Conversion of Security Proofs from One Leakage Model to Another: A New Issue

Jean-Sébastien Coron[1,6], Christophe Giraud[2], Emmanuel Prouff[3], Soline Renner[2,5], Matthieu Rivain[4], and Praveen Kumar Vadnala[1]

[1] Université du Luxembourg
{jean-sebastien.coron,praveen.vadnala}@uni.lu

Oberthur Technologies
Crypto and Security Group
[2] 4, allée du Doyen Georges Brus, 33 600 Pessac, France
[3] 71-73, rue des Hautes Pâtures, 92 726 Nanterre, France
{c.giraud,e.prouff,s.renner}@oberthur.com

[4] CryptoExperts
41, boulevard des Capucines, 75 002 Paris, France
matthieu.rivain@cryptoexperts.com

[5] Université Bordeaux I
351, cours de la Libération, 33 405 Talence cedex, France

[6] Tranef
jscoron@tranef.com

Abstract. To guarantee the security of a cryptographic implementation against Side Channel Attacks, a common approach is to formally prove the security of the corresponding scheme in a model as pertinent as possible. Nowadays, security proofs for masking schemes in the literature are usually conducted for models where only the *manipulated data* are assumed to leak. However in practice, the leakage is better modeled encompassing the memory transitions as *e.g.* the Hamming distance model. From this observation, a natural question is to decide at which extent a countermeasure proved to be secure in the first model stays secure in the second. In this paper, we look at this issue and we show that it must definitely be taken into account. Indeed, we show that a countermeasure proved to be secure against second-order side-channel attacks in the first model becomes vulnerable against a first-order side-channel attack in the second model. Our result emphasize the issue of porting an implementation from devices leaking only on the manipulated data to devices leaking on the memory transitions.

1 Introduction

1.1 Context

Side Channel Analysis (SCA for short) is a class of attacks that extracts information on sensitive values by analyzing a physical leakage during the execution of a cryptographic algorithm. They take advantage of the dependence between one or several manipulated value(s) and physical measurements. Implementations of block ciphers have been a privileged target and a wide number of countermeasures have been published during the last decade to protect them [1, 4–8, 10, 12–15, 17].

One of the most common techniques to protect block ciphers against SCA consists in randomly splitting each sensitive value of the processing into several shares [2,5,14]. These shares must then be propagated throughout the algorithm in such a way that no intermediate value is key-dependent, making SCA difficult to perform. This kind of countermeasures can be characterized by the number of random shares per sensitive variable: a so-called d^{th}*-order masking* splits each sensitive value into $d+1$ shares. Theoretically, such a countermeasure can always be broken by a so-called $(d+1)^{th}$-order side channel analysis, where the adversary is assumed to be able to observe the physical leakage related to the manipulation of the $d+1$ shares. However, in practice the difficulty of carrying out a higher order SCA increases exponentially with the order. As a consequence, the use of a first or second order masking scheme is often sufficient to achieve practical resistance.

When applying masking to protect a block cipher implementation, the most critical parts to deal with are the non-linear functions, also called *s-boxes*. Among the numerous methods that have been proposed in the literature, many of them have been broken, which has risen the need for a formal analysis of the security provided by such countermeasures. When the purpose is to thwart first-order SCA only, a secure and efficient solution is to use pre-computed look-up tables in RAM [6, 8]. When the countermeasure must also defeat second-order SCA, there exists no solution which is at the same time secure and very efficient for any kind of s-box. To the best of our knowledge only the schemes [4, 12, 14, 15] have a formal proof of security. The schemes proposed in [4], [12] and [15] are quite efficient but dedicated to the AES s-box only. In comparison, [14] is less efficient but it can be applied to protect any s-box implementation. In this paper, we focus on the latter one.

To guarantee the security of a cryptographic implementation against d^{th}-order SCA or to simply enable comparison between the resistance of several countermeasures, it is nowadays a common approach to formally prove the security of a scheme in a model as pertinent as possible. Two different models are generally considered in the literature. We recall these models hereafter.

When the device writes a value Z into the memory, the first leakage model assumes that the leakage L satisfies:

$$L = \varphi(Z) + B, \tag{1}$$

with φ a (non-constant) function and B an independent gaussian noise with zero mean. Such a model is said *to leak on the manipulated data bits only*. For example the leakage function φ is often the Hamming weight (HW) function (or an affine function of the HW). In that case, we usually speak about *Hamming weight model*. A more conservative choice in terms of security is to suppose that φ might be the identity function *i.e.* the leakage reveals the value of Z.
The second model assumes that the device leaks on the *memory transitions* when a value Z is manipulated. In this situation the function φ depends on Z but also on a second value Y corresponding to the initial state of the memory before the writing of Z. More precisely, we have:

$$L = \varphi(Z \oplus Y) + B. \tag{2}$$

In the particular case where φ is the HW function, the leakage L defined in (2) corresponds to the so-called Hamming distance (HD) model.
Several works have demonstrated the validity of HW and HD models in practice, which are today commonly accepted by the SCA community. However other more precise models exist in the literature (see for instance [3, 9, 16]).

In the rest of this paper, we keep the generality by considering two models : ODL model (*Only manipulated Data Leak*) and MTL model (*Memory Transition Leak*), each of them being defined by the leakage function expressed in (1) and (2) respectively.

1.2 ODL Model *vs.* MTL Model

Except very rare exceptions (*e.g.* [10]), security proofs in the literature are usually conducted in ODL model. This is in particular the case of the countermeasures proposed in [14]. However, in practice, the leakage is better modeled by MTL model. Starting from this observation, a natural question is to decide at which extent a countermeasure proved to be secure in ODL model stays secure in MTL model. Very close to this question an interesting and practically relevant problem is the design of methods to transform an implementation secure in the first model into a new implementation secure in the second. Hence, if we assume that the memory transitions leak information, the leakage is modeled by $\varphi(Y \oplus Z) + B$. In such a model a masking countermeasure may become ineffective. For instance, if Z corresponds to a masked variable $X \oplus M$ and if Y equals the mask, then the leakage reveals information on X. A very straightforward idea to deal with this issue is to erase the memory before each new writing (*e.g.* set Y to 0 in our example). One may note that such a technique is often used in practice at either the hardware or software level. Using such a method, the leakage $\varphi(Y \oplus Z) + B$ is replaced by the sequence of consecutive leakages $\varphi(Y \oplus 0) + B_1$ and $\varphi(0 \oplus Z) + B_2$ that is equivalent to

$\varphi(Y) + B_1$ and $\varphi(Z) + B_2$. The single difference with classical ODL model is the additional assumption that the execution leaks the content of the memory before the writings. Since this leakage corresponds to a variable that has been manipulated prior to Z, it is reasonable to assume that the leakage $\varphi(Y) + B_1$ has already been taken into account when establishing the security of the countermeasure. As a consequence, this way to implement a countermeasure proved to be secure in ODL model seems at a first glance also offers security on a device leaking in MTL model.

In this paper, we emphasize that a countermeasure proved to be secure in ODL model may no longer stay secure in MTL model. Indeed, we exhibit a case where a countermeasure proved to be second-order resistant in ODL model does no longer provide security against first-order SCA when implemented in a device leaking on the memory transitions. Then, we show that the natural method proposed above to transfer a countermeasure resistant in ODL model into a countermeasure resistant in MTL model is flawed. Those two results enlighten the actual lack of a framework to solve the (practically) important issue of porting an implementation from one family of devices to the other one.

1.3 Paper Organization

This paper is organized as follows. In Section 2, we briefly recall a second-order countermeasure proved to be secure in ODL model [14]. In Section 3, we show that such a countermeasure can be broken by using a first-order attack in MTL model. To thwart this attack, we apply in Section 4.1 the method described previously which erases the memory before each new writing and we show that this method does not provide an implementation secure in the second model. We provide the results of a practical implementation of our attacks in Section 5. Finally we conclude this paper in Section 6.

2 Securing Block Cipher Against 2O-SCA

Most of the countermeasure published in the literature to thwart SCA are based on the algebraic properties of the targeted algorithm (*e.g.* AES). However, when the corresponding algorithm involves s-boxes with no particular algebraic structure (*e.g.* those in DES, PRESENT or FOX ciphers), then only the methods proposed in [14] enable to achieve second-order security. In the following, we focus on the last case where a *random-like* s-box must be implemented in a secure way w.r.t. 2O-SCA. For such a purpose, we focus on the second variant proposed in [14] (this choice can for instance have been made because of its low RAM consumption compared to the first variant).

Based on a secure primitive $compare_b$ defined such that $compare_b(x, y)$ equals b if $x = y$ and $\overline{b}$ otherwise (see [13, Appendix A] for more details), the authors in [14] propose the algorithm below:

Algorithm 1 Computation of a 2O-masked s-box output from a 2O-masked input
INPUTS: a masked value $\tilde{x} = x \oplus t_1 \oplus t_2 \in \mathbb{F}_2^n$, the pair of input masks $(t_1, t_2) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, a pair of output masks $(s_1, s_2) \in \mathbb{F}_2^m \times \mathbb{F}_2^m$, a (n, m) s-box function F
OUTPUT: the masked s-box output $F(x) \oplus s_1 \oplus s_2 \in \mathbb{F}_2^m$

1. $b \leftarrow rand(1)$
2. **for** $a = 0$ **to** $2^n - 1$ **do**
3. $\quad cmp \leftarrow compare_b(t_1 \oplus a, t_2)$
4. $\quad R_{cmp} \leftarrow (F(\tilde{x} \oplus a) \oplus s_1) \oplus s_2$
5. **return** R_b

To compute $F(x) \oplus s_1 \oplus s_2$, the core idea of Algorithm 1 is to successively read all values of the lookup table F from index $\tilde{x} \oplus a$ with $a = 0$ to index $\tilde{x} \oplus a$ with $a = 2^n - 1$. When the correct value $F(x) \oplus s_1 \oplus s_2$ is accessed, it is stored in a pre-determined register R_b whereas the other values $F(\tilde{x} \oplus a) \oplus s_1 \oplus s_2$, with $\tilde{x} \oplus a \neq x$, are stored in a garbage register $R_{\overline{b}}$. In practice two registers R_0 and R_1 are used and their roles are chosen thanks to a random bit b.

Depending on the loop index a, the fourth step of Algorithm 1 processes the following operation:

$$\begin{cases} cmp \leftarrow b \ ; & R_{cmp} \leftarrow F(x) \oplus s_1 \oplus s_2 & \text{if } a = t_1 \oplus t_2 \\ cmp \leftarrow \overline{b} \ ; & R_{cmp} \leftarrow F(\tilde{x} \oplus a) \oplus s_1 \oplus s_2 & \text{otherwise} \end{cases} . \tag{3}$$

In view of (3), it may be observed that the register R_b is modified only once whereas $R_{\overline{b}}$ changes $2^n - 1$ times. As proven in [14], this behavior difference between the registers R_b and $R_{\overline{b}}$ cannot be successfully exploited by a second-order attack when the device leaks in the ODL model. The proof can be straightforwardly extended to any leakage model called *linear*, in which all bits of the manipulated data leak independently. However, if Algorithm 1 must be implemented on a physical device with a different leakage model, then the security proof in [14] can no longer be invoked. Hence, since the most common alternative is MTL model, it is particularly interesting to investigate whether Algorithm 1 stays secure in this context. In the next section, we put forward the kind of security issues brought by a straightforward implementation of Algorithm 1 on a device leaking the memory transition. In particular, for a specific (but quite natural) implementation, we exhibit a first-order SCA.

3 Attack of Algorithm 1 in the MTL Model

This section is organized as follows: first we present a straightforward implementation of the 2O-SCA countermeasure described in Algorithm 1. Then we expose how a first-order attack in MTL model can break this second-order countermeasure.

In the analysis developed in this paper, we will denote random variables by capital letters (*e.g.* X) and their values by small letters (*e.g.* x).

3.1 Straightforward Implementation of Algorithm 1

In the following, we assume that the considered device is based on an assembler language for which a register R_A is used as accumulator. Moreover we assume that registers R_A, R_0 and R_1 are initially set to zero.

Based on these assumptions, the fourth step of Algorithm 1 can be implemented in the following way:

$$\begin{array}{ll} 4.1 & R_A \leftarrow \tilde{x} \oplus a \\ 4.2 & R_A \leftarrow F(R_A) \\ 4.3 & R_A \leftarrow R_A \oplus s_1 \\ 4.4 & R_A \leftarrow R_A \oplus s_2 \\ 4.5 & R_{cmp} \leftarrow R_A \end{array} \tag{4}$$

During this processing where $\tilde{X} = X \oplus T_1 \oplus T_2$, the initial content of register R_{cmp}, denoted by Y, satisfies the following equation depending on the values of the loop index a, T_1 and T_2:

$$Y = \begin{cases} 0 & \text{if } a = 0 \ , \\ 0 & \text{if } a = 1 \text{ and } T_1 \oplus T_2 = 0 \ , \\ 0 & \text{if } a > 0 \text{ and } T_1 \oplus T_2 = a \ , \\ F(\tilde{X} \oplus (a-2)) \oplus S_1 \oplus S_2 & \text{if } a > 1 \text{ and } T_1 \oplus T_2 = (a-1) \ , \\ F(\tilde{X} \oplus (a-1)) \oplus S_1 \oplus S_2 & \text{otherwise.} \end{cases} \tag{5}$$

In the following we will show that the distribution of the value Y defined in (5) brings information on the sensitive variable X. We will consider two cases depending on whether R_A equals R_{cmp} or not.

3.2 Description of the First-Order Attack when $R_A = R_{cmp}$

According to this decomposition, if we assume that the register R_{cmp} is the accumulator register, then Step 4.5 of (4) is unnecessary and the register R_{cmp} leaks at each state. This is in particular the case at Step 4.1,

In this part, we assume that the physical leakage of the device is modeled by MTL model and hence the leakage L associated to Step 4.1 of (4) satisfies:

$$L \sim \varphi(Y \oplus \tilde{X} \oplus a) + B \ , \tag{6}$$

where Y denotes the initial state of R_{cmp} before being updated with $\tilde{X} \oplus a$, defined above by (5).

From (5) and (6), we deduce:

$$L = \begin{cases} \varphi(\tilde{X}) \quad + B & \text{if } a = 0 \ , \\ \varphi(X \oplus 1) + B & \text{if } a = 1 \text{ and } T_1 \oplus T_2 = 0 \ , \\ \varphi(X) \quad + B & \text{if } a > 0 \text{ and } T_1 \oplus T_2 = a \ , \\ \varphi(F(\tilde{X} \oplus (a-2)) \oplus S_1 \oplus S_2 \oplus \tilde{X} \oplus a) + B & \text{if } a > 1 \text{ and } T_1 \oplus T_2 = (a-1) \ , \\ \varphi(F(\tilde{X} \oplus (a-1)) \oplus S_1 \oplus S_2 \oplus \tilde{X} \oplus a) + B & \text{otherwise.} \end{cases}$$

When $a = 0$, the leakage L is an uniform value which brings no information on the value X. Therefore in the following, we omit this particular case.

Hence, we have

$$L = \begin{cases} \varphi(X) \quad +B & \text{if } T_1 \oplus T_2 = a \ , \\ \varphi(X \oplus 1) +B & \text{if } T_1 \oplus T_2 = 0 \ \text{ and } \ \ a = 1 \ , \\ \varphi(Z) \quad +B & \text{otherwise ,} \end{cases} \tag{7}$$

with Z a variable independent of X and with uniform distribution.

In view of (7), the leakage L depends on X. Indeed, the mean of $(L|X = x)$ satisfies:

$$E(L \mid X = x) = \begin{cases} \frac{1}{2^n} \times (\varphi(x) + \varphi(x \oplus 1)) + \frac{2^n - 2}{2^n} \times E(\varphi(Z)) & \text{if } a = 1 \ , \\ \\ \frac{1}{2^n} \times \varphi(x) + \frac{2^n - 1}{2^n} \times E(\varphi(Z)) & \text{if } a > 1 \ , \end{cases}$$

or equivalently (since Z has uniform distribution):

$$E(L \mid X = x) = \begin{cases} \frac{1}{2^n} \times (\varphi(x) + \varphi(x \oplus 1)) + \frac{n \times (2^n - 2)}{2^{n+1}} & \text{if } a = 1 \ , \\ \\ \frac{1}{2^n} \times \varphi(x) + \frac{n \times (2^n - 1)}{2^{n+1}} & \text{if } a > 1 \ . \end{cases} \tag{8}$$

When $a > 1$, the mean in (8) is an affine function of $\varphi(x)$ and it is an affine function of $(\varphi(x) + \varphi(x \oplus 1))$ otherwise. Therefore in both cases the mean leakage reveals some information on X.
An adversary can thus target the second round in Algorithm 1 (*i.e.* $a = 1$) and get a sample of observations for the leakage L defined as in (6). The value X typically corresponds to the bitwise addition between a secret sub-key K and a known plaintext subpart M. In such a case and according to the statistical dependence shown in (8), the set of observations can be used to perform a first-order SCA allowing an attacker to recover the secret value K.

As an illustration, we simulated a first-order CPA in the Hamming weight model without noise targeting the second loop (namely $a = 1$) with the AES s-box. The secret key byte was recovered with a success rate of 99% by using 1.000.000 acquisitions.

3.3 Description of the First-Order Attack when $R_A \neq R_{cmp}$

In this part, the accumulator register R_A is assumed to be different from the register R_{cmp}. Under such an assumption, Step 4.5 in (4) leaks the transition between the initial content Y of R_{cmp} and the current content of R_A. Namely, after denoting $T_1 \oplus T_2$ and $S_1 \oplus S_2$ by T and S respectively, we have:

$$L = \varphi(Y \oplus F(X \oplus T \oplus a) \oplus S) + B. \tag{9}$$

Due to (5), Relation (9) may be developed in the following way according to the values of a and T:

$$L = \begin{cases} \varphi(\ F(X \oplus T) \oplus S\ ) + B & \text{if } a = 0, \\ \varphi(\ F(X) \oplus S\ ) + B & \text{if } a = 1 \text{ and } T = (a-1), \\ \varphi(\ F(X) \oplus S\ ) + B & \text{if } a > 0 \text{ and } T = a, \\ \varphi(\ D_{a\oplus(a-2)}F(X \oplus (a-2) \oplus (a-1)\ ) + B & \text{if } a > 1 \text{ and } T = (a-1), \\ \varphi(\ D_{a\oplus(a-1)}F(X \oplus (a-1) \oplus T)\ ) + B & \text{otherwise,} \end{cases} \tag{10}$$

where D_yF denotes the *derivative* of F *with respect to* $y \in \mathbb{F}_2^n$, which is defined for every $x \in \mathbb{F}_2^n$ by $D_yF(x) = F(x) \oplus F(x \oplus y)$.

In the three first cases in (10), the presence of S implies that the leakage L is independent of X. Indeed, in these cases the leakage is of the form $\varphi(Z) + B$ where Z is an uniform random variable independent of X. In the last two cases, S does not appear anymore. As a consequence it may be checked that the leakage L depends on X. Indeed, due to the law of total probability, for any x and $a = 1$, the mean of $(L|X = x)$ satisfies:

$$E(L|X = x) = \frac{2\mu}{2^n} + \frac{1}{2^n} \sum_{t=2}^{2^n-1} \varphi(D_aF(x \oplus t)), \tag{11}$$

where μ denotes the expectation $E[\varphi(U)]$ with U uniform over $\mathbb{F}_2^n$ (*e.g.* for $\varphi =$ HW we have $\mu = n/2$). And when $a > 1$, the mean of $(L|X = x)$ satisfies:

$$E(L|X = x) = \frac{\mu}{2^n} + \frac{1}{2^n}\ \varphi(D_{a\oplus(a-2)}F(x \oplus (a-2) \oplus (a-1)))$$

$$+\frac{1}{2^n} \sum_{t=0, t\neq a,(a-1)}^{2^n-1} \varphi(D_{a\oplus(a-1)}F(x \oplus (a-1) \oplus t)). \tag{12}$$

From an algebraic point of view, the sums in (11) and (12) may be viewed as the mean of the value taken by $D_aF(x \oplus t)$ (respectively $D_{a\oplus(a-1)}F(x \oplus (a-1) \oplus t)$) over the coset $x \oplus \{t, t \in [2, 2^n - 1]\}$ (respectively $x \oplus \{t, t \in [0, 2^n - 1] \setminus \{a-1, a\}\}$). Since those cosets are not all equal, the means are likely to be different for some values of x. Let us for instance consider the case of F equal to the AES s-box and let us assume that φ is the identity function. In Relation (11), the sum equals 34066 if $x = 1$ and equals

34046 if $x = 2$. When $a > 1$, we have the similar observation.

From (11) and (12), we can deduce that the mean leakage reveals information on X and thus, the set of observations can be used to perform a first-order SCA.

By exhibiting several attacks in this section, we have shown that the second-order countermeasure proved to be secure in ODL model may be broken by a first-order attack in MTL model. These attacks demonstrate that a particular attention must be paid when implementing Algorithm 1 on a device leaking in MTL model. Otherwise, first-order leakage may occur as those exploited in the attacks presented above. As already mentioned in the introduction, a natural solution to help the security designer to deal with those security traps could be to systematically erase the registers before any writing. This solution is presented and discussed in the next section.

4 Study of a Straightforward Patch

In the following, we present a straightforward method to patch the flaw exhibited in the previous section. The aim of this patch is to transform an implementation secure in ODL model into an implementation secure in MTL model. It essentially consists in erasing the memory before each new writing. In this section, we evaluate this strategy when applied to implement Algorithm 1 leaking in MTL model. Then, we show that this natural method does not suffice to go from security in ODL model to security in MTL model. Indeed, we present a second-order attack against the obtained second-order countermeasure.

4.1 Transformation of Algorithm 1 into a MTL-Resistant Scheme

As in the previous section, we assume that the leakage model is MTL model and that the registers R_b and $R_{\overline{b}}$ are initially set to zero. In order to preserve the security proof given in the first model, we apply a solution consisting in erasing the memory before each new writing.

Based on these assumptions, the fourth step of Algorithm 1 can be implemented in the following way:

$$\begin{array}{l} 4.1\ R_{cmp} \leftarrow 0 \\ 4.2\ R_{cmp} \leftarrow F(\tilde{x} \oplus a) \oplus s_1 \oplus s_2 \end{array} \tag{13}$$

As previously, we assume that the initial state of R_{cmp} before Step 4.1 is equal to Y. Then, according to this decomposition, the register R_{cmp} is set to 0 before the writing of $Z = F(\tilde{X} \oplus a) \oplus S_1 \oplus S_2$ in the Step 4.2. Hence, the leakage defined by (6) is replaced by the sequence of consecutive leakages $\varphi(Y, 0) + B_1$ (Step 4.1), $\varphi(0, Z) + B_2$ (Step 4.2), that is $\varphi(Y) + B_1$, $\varphi(Z) + B_2$. However this model is not equivalent to the ODL model since here the previous value in R_{cmp} leaks whenever it is erased. And as we show hereafter, such a leakage enables a second-order attack breaking the countermeasure althought secure in the ODL model.

4.2 Description of a Second-Order Attack

To perform our second-order attack, we use two information leakages L_1 and L_2 during the same execution of Algorithm 1 implemented with (13).

The first leakage L_1 corresponds to the manipulation of $\tilde{X}$ prior to Algorithm 1. L_1 thus satisfies:

$$L_1 \sim \varphi(\tilde{X}) + B_0. \tag{14}$$

The second leakage L_2 corresponds to Step 4.1 of (13). Thus it satisfies:

$$L_2 \sim \varphi(Y) + B_1. \tag{15}$$

From (5) and (15), we deduce:

$$L_2 = \begin{cases} \varphi(0) + B_1 & \text{if } a = 0 \ , \\ \varphi(0) + B_1 & \text{if } a = 1 \text{ and } T_1 \oplus T_2 = 0 \ , \\ \varphi(0) + B_1 & \text{if } a > 0 \text{ and } T_1 \oplus T_2 = a \ , \\ \varphi(F(\tilde{X} \oplus (a-2)) \oplus S_1 \oplus S_2) + B_1 & \text{if } a > 1 \text{ and } T_1 \oplus T_2 = (a-1) \ , \\ \varphi(F(\tilde{X} \oplus (a-1)) \oplus S_1 \oplus S_2) + B_1 & \text{otherwise.} \end{cases} \tag{16}$$

which implies that:

$$L_2 = \begin{cases} \varphi(0) + B_1 & \text{if } a = 0 \ , \\ \varphi(0) + B_1 & \text{if } a = 1 \text{ and } T_1 \oplus T_2 = 0 \text{ or } 1 \ , \\ \varphi(Z) + B_1 & \text{if } a = 1 \text{ and } T_1 \oplus T_2 \neq 0 \text{ or } 1 \ , \\ \varphi(0) + B_1 & \text{if } a > 1 \text{ and } T_1 \oplus T_2 = a \ , \\ \varphi(Z) + B_1 & \text{if } a > 1 \text{ and } T_1 \oplus T_2 \neq a \ , \end{cases} \tag{17}$$

where Z is a variable independent of X and with uniform distribution.

From (17), the leakage is independent from $T_1 \oplus T_2$ when $a = 0$. For this reason, in the following we only study the mean of L_2 for $a > 0$:

$$E(L_2) = \begin{cases} \varphi(0) & \text{if } a = 1 \text{ and } T_1 \oplus T_2 = 0 \text{ or } 1 \ , \\ \varphi(Z) & \text{if } a = 1 \text{ and } T_1 \oplus T_2 \neq 0 \text{ or } 1 \ , \\ \varphi(0) & \text{if } a > 1 \text{ and } T_1 \oplus T_2 = a \ , \\ \varphi(Z) & \text{if } a > 1 \text{ and } T_1 \oplus T_2 \neq a \ , \end{cases}$$

or equivalently (since Z has uniform distribution):

$$E(L_2) = \begin{cases} \varphi(0) & \text{if } a = 1 \text{ and } T_1 \oplus T_2 = 0 \text{ or } 1 \ , \\ \varphi(0) & \text{if } a > 1 \text{ and } T_1 \oplus T_2 = a \ , \\ \frac{n}{2} & \text{otherwise.} \end{cases} \tag{18}$$

On the other hand, the leakage L_1 depends by definition on $X \oplus T_1 \oplus T_2$. As a consequence, one deduces that the pair (L_1, L_2) statistically depends on the sensitive value X. Moreover, it can be seen in (18) that the leakage on

$T_1 \oplus T_2$ is maximal when $a = 1$. An adversary can thus target the second loop in Algorithm 1 (*i.e.* $a = 1$), make measurements for the pair of leakages (L_1, L_2) and then perform a 2O-CPA to extract information on X from those measurements.

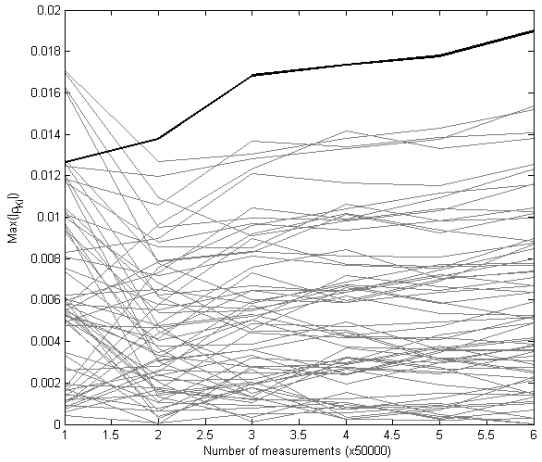

Fig. 1. Convergence with simulated curves without noise, for $a = 1$.

We have simulated such a 2O-SCA with $X = M \oplus K$ where M is a 8-bit value known to the attacker and K a 8-bit secret key value. By combining L_1 and L_2 using the normalized multiplication and the optimal prediction function as defined in [11], the secret value k is recovered with a success rate of 99% by using less than 200.000 curves. Fig.1 represents the convergence of the maximal correlation value for different key guesses over the number of leakage measurements. Each curve corresponds to some hypothesis on the secret K. In particular the black curve corresponds to the correct hypothesis k.

The second-order attack presented in this section show that erasing registers before writing a new value does not suffice to port the security of an implementation from ODL model to MTL model. For the case of Algorithm 1, a possible patch is to erase R_{cmp} using a random value. However, though this patch works in the particular case of Algorithm 1, it does not provide a generic method to transform a dth-order countermeasure secure in the ODL model to a dth-order countermeasure secure in the MTL model. The design of such a generic method is an interesting problem that we leave open for future research.

5 Experimental Results

This section provides the practical evaluation of the attacks presented above. We have verified the attacks on block ciphers with two different kinds of s-boxes: an 8-bit to 8-bit s-box (AES) and two 4-bit to 4-bit s-boxes (PRESENT and Klein). We have implemented Algorithm 1 as described in Section 4.1 on a 8-bit microcontroller. Using 2O-CPA, we were able to find the secret key for all three s-boxes. In case of the 4×4 s-boxes, we needed fewer than 10.000 power traces to find the correct key. However, for the 8×8 s-box, the number was much higher, since more than 150.000 traces were required to distinguish the correct key from the rest of the key guesses.

Initially, we set the value in the two memory locations R_0 and R_1 to zero. We randomly generate the plaintexts m_i and the input/output masks $t_{i,1}, t_{i,2}$ and $s_{i,1}, s_{i,2}$ using a uniform pseudo-random number generator where the value of i varies from 1 to N (i.e., the number of measurements). Then, we calculate $\tilde{x_i}$ from the correct key k via $\tilde{x_i} = k \oplus m_i \oplus t_{i,1} \oplus t_{i,2}$. As described in Section 4.1, before writing a new value to any memory location, we first erase its contents by writing 0, and then write the new value as shown in (13). For verifying the attacks, we only consider the power traces where $a = 1$. During respectively the manipulation of the $\tilde{x}_i$ and the memory erasing, we measure the power consumption of the device. This results in a sample of pairs of leakage points that are combined thanks to the centered product combining function defined in [11]. For each key hypothesis k_j, the obtained combined leakage sample $(\mathcal{L}_i)_{1 \leq i \leq N}$ is correlated with the sample of hypotheses $(HW(m_i \oplus k_j))_{1 \leq i \leq N}$. The key guess for which the correlation coefficient is the maximum will be the correct key.

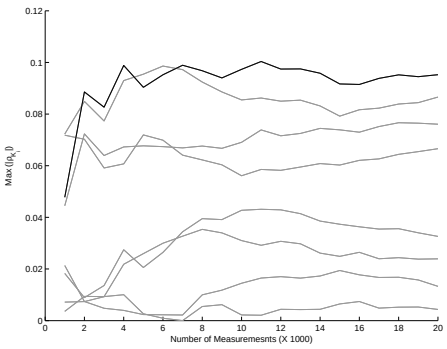

Fig. 2. Convergence with practical implementation of 20-CPA for Klein.

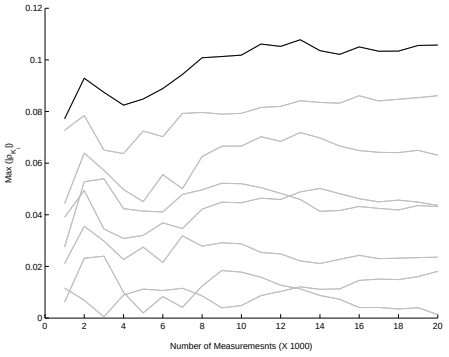

Fig. 3. Convergence with practical implementation of 20-CPA for PRESENT

Figure 2 and Figure 3 show the correlation traces for a 2O-CPA on the Klein and PRESENT s-boxes, respectively. As it can be observed, the right key is found in both cases with less than 10.000 power traces. Figure 4 shows the correlation

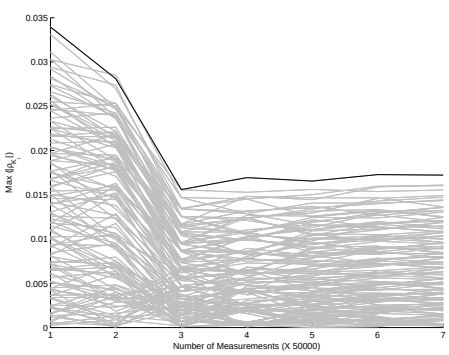

Fig. 4. Convergence with practical implementation of 20-CPA for AES.

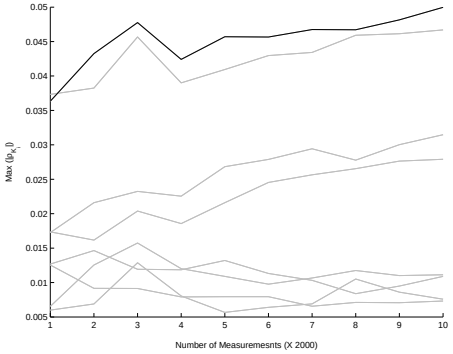

Fig. 5. Convergence with practical implementation of 10-CPA for PRESENT.

traces for a 2O-CPA on the AES s-box. Here the convergence of the traces to the correct key is observable only after 150.000 traces. Finally, Figure 5 shows the first-order attack on the PRESENT s-box in the Hamming Distance model as described in Section 3.2. Here we implemented Algorithm 1 directly without the additional step of erasing the memory contents before performing a write operation. The power traces are collected for 50.000 inputs, and only the traces corresponding to the case $a = 1$ are considered. The correct key candidate can be identified with less than 10.000 traces.

6 Conclusion and Perspectives

In this paper, we have shown that a particular attention must be paid when implementing a countermeasure proved to be secure in one model on devices leaking in another one. In particular we have shown that the second-order countermeasure proposed in [14] together with a security proof in ODL model is broken by first-order SCA when running on a device leaking in MTL model. Then, we have focused on a method that looked at first glance very natural to convert a scheme resistant in ODL model in a new one secure in MTL model. Our analysis pointed out flaws in the conversion method and hence led us to identify two new issues that we think to be very promising for further research. The first issue is the design of a generic countermeasure proved to be secure in any practical model and the second is the design of a method of porting the security from a model to another one.

References

1. J. Blömer, J. G. Merchan, and V. Krummel. Provably Secure Masking of AES. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004.

2. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
3. J. Doget, E. Prouff, M. Rivain, and F. Standaert. Univariate side channel attacks and leakage modeling. In W. Schindler and S. Huss, editors, *Second International Workshop on Constructive Side-Channel Analysis and Secure Design – COSADE 2011*, 2011.
4. L. Genelle, E. Prouff, and M. Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*. Springer, 2011.
5. L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
6. T. Messerges. Securing the AES Finalists against Power Analysis Attacks. In B. Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2000.
7. E. Oswald, S. Mangard, and N. Pramstaller. Secure and Efficient Masking of AES – A Mission Impossible ? Cryptology ePrint Archive, Report 2004/134, 2004.
8. E. Oswald and K. Schramm. An Efficient Masking Scheme for AES Software Implementations. In J. Song, T. Kwon, and M. Yung, editors, *WISA 2005*, volume 3786 of *Lecture Notes in Computer Science*, pages 292–305. Springer, 2006.
9. E. Peeters, F.-X. Standaert, and J.-J. Quisquater. Power and Electromagnetic Analysis: Improved Model, Consequences and Comparisons. *Integration*, 40(1):52–60, 2007.
10. E. Prouff and M. Rivain. A Generic Method for Secure SBox Implementation. In S. Kim, M. Yung, and H.-W. Lee, editors, *WISA*, volume 4867 of *Lecture Notes in Computer Science*, pages 227–244. Springer, 2008.
11. E. Prouff, M. Rivain, and R. Bévan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Comput.*, 58(6):799–811, 2009.
12. E. Prouff and T. Roche. Higher-order glitches free implementation of the aes using secure multi-party computation protocols. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2011.
13. M. Rivain, E. Dottax, and E. Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. Cryptology ePrint Archive, Report 2008/021, 2008. `http://eprint.iacr.org/`.
14. M. Rivain, E. Dottax, and E. Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In T. Baignères and S. Vaudenay, editors, *Fast Software Encryption – FSE 2008*, Lecture Notes in Computer Science, pages 127–143. Springer, 2008.
15. M. Rivain and E. Prouff. Provably secure higher-order masking of aes. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, Lecture Notes in Computer Science. Springer, 2010.
16. W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*. Springer, 2005.

17. K. Schramm and C. Paar. Higher Order Masking of the AES. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.