Semi-Supervised Template Attack

Liran Lerman, Stephane Fernandes Medeiros, Nikita Veshchikov, Cedric Meuter, Gianluca Bontempi, and Olivier Markowitch

Département d'Informatique,
Université Libre de Bruxelles,
Boulevard du Triomphe,
1050 Brussels, Belgium.

Abstract. Side channel attacks take advantage of the information leakage in a cryptographic device. A template attack is a family of side channel attacks which is reputed to be extremely effective. This kind of attacks supposes that the attacker can fully control a cryptographic device before attacking a similar one. In this paper, we propose a method based on a semi-supervised learning strategy to relax this assumption. The effectiveness of our proposal is confirmed by software simulations as well as by experiments on a 8-bit microcontroller.

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

1 Introduction

Side Channel Attacks (SCA) take advantage of the fact that instantaneous power consumption [10], execution time [9] or/and electromagnetic emanations leaks [4] of a cryptographic device depending 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 of the techniques proposed for power analysis attacks along the years has been characterized by an increase in the complexity of the statistical analysis.

Simple Power Analysis (SPA) [10] was the first proposed approach to realize 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 detect in the power consumption a pattern linked to information about the executed operations. For example, Hollestelle et al. [7] showed that such attack against RSA implemented with a square and multiply algorithm allows the recovery of the key.

Differential Power Analysis (DPA) [10] uses more advanced statistical analysis than SPA by modeling theoretical power consumption for each key. The likelihood of the observed power consumption for each model is then used to predict the key.

Template Attacks (TA) [2] make an additional step by estimating the conditional probability of the trace for each key (profiling step). It extracts all available

information from each trace and can be considered as the strongest form of side channel attack [2]. This kind of attacks is feasible if the attacker can have access to the values of the keys during the profiling step. This paper intends to make one further step by relaxing this hypothesis. More precisely the attacker needs to know only two keys and their related power consumption.

We have analyzed the pertinence of our attack with simulations as well as with real data experiments. 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 is the number of bits of the key. For each key let's observe N times over a time interval of length n the power consumption of such device and lets denote by *trace* the series of observations $T_{(i)}^{(k)} = \left\{ T_{(i)(t)}^{(k)} \in \Re \mid t \in [1; n] \right\}$, $i = 1, \ldots, N$ associated to the k^{th} key. The state-of-the-art TA modelizes the stochastic dependency between the key and a trace by means of a multivariate normal conditional density:

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

where $\mu_k \in \Re^n$ and $\Sigma_k \in \Re^{n \times n}$, $k = 1, \ldots, K$, are respectively the expected value and the covariance of the n variate trace 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 μ_k and the covariance Σ_k by:

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

and:

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

Once a new trace T is observed on the attacked device, TA returns the key which maximizes the likelihood:

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

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.

Lets consider the power consumption of the device at time t depending on an internal value $f_k(x)$ where $x \in \chi$ is a (part of) plaintext and $k = 1, \ldots, K$. In other words, we have that $T^{(k)}_{(i)(t)} = L(f_k(x)) + \epsilon$ where L is the data-dependent device leakage and ϵ 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 [3, 13, 14, 16, 17]. 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 $\lambda \in \Re$ and $\xi \in \Re$).

3 The SSTA approach

As seen previously, template attacks are performed in two steps: a profiling step and an attacking step. The state-of-the-art approach of TA assumes that during the profiling step the attacker can fully control (change 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 (at least) two different chosen keys but he can still collect traces from the device with unknown keys.

This section discusses Semi-Supervised Template Attack (SSTA) approach and present related works.

3.1 Context and hypothesis

We assume that the attacker does not fully control the device and does not know the values of all keys. We suppose that the device is such that each user is linked to a particular key (e.g. a digipass or a bank card). In other words the attacker cannot change the key but he knows that the same fixed key is used when he manipulates the device. The attacker can collect two sets of traces linked to two different keys O_α and O_β. The Hamming weights of O_α and O_β are different and their values are not necessarily known by the attacker.

The attacker can collect a set of traces (linked to different unknown keys) measured when the device executes cryptographic operations. He can collect these traces while the device is used by different users having different keys.

The last hypothesis is that the power consumption is dependent on the Hamming weight of the manipulated data. We focused on 8-bit architectures during software and actual experiments, therefore we attacked one byte at a time in order to find the entire key. However this approach can be generalized to n-bit architectures (e.g. $n = 32$); in this case the attacker can focus on at most n bits at a time.

3.2 Overview of the SSTA

In the machine learning domain, supervised learning is a model which infers a function from traces and their respective keys. 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 [20].

In SSTA the attacker collects a set of traces $T_{(i)}^{(k)}$. For the sake of simplicity we restrict to consider attacks on a single byte of the key. Suppose that the cryptographic device manipulates the b^{th} byte of the key k (namely k_b) at time t (i.e. $T_{(i)(t)}^{(k)} = \lambda \mathrm{HW}(k_b) + \xi + \epsilon$). 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 closer than if (sub)keys had different Hamming weights. Therefore the attacker can regroup traces that have the same power consumption when the cryptographic device manipulated the b^{th} byte of the key by using the advantages offered by machine learning techniques [6]. In other words, clustering techniques allow us to find all traces that have the same Hamming weight in order to group them into clusters.

Next the attacker has to find the value of the Hamming weight of each cluster. For this, we have to focus on the density distribution of the Hamming weights of a byte as shown in Table 1 where p_i denotes the probability to have a trace linked to a key which has a Hamming weight of i. As we can see, there is approximately $\frac{1}{256}$ 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.

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:

$$\hat{h} = \arg\min_{h} d_t(T, T^{(h)}) \tag{5}$$

where d_t is a distance measurement between two traces on the instant t (where k_b is manipulated) and $T^{(h)}$ plays the role of prototype of the set of neighboring traces linked to keys which have the b^{th} byte of Hamming weight h.

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 the attack against another execution. In other words, the model can improve its success rate at each execution of the attacked device.

Once the Hamming weight is known, the attacker has to 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 (enumerate all 256 possible values), see Table 1.

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 try the closest neighbors (i.e. $\hat{h}+1$ and $\hat{h}-1$).

Hamming weight	Number of different values	Probability (p_i)
0	1	$\frac{1}{256}$
1	8	$\frac{8}{256}$
2	28	$\frac{28}{256}$
3	56	$\frac{56}{256}$
4	70	$\frac{70}{256}$
5	56	$\frac{56}{256}$
6	28	$\frac{28}{256}$
7	8	$\frac{8}{256}$
8	1	$\frac{1}{256}$

Table 1. Number of possible values of one byte depending on its Hamming weight. The probability to have a trace linked to a key which has a Hamming weight of i is denoted by p_i.

Up to now, we supposed that we know the instant t when the device manipulates the attacked byte. In order to find this instant t, we suppose that the attacker has two keys (e.g. his key and his wife's key) of different 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. In our experiment we used mutual information in order to find this instant.

Finally, Algorithm 1 gives a pseudo-code of SSTA on a single byte of the key.

3.3 Overview of related works

As seen previously, SSTA is related to TA but also to other types of attacks.

Batina et al. [1] presented the Differential Cluster Analysis (DCA) against a cryptographic device using an unknown fixed subkey. This technique uses cluster analysis to detect internal collisions in their 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 that have the same value. Finally it assesses the quality of the cluster separation thanks to cluster criterion such as the "Sum-Of-Squared-Error". The main difference with SSTA is that DCA must know the cryptographic algorithm which is implemented on the target device and the key cannot change during the attack.

Algorithm 1 SSTA ALGORITHM: PSEUDO-CODE

Require: $traces, attacked\_trace, attackersKey1, attackersKey2$
Ensure: $byteValue$

1: $traces1 = getSetOfTraces(attackersKey1)$
2: $traces2 = getSetOfTraces(attackersKey2)$
3: $manipInst = byteManipulationInstant(traces1, traces2)$
4: $SetOfClusters\ clusters = emptySet$
5: **for all** *trace* in *traces* **do**
6: $\quad putIntoCluster(trace, clusters, manipInst)$
7: **end for**
8: $HW = prediction(attacked\_trace, clusters, manipInst)$
9: $byteValue = recoverKey(HW)$ {enumeration by brute force}

Lerman et al. [11] and Hospodar et al. [8] discussed the role of machine learning in TA. They showed that a 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.

Dyrkolbotn et al. [3] 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 fully control a clone device in order to build their classification model.

3.4 Discussion of assumptions

In this section we will discuss three important issues: the probability of having two keys of different Hamming weights, the required number of traces to obtain at least one trace per Hamming weight and the number of tests to perform in order to recover the key.

Probability of having two keys of different HW
We suppose that the attacker has two (sub)keys with different Hamming weights of the attacked byte. If the attacker cannot choose the values of these keys, then the probability that he has such pair of keys when he takes two random keys is:

$$P(\mathrm{HW}(O_\alpha) \neq \mathrm{HW}(O_\beta)) = 1 - P(\mathrm{HW}(O_\alpha) = \mathrm{HW}(O_\beta)) = 1 - \sum_{i=0}^{8} p_i^2 = 0.80 \quad (6)$$

In other words, the probability that these keys have different Hamming weights is very high. So the attacker can easily find such pair of keys in order to find out when the key is manipulated.

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 his model. Our approach allows to estimate 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.

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. 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.

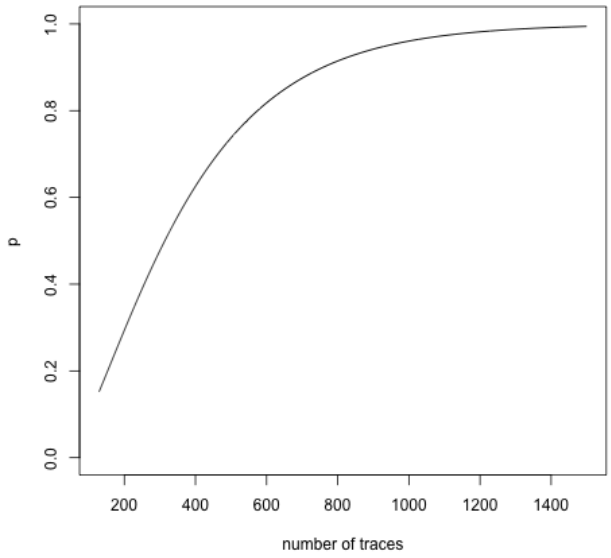


Fig. 1. Probability to have a least one trace per Hamming weight value depending on the number of collected traces.

Number of tests to perform in order to recover the key
Once the attacker finds the Hamming weight of the attacked byte he has to retrieve the exact value of this byte. The complexity to find this exact value depends on the value of the Hamming weight. We denote p_i the probability to have a trace linked to a key which has a Hamming weight of i, see Table 1.

In the case the Hamming weight of the byte is zero or eight, it takes no time to find its value ($k_b = 0$ or $k_b = 255$ respectively) while there are 70 values of a byte that have the Hamming weight of four. However the probability that the trace is linked to a Hamming weight of zero or eight is lower than the probability of having a Hamming weight of four.

In order to estimate the average number of attempts (named θ) needed to find the value of a byte of the key knowing its Hamming weight we can use the expected value:

$$\theta = \sum_{i=0}^{8} p_i \times G_i \tag{9}$$

The equation (9) can be rewritten as:

$$\begin{aligned} \theta = & \frac{1}{256} \times 1 + \frac{8}{256} \times 8 + \frac{28}{256} \times 28 + \frac{56}{256} \times 56 + \frac{70}{256} \times 70 \\ & + \frac{56}{256} \times 56 + \frac{28}{256} \times 28 + \frac{8}{256} \times 8 + \frac{1}{256} \times 1 \\ = & \ 50.27344 \end{aligned} \tag{10}$$

In other words, the average number of attempts needed to find the value of a byte knowing its Hamming weight is 51. In a general case where the key is i bytes length, brute force attack will have to test an average of 51^i values.

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 a real data experiment: we attacked the initial round and the first round of AES[1] implemented on a microcontroller.

4.1 Validation

In order to assess the quality of SSTA, we adopt a holdout validation technique. 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.

[1] AddRoundKey, SubBytes, ShiftRows, MixColumns, AddRoundKey.

We say that the attack is successful if it estimates the right Hamming weight of the key based on a trace measured on the target device. It is therefore depending on the target intermediate function, the form of the data-dependent device leakage L and the distribution of the noise ϵ. Following the idea of Whitnall et al. [19] we say that the attack is theoretically successful if it always succeeds with any trace measured on the target device and it is ideally successful in a noise-free scenario.

It is impossible to verify theoretically successful and ideally successful (an infinite number of traces is required for this verification). However, an estimation can be computed with a large number of holdout validations.

4.2 Software simulations

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

The estimation of the success rate[2] 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" [15][3] clustering technique was used. 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.

In other words, SSTA is ideally successful but not theoretically successful.

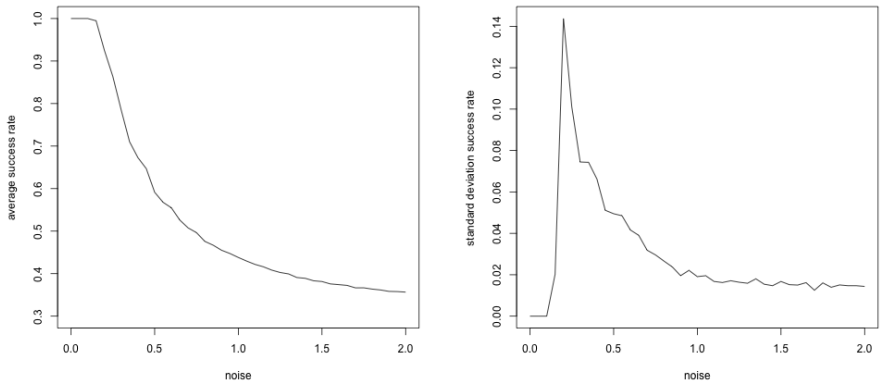
4.3 Real data experiment

After a simulation where each parameter can be controlled, we did a read data experiment on an 8-bit microcontroller ATmega328P. The microcontroller was programmed using an Arduino Uno board [18]. 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 $5.3V$ supply. It used an external $16MHz$ clock. We measured the power consumption by inserting a 47Ω resistor on the power pin V_{CC} of ATmega328P. For all acquisitions we used an Agilent Infinium 80000B Series oscilloscope $2GHz$ $40GSa/s$. See the acquisition scheme in Figure 3.

Two different signals (handled using interruptions) could be send 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 last value of the key.

[2] Which is the average of one hundred success rate obtained in the same conditions (i.e. with the same value of the standard deviation of the noise).

[3] Package "cluster" [12] available on CRAN.



(a) Average of success rates (b) Standard deviation of success rates

Fig. 2. The Figure (a) shows the average success rates depending on the value of the noise. Figure (b) shows the standard deviation of success rates depending on the noise. Both figures were generated using 100 experiments on a simulated device. "Partitioning Around Medoids" clustering technique was used.

The attack was realized 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[4], its value changed from 0 to 255.

For each value of the attacked byte we realized 10 acquisitions. Each acquisition is the average, realized on the oscilloscope during the acquisitions, of 128 single acquisitions using the same key and plaintext. So, for each key we have 10 traces each representing the average of 128 acquisitions. These values (10 traces and average of 128 acquisitions) were chosen with respect to logistical constraints.

The result of a measure (an average of 128 acquisitions) is a trace such as the one plotted in Figure 4. Traces were aligned using a trigger on the 'encipher' signal send to the device. In order to realize the attack we compressed the traces by using each fifth point.

The simple model based on the probability density function of Hamming weight (see Table 1) will always give the answer 4 for the Hamming weight of a byte, since it is the most probable value. Its success rate is $\frac{70}{256} = 27.34\%$.

We realized 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. The best point returned by the feature selection model was located at the

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

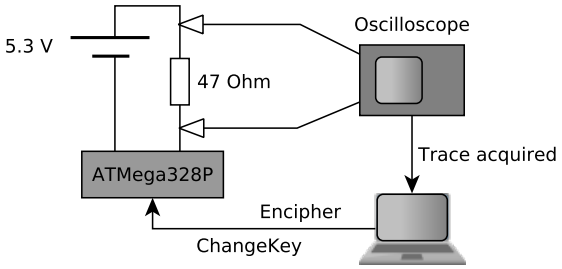


Fig. 3. Trace acquisition scheme.

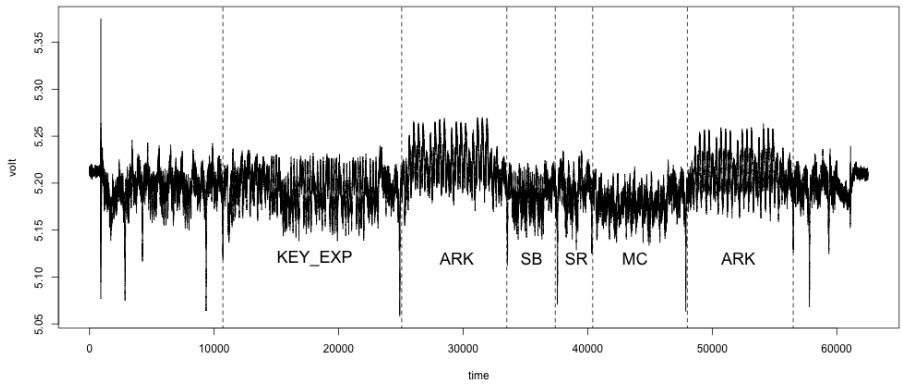


Fig. 4. Average of 128 traces. Mapping of AES steps on the trace: KEY_EXP – KeyExpansion, ARK – AddRoundKey, SB – SubBytes, SR – ShiftRows, MC – MixColumns.

end of the initial AddRoundKey, which is the moment in time where the attacked byte (i.e. the 13^{th}) is being manipulated. In real data experiment we also used the "Partitioning Around Medoids" clustering technique. The result of our attack is shown in Table 2. The success rate is higher than the success rate of the simple model.

Number of traces per key in the learning set	Number of traces per key in the validation set	Number of traces in each average-trace	Success rate
8	2	128	61.5%

Table 2. Success rate depending on the number of traces per key.

5 Conclusions

We presented and assessed a 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 only to know two different keys of different Hamming weights.

We implemented our technique using simulated and real data traces. In both cases we show that we can find the Hamming weight of a byte of the key.

We analyzed the limits of our attack. Firstly we estimated the probability that the attacker has two keys with two different Hamming weights. Secondly by computing the number of traces which have to be collected in order to complete a real data attack. Finally by computing the average number of attempts needed to find the value of a byte of the key knowing its Hamming weight.

We suppose that the keys are uniformly distributed which is not necessarily the case in asymmetric algorithms.

Future works will focus on the generalization of these preliminary results: firstly by considering larger parts of the key, secondly by assessing the impact of the plaintext on the prediction accuracy, thirdly by considering all rounds of AES as well as other algorithms and finally by varying the cryptographic device and its architecture.

Interesting future research perspectives are also to consider the adaptation of clustering multivariate technique [5] as well as specific feature selection techniques for the dimensionality reduction of traces.

Eventually, protected implementations against this kind of analysis will be investigated.

References

1. Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Differential Cluster Analysis. In Christophe Clavier and Kris Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2009.
2. Suresh Chari, Josyula Rao, and Pankaj Rohatgi. Template Attacks. In Burton Kaliski, etin Ko, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 51–62. Springer Berlin / Heidelberg, 2003.
3. Geir Olav Dyrkolbotn and Einar Snekkenes. Modified Template Attack: Detecting Address Bus Signals of Equal Hamming Weight. In NTNU Stig F. Mjlnes, editor, *The 2nd Norwegian Information Security Conference (NISK2009)*, pages 43–56. Tapir Akademisk Forlag, Nov 2009.
4. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.
5. Wolfgang Hardle and Zdenek Hlavka. *Multivariate Statistics: Exercises and Solutions*. Springer Publishing Company, Incorporated, 1st edition, 2007.
6. Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer, 2nd ed. 2009. corr. 3rd printing 5th printing. edition, September 2009.
7. Gijs Hollestelle, Wouter Burgers, and Jerry I. Den Hartog. Power Analysis on Smartcard Algorithms Using Simulation, 2004. Imported from DIES.
8. Gabriel Hospodar, Elke De Mulder, Benedikt Gierlichs, Joos Vandewalle, and Ingrid Verbauwhede. *Least Squares Support Vector Machines for Side-Channel Analysis*, page 99–104. Center for Advanced Security Research Darmstadt, 2011.
9. Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
10. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
11. Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. *Side Channel Attack: an Approach Based on Machine Learning*, page 29–41. Center for Advanced Security Research Darmstadt, 2011.
12. Martin Maechler, Peter Rousseeuw, Anja Struyf, and Mia Hubert. Cluster Analysis Basics and Extensions. 2005.
13. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, pages 17–17, Berkeley, CA, USA, 1999. USENIX Association.
14. Mathieu Renauld and Franois-Xavier Standaert. Algebraic Side-Channel Attacks. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Information Security and Cryptology (INSCRYPT) 2009*, volume 6151 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 12 2009.
15. Theodoridis Sergios and Koutroumbas Konstantinos. *Pattern Recognition, Third Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

16. François-Xavier Standaert, Eric Peeters, Cédric Archambeau, and Jean-Jacques Quisquater. Towards Security Limits in Side-Channel Attacks. *IACR Cryptology ePrint Archive*, 2007:222, 2007.
17. Shiqian WANG, Thanh-Ha Le, and Mael Berthier. *When CPA and MIA go hand in hand*, page 82–98. Center for Advanced Security Research Darmstadt, 2011.
18. Arduino website. http://www.arduino.cc, consulted 8 December 2011.
19. Carolyn Whitnall and Elisabeth Oswald. A Comprehensive Evaluation of Mutual Information Analysis Using a Fair Evaluation Framework. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference*, volume 6841, page 311, 2011.
20. Xiaojin Zhu. Semi-Supervised Learning Literature Survey Contents. *SciencesNew York*, 10(1530):10, 2008.