# Profiled Model Based Power Simulator for Side Channel Evaluation

Nicolas Debande[1,2], Maël Berthier[1], Yves Bocktaels[1] and Thanh-Ha Le[1]

[1] Morpho
18 chaussée Jules César, 95520 Osny, France
firstname.familyname@morpho.com
[2] TELECOM ParisTech
46 rue Barrault F-75634 Paris Cedex 13, France
familyname@enst.fr

**Abstract**

An embedded cryptographic device performs operation on sensitive data and as such, is vulnerable to side-channel attacks. This forces smart-card manufacturers to carefully consider development of security mechanisms. To accelerate this procedure, the use of power and electromagnetic simulator can be relevant and saves non negligible time. Based on a high level simulator, we propose to use profiled abstract models to gain accuracy on the simulated traces. These abstract models are obtained by profiling some parts of the target device which is physically available by the evaluator.

**keyword:** Smart Cards, Power Simulation, Side Channel Analysis, Security Evaluation.

## 1 Introduction

The amount of embedded devices have considerably increased these last years. Consequently, there is a real need for protecting information security from malicious outsider. Nowadays, there exists a wide variety of techniques to extract secret information from a device. In the context of embedded system, Side Channel Analysis (SCA) exploits information leakages generated by the hardware implementation of the system [6] [7]. The leakage can be

observed from the power consumption, the electromagnetic emanation or even in the time execution of the device. Also, this context allows the attacker to inject faults on the chip, which perturbs the correct behaviour of the device. The efficiency of these attacks combined with the amount of embedded device users implies the interest of the smart-card designers, smart-card manufacturers, certification centres and research laboratories to this subject. Indeed, a better knowledge of these physical attacks leads to a better evaluation of the vulnerability or robustness of a device against Side Channel attacks. Thus, a lot of countermeasures have been worked out in order to secure embedded devices from these attacks. Companies in the embedded systems sector are concerned by these security questions, and have to guarantee information security on their products which ensures information confidentiality or impossibility of reproduction, *etc.*

A way to evaluate the security of an embedded device is to attack it. Indeed, the more efficient the attack, the less secured the device. Besides having a good knowledge about SCA, to perform an attack requires an amount of data curves, acquired from a physical device or from a simulator.

In the first case, the attacked device has to be in a post-conception step, *i.e.* the device is ready to be sold (except the security validation), in order to have relevant and realistic results. Indeed, the companies want to test a device as near as possible than the final product. This is a non-negligible constraint. When an software information leakage is found, developers include suitable countermeasures and load the new code on the device. If the leakage comes from a hardware weakness, then the smart-card manufacturers have to patch the device before a security test is performed again. The use of a power consumption (or electromagnetic emanation) simulator is motivated by a gain of speed in the security evaluation. However, this method is naturally less realistic than with real physical device.

To generate power consumption of a device while executing an operation (*e.g.* an encryption) requires the knowledge of the hardware design. Indeed, power consumption is deduced from the number of transistors used by the device at each instant.

However in practice, many companies needs to verify the security of their products while they do not even know precisely the hardware design of their devices. Then in this context, the power consumption is particularly difficult to simulate with the current tools. The high level simulator introduced in this paper aims at creating simulated curves in this context, *i.e.* without the knowledge of the hardware design. As the substitute for the design, the simulator characterizes leakages thanks to side channel observations. After the profiling phase, models are used to generate new traces.

The simulator allows to evaluate the device robustness for various leakage models. Also, it can be used to deduce, from a given model, a new model according to a code revising or a countermeasure adding. Additionally, there is many approaches for the combination of the high level simulator with the low level simulator. The high level simulator can take as inputs the curves simulated by the low level simulator, in order to characterize a leakage model. This model will be used to speed up the next simulation processing. Another way to combine the two tools is to simulate a part of the device with one simulator and another part of the device with the other. The simulated curves will be construct according to some countermeasures, selected by the user (noise, temporal warping, *etc.* ). The simulator aims at approaching as better as possible the effect of a given countermeasure, in order to better predict its consequences. Also, the high level simulation is light and fast, being suitable to generate training sets for academic purpose, especially when no acquisition equipment is available.

In this paper, we considered attacks which exploit power consumption. The introduced simulator aims at being a high level (or abstract level) platform for evaluation of SCA vulnerability for embedded device.

We recall some existing simulators in Sec. 2. Then the simulator is introduced in Sec. 3. The profiling phase and the pattern reconstruction are described in Sec. 4 and Sec. 5 respectively. At last, some experimental results are described in Sec. 6.

# 2 Previous Work

Sec. 2.1 shows several power simulators from the state of the art. Sec. 2.2 recalls some mathematical backgrounds about the stochastic methods.

## 2.1 Simulation

In this section are showed some low level power consumption simulator: Nanosim, PINPAS, SCARD, MP-ARM and SystemC. Embedded device simulators can be classified either "analog", based on differential equations solvers (*e.g.* SPICE, ADS) or "digital", based on logical events propagation (*e.g.* Ncsim, PrimePower, Modelsim). Some simulators are both analogue and digital, as NanoSim. Analog simulators are low level but works with only a small part of the circuit. Digital simulator are well-studied for big circuits. However, these simulators do not have enough accuracy to extract relevant information.

NanoSim is a transistor-level power simulator developed for CMOS and BiCMOS circuit designs [9]. Transistor-level is the lowest possible level. NanoSim contains also some analysis tools. However, the main goal of Nanosim is to help designers for lower power but cannot works with high level models so, it is not suitable for side channel analysis.

PINPAS (Program INferred Power Analysis in Software), developed by the Eindhoven University of Technology and TNO-TPD in 2004, is a tool which permit to generate power curves without physical device (see [5]). The algorithm can also be chosen (DES, IDEA, etc) and even different hardware implementation. However, PINPAS needs to know the hardware design and the assembler code to work.

SCARD (Side Channel Analysis Resistant Design flow) is a tool which aims to simulate side channel analysis effects, developed in 2005 [1]. This tool proposes to evaluate the efficiency of a given countermeasure by attacking the virtual device with the generated curves and with side channel analysis. The distinctive characteristic of SCARD is that it already includes an approach about high level simulation.

MP-ARM is a simulation platform for MP-SoC (Multi-Processor Systems-on-Chip) based on SystemC (see [3]). This tool aims to ease the design step of a MP-SoC. Its includes processor models, memory models and some other pratical tools.

SystemC is used to describe material at a high level [2]. This permits to simulate systems with a very high speed. However, it is only suitable for functionality check and not for secret information leakages.

More recently, Thuillet *et al.* [10] introduced a high level simulator which allows to construct traces without the knowledge of hardware design. Based on a software code, the simulator compute all states for all registers. Then, power consumption traces are deduced from these states using abstract models as the Hamming distance.

### 2.2 Stochastic Models

Stochastic model has been introduced by Schindler *et al.* in [8]. It allows to profile the power consumption or electromagnetic emanation behaviour of a device. Let's assume that the device's activity is expressed as:

$$\mathrm{C}(t) = \beta_0(t) + \sum_{i=1}^{n} \beta_i(t) f_i, \tag{1}$$

where $f_i$ are $n$ chosen functions and $\mathrm{C}(t)$ is an estimation of the power consumption or electromagnetic emanation. Let's assume that we want to

model a register $r$, during the storage process, then $f_i$ could be a function of $\mathrm{S}_0$, the initial state of $r$ and $\mathrm{S}_1$, the final state of $r$. In this case, profiling step aims at computing weighting curves $\beta_i$ depending on all bits of a $N$-bits register $r$ and an additional weighting curve $\beta_0$ which models the rest of the circuit. The chosen function can be:

- $n = 1$, Hamming weight of $\mathrm{S}_0 \oplus \mathrm{S}_1$
- $n = N$, $f_i$ is the $i$th bit of $\mathrm{S}_0 \oplus \mathrm{S}_1$
- ...

Note: The choice of $f_i$ can be different between profiling and estimating. Indeed, a countermeasure can be simulate by a suitable choice of $f_i$ functions. For example, we can simulate a balanced power consumption or electromagnetic emanation by chosen $f_i$ such as:

$$f_i = \begin{cases} 1 & \text{if } \mathrm{HW}(\mathrm{S}_0 \oplus \mathrm{S}_1)_i = 1 \\ 0.9 & \text{if } \mathrm{HW}(\mathrm{S}_0 \oplus \mathrm{S}_1)_i = 0 \end{cases}$$

where $\mathrm{HW}(\mathrm{S}_0 \oplus \mathrm{S}_1)_i$ is $i$th bit of the Hamming weight of $\mathrm{S}_0 \oplus \mathrm{S}_1$.

When averaged activity has been computed, the variance of the noise is characterized as follows:

$$\mathrm{v}(t) = \mathrm{Var}(C(t) - \tilde{C}(t)), \tag{2}$$

where $C(t)$ and $\tilde{C}(t)$ are real and reconstructed traces respectively.

# 3 Description of the simulator

## 3.1 General Behaviour

The global behaviour of the simulator is described in Fig. 1. For the characterization, we used the profiling phase introduced in [8]. Unlike template-based profiling [4], stochastic models allows to easily reconstruct traces, thanks to the linear regression. The characterization step is not dependent of cryptosystems and implementations, when profiling registers behaviour. Indeed, it takes on parameter only two successive states of a given register (for instance, $\mathrm{S}_0$ is a part of the plain text and $\mathrm{S}_1$ the corresponding part of the intermediate value at the first round). This step has to be repeated until all registers is modelled (and possibly on each round). As the original profiling method characterizes only one byte, we have to repeat this step as many times than the number of registers. Then, these models are merged in the

final model. If the profiling phase is applied on a single round, the profiled $\beta_i$ are used on all the other rounds. In Equ. 1, $\beta_0$ is useless and never saved in our case. However, when computing the noise variance, we compute the averaged activity $A$ which will be used to simulate new traces. Note that all linear regression methods, and more generally all profiling methods, can be used for the characterization step, as long as the traces reconstruction is available with the abstract models.

The "'Algorithmic Behaviour"' part aims at providing intermediate states necessary for using previous profiled model. For instance, for using an abstract model which characterize the Hamming weight of a variable, the simulator has to know all values of this variable. This part is similar than the work provided by Thuillet *et al.* [10]. Additionally, the evaluator can add functional information about leakages on other part of the device, as the noise generated by the clock or the hardware design of a specific countermeasure. This allows to combine information from different sources to construct a single trace.

Fig. 2 shows an example of a simulated power consumption trace.

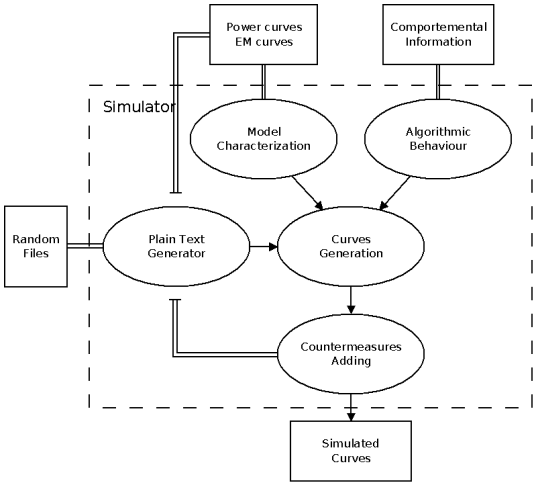


Figure 1: Conceptual model of the simulator

## 3.2 Objectives

The first goal of a simulator is to provide information something which does not exist yet. In our case, we want to learn about secret leakage of an embedded device, but we a need real device as an input for the simulator.

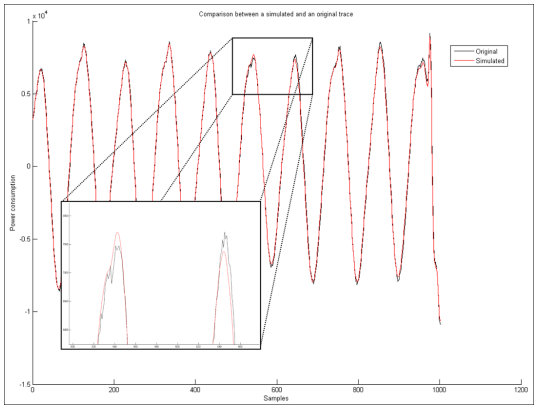


Figure 2: Comparison between a real and a simulated power trace

This implies we cannot use this kind of simulator in pre-conception. This section clarifies the objectives of the high level simulator.

In the hardware context, we have a post-conception device and we characterize a cryptoprocessor (with an unknown design). If some leakages are found, one can wonder which hardware modifications could improve the device's robustness. It seems complex to predict any modifications directly on the characterized cryptoprocessor (*e.g.* to upgrade from unprotected to masking implementation). However, models can be used to simulate modules adding such as noise or delay generator. The question is if stochastic models are useful to give good predictions on hypothetical hardware countermeasure adding. Hardware modules activity (as cryptoprocessors) depends strongly of the design and the used technology. Thus, even a perfect characterization of a hardware module could not lead to a realistic prediction.

An other possibility is to analyse how a micro-controller leaks in SCA context. Indeed, many works show that SCA attacks are also very efficient on software codes. Furthermore, reverse engineering techniques are actively studied. From a known code and a real device, the high level simulator can characterize each component of the microcontroller. This allows to developers to validate a modified code without load it on the target device. The gain in time is non negligible, especially if the device does not have flash memory. Note that, unlike the hardware case, there no more limitations about which countermeasures can be evaluated in the software case. Since each CPU component is modelled, any code sequences can be simulated without difficulty.

# 4 Profiling Phase

This section describes how the profiling phase is computed. The profiling step allows to compute averaged power activity according to the chosen models. Hardware and software case are dealt within two different sections, Sec. 4.1 and Sec. 4.2 respectively. Then for both cases, the global averaged activity $A(t)$ is computed as follows:

$$A(t) = E[\mathrm{C}(t) - P(t)], \tag{3}$$

where $\mathrm{C}(t)$ is the profiled traces and $P(t)$ is the reconstructed profiled signal. The noise variance is modelled by computing:

$$v(t) = \mathrm{Var}[\mathrm{C}(t) - P(t)], \tag{4}$$

## 4.1 The Hardware Case: cryptoprocessor

A first approach is to model a cryptoprocessor is to target state registers using an arbitrary leakage model. By using first-order stochastic model, we obtain as many weighting curves $\beta_i$ as bit register. Each $\beta_i$ represents the main activity of one bit, in function of the time. Then in Equ. 3, $A$ is the averaged activity of the device. To improve the accuracy of the model, one can repeat the process. Indeed, when computing the $\beta_i$ for the first time, raw observations $\mathrm{C}(t)$ are used. With a second process, the $\beta_i$ are computed again, but using modified observations $g(\mathrm{C}(t))$ as follows:

$$g(\mathrm{C}(t)) = \mathrm{C}(t) - \sum_j \beta_j^1(t) f_j = \beta_0(t) + \sum_{i=1}^{n} \beta_i^2(t) f_i, \tag{5}$$

where the $\beta^k(t)$ are the models computed at the $k$th iteration.

A natural assumption is to consider that the state register's model are invariant from a round to another. Fig. 4 shows averaged bit register models of an hardware AES on the five last rounds. Clearly, the global activity is constant modulo round. However, by analyzing models more precisely, we noticed that the activity strongly differs when regarding on a single bit. Except state registers, only key values are varying during the encryption. That's why we think that the variance of bit activity could be caused by the key. Fig 3 shows clearly the influence of a key bit on the corresponding bit activity. This shows that we have to also consider the different round keys to construct our final model.

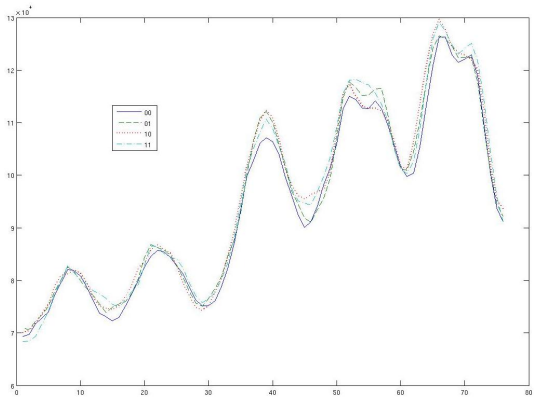


Figure 3: Influence of key bits on the model

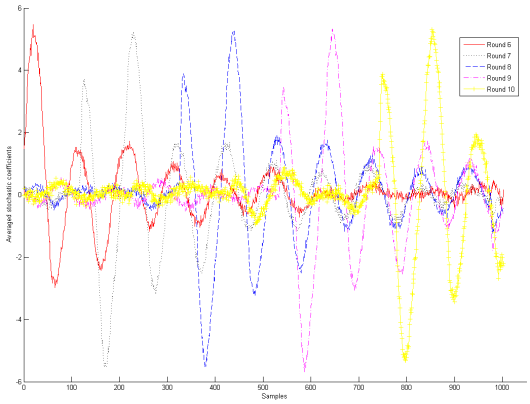


Figure 4: Stochastic profiling applied on several rounds

### 4.2 The Software Case: microcontroller

In the software case, a first possibility is to study each CPU instruction individually. For instance, registers which store opcodes and operands can be characterized. In the other hand, it may be more relevant to analyze instruction sequences, losing accuracy on each instructions but exploiting additional information. As we want to the most independent as possible than the software code (and the sequences), we choose to rule out the last proposal (more appropriate in reverse engineering context). More precisely, all microcontroller's registers (instruction registers, program counter, control register, *etc.* ) can be characterized using the same methodology than for the hardware case, *i.e.* each register independently. For the profiling

step, we tried three ways to make partitions. First we focused only on the target register (without taking into account the other registers), as usually. However, this method cannot lead to a relevant characterization of the register. The reason is that partitions are strongly affected by other activity, especially caused by the opcode value. Second, we focused on the target register and the opcode register: we compute an averaged trace for each opcode, then subtract the corresponding pattern to data before partitioning. Again, this failed. At last, by only focusing on the target register and according to a single opcode, we had relevant register's model. Fig. 5 shows different bit activity.

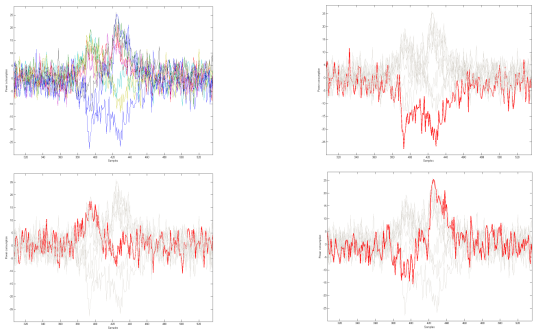

Figure 5: Different shapes for a bit flipping activity

## 5 Pattern Reconstruction

Curves are generated using all the models as follows:

$$\tilde{\mathrm{C}} = \tilde{\mathrm{C}}_0||\tilde{\mathrm{C}}_1||...||\tilde{\mathrm{C}}_p, \tag{6}$$

where $\tilde{\mathrm{C}}_i$ is $i$th cycle of the encryption. Each $\tilde{\mathrm{C}}_i$ are in function of some parameters, varying on the time (as intermediate values, operands, *etc.* ). They are defined as:

$$\tilde{\mathrm{C}}_i(t) = A_i(t) + \sum_{j=1}^{n} \beta_j(t) f_j. \tag{7}$$

For instance, an hardware AES can be modelled with:

$$\tilde{\mathrm{C}}_i(t) = A_i(t) + \sum_{j=1}^{128} \beta_{1,j}(t) f_{1,j}(\mathrm{S0}_i, \mathrm{S1}_i) + \sum_{j=1}^{128} \beta_{2,j}(t) f_{2,j}(\mathrm{S1}_i) \tag{8}$$

with

$$f_{1,j}(\mathrm{S0}_i, \mathrm{S1}_i) = \mathrm{S0}_i(j) \oplus \mathrm{S1}_i(j) \text{ and}$$
$$f_{2,j}(\mathrm{S1}_i) = \mathrm{S1}_i(j),$$

where $\mathrm{S0}_i(j)$ is the $j$th bit of the state S0.

# 6 Experimental Results

This section provide some experiments using the high level simulator. Sec. 6.1 discuss about the number of traces needed to obtain a model sufficiently accurate. Then, we compare simulated traces with acquired ones is Sec. 6.3.

## 6.1 Quality of the obtained stochastic models

In order to evaluate the quality of our simulation method, we compare the efficiency of standard SCA attacks on simulated sets and experimental acquired sets. As a metric, we used the success rate of the attacks. Fig. 6 shows success rates of CPA on an acquired set and several simulated sets, according to the number of traces used to construct the stochastic models. As expected, success rate is very sensitive to the size of used set. Furthermore, we can see that, comparing to the real traces, it is easier to attack with CPA on simulated traces when the model is sufficiently accurate (*i.e.* when computed with a sufficient number of traces).

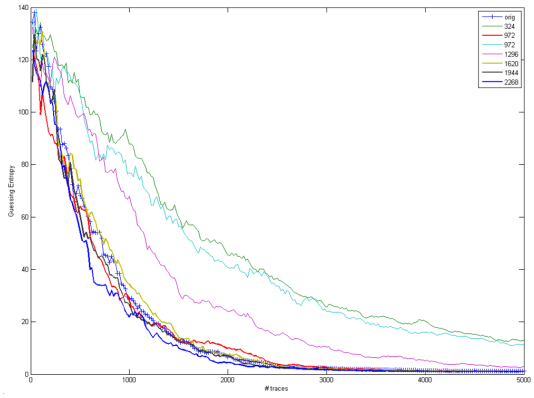


Figure 6: CPA success rates according to different size of profiled sets

### 6.2 Effect of several iterations during the profiling step

As mentionned in Sec. 4.1, the evaluator can repeat the profiling process using Equ. 5 in order to obtain more accurate models. In this section, we compare profiled models using one and two iterations. We simulate an amount of 10000 traces by fixing manually the $\beta_i$ values and the standard deviation of Gaussian noise. As metric, we compute the square distance between the fixed $\beta_i$ and profiled ones. Tab. 1 shows the results for five different noise level. Clearly, we see that the efficiency of the second process highly depends on the standard deviation of the noise. With a low level, the abstract models can significantly be improved with an extra iteration, while it practically has no effect when the noise is too high.

Table 1: Square distances between real and profiled models

| | $\sigma = 1$ | $\sigma = 2$ | $\sigma = 5$ | $\sigma = 10$ | $\sigma = 20$ |
|---|---|---|---|---|---|
| **#iter = 1** | 0.6944 | 0.6826 | 1.3254 | 5.2614 | 24.903 |
| **#iter = 2** | 0.0665 | 0.2556 | 1.1010 | 4.9972 | 23.749 |
| **Ratio** | 10.4 | 2.67 | 1.20 | 1.05 | 1.05 |

### 6.3 Comparison with acquired traces

The main goal of the simulator is to be realistic as far as possible. Using guessing entropy metric, we can evaluate the realistic property of a simulated set of traces. We used five sets of traces to make the comparison. The first set is made up of acquired traces of a hardware AES on SASEBO-B. Using this real set, we made profiling according to Hamming weigth and Hamming distance model. This leads to three simulated set of traces: using the Hamming weight model (HW), the Hamming distance model (HD) and the combination of both models. At last, we simulate three sets using a very high level model (without profiling phase). Actually the last sets corresponds to stochastic models with all coefficient equal to one. Fig. 7 and Fig. 8 shows the guessing entropy of a CPA on all these sets. In Fig. 7, we can see that, simulated attacks using profiled model are quite close to the real attack. Furthermore, using both profiled models (HD and HW) leads to a better prediction of the attack efficiency. In Fig. 8, guessing entropy curves are quite different from the simulated to the real one. We can explain this by assuming that there is some leakages we did not consider and which are dependent on the Hamming weight of the manipulated data. Thus, by reconstructing the traces, the simulator increased the signal to noise ratio

and so favour the CPA attack.

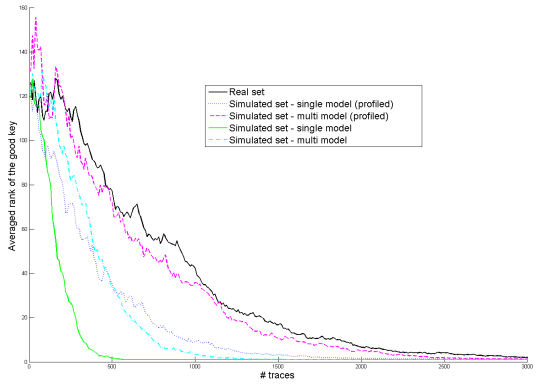


Figure 7: CPA guessing entropy using Hamming Distance model

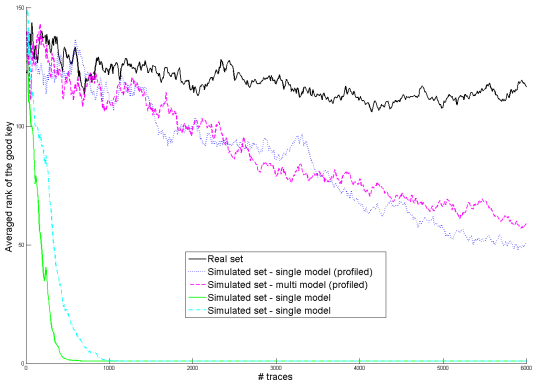


Figure 8: CPA guessing entropy using Hamming Weight model

# 7 conclusion

In this paper, we show how to make a high level simulator more realistic using acquired traces as input for profiling. The profiling phase brings to the simulator abstract models which are then used to construct new traces. The advantages of this approach is the possibility of combining these profiled models with new assumption on the device as some hardware countermeasures or a different code sequence. We show that using profiled models leads to a more realistic simulation compared to a very high level simulator.

## References


[1] M. J. Aigner, S. Mangard, F. Menichelli, R. Menicocci, M. Olivieri, T. Popp, G. Scotti, and A. Trifiletti. Side channel analysis resistant design flow. In *ISCAS*. IEEE, 2006.

[2] D. Automation. Ieee standard systemc language reference manual. *IEEE Computer Society*, 2002(March):1666–2005, 2006.

[3] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. Mparm: Exploring the multi-processor soc design space with systemc. *VLSI Signal Processing*, 41(2):169–182, 2005.

[4] S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[5] J. den Hartog, J. Verschuren, E. P. de Vink, J. de Vos, and W. Wiersma. Pinpas: A tool for power analysis of smartcards. In D. Gritzalis, S. D. C. di Vimercati, P. Samarati, and S. K. Katsikas, editors, *SEC*, volume 250 of *IFIP Conference Proceedings*, pages 453–457. Kluwer, 2003.

[6] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[7] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[8] W. Schindler, K. Lemke, and C. Paar. A stochastic model for differential side channel cryptanalysis. In *CHES*, pages 30–46, 2005.

[9] B. Sukhwani, U. Padmanabhan, and J. M. Wang. Nano-sim: A step wise equivalent conductance based statistical simulator for nanotechnology circuit design. *CoRR*, abs/0710.4633, 2007.

[10] C. Thuillet, P. Andouard, and O. Ly. A smart card power analysis simulator. In *CSE (2)*, pages 847–852, 2009.