnownKeys, attackersKeys$
**Ensure:** $byteHW$

1: $manipInst = byteManipulationInstant(tracesKnownKeys, attackersKeys)$ {correlation}
2: $SetOfClusters\ clusters = emptySet$
3: $traces = [tracesUnknownKeys, tracesKnownKeys]$
4: **for all** $trace$ in $traces$ **do**
5: $putIntoCluster(trace, clusters, manipInst)$
6: **end for**
7: $HW = prediction(attackedTrace, clusters, manipInst)$

## A.2 Figures

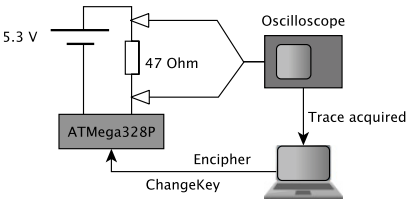



**Fig. 4.** Trace acquisition scheme

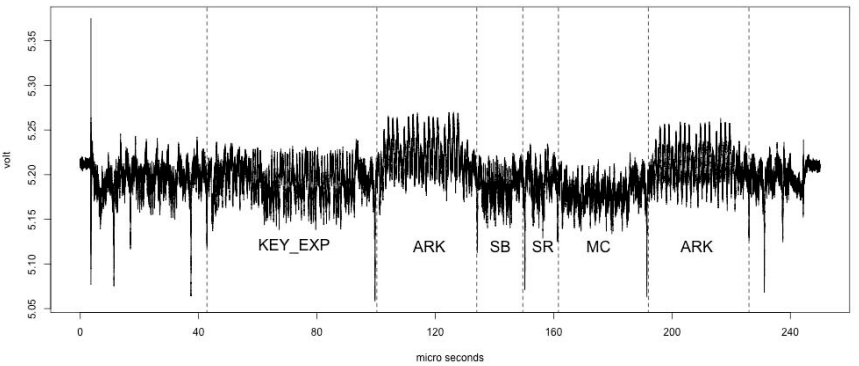



**Fig. 5.** Average of 128 traces. Mapping of AES steps on the trace: $KEY\_EXP$ – KeyExpansion, $ARK$ – AddRoundKey, $SB$ – SubBytes, $SR$ – ShiftRows, $MC$ – MixColumns.