A Generic Method for Secure SBox Implementation

Emmanuel Prouff[2] and Matthieu Rivain[1,2]

[1] University of Luxembourg
Faculty of Sciences, Technology and Communication
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
[2] Oberthur Card Systems,
71-73 rue des Hautes Pâtures,
92726 Nanterre Cedex, France
{m.rivain,e.prouff}@oberthurcs.com

Abstract. Cryptographic algorithms embedded in low resource devices are vulnerable to side channel attacks. Since their introduction in 1996, the effectiveness of these attacks has been highly improved and many countermeasures have been invalidated. It was especially true for countermeasures whose security was based on heuristics and experiments. Consequently, there is not only a need for designing new and various countermeasures, but it is also necessary to prove the security of the new proposals in formal models. In this paper we provide a simple method for securing the software implementation of functions called SBoxes that are widely used in symmetric cryptosystems. The main advantage of the proposed solution is that it does not require any RAM allocation. We analyze its efficiency and we compare it with other well-known countermeasures. Moreover, we use a recently introduced proof-of-security framework to demonstrate the resistance of our countermeasure from the viewpoint of Differential Power Analysis. Finally, we apply our method to protect the AES implementation and we show that the performances are suitable for practical implementations.

1 Introduction and Motivations

Side Channel Analysis are powerful attacks that utilize side channel leakage of embedded devices such as timing execution or power consumption to obtain information about secret data. It essentially allows two kinds of *Power Attacks*: the *Simple Power Analysis* (SPA) and the *Differential Power Analysis* (DPA). SPA consists in directly interpreting power consumption measurements and in identifying the execution sequence. In a DPA, the attacker focuses on the power consumption of a single instruction and performs statistical tests to reveal some correlations between the distribution of the measurement values and the *sensitive data* (*i.e.* depending on a secret value) manipulated by the instruction. Since the publication of the first DPA, many papers describing either countermeasures or attack improvements have been published (see [1,3,4,12] for example). Among these improvements, *Higher order DPA attacks* (HODPA) are of particular interest. They extend the DPA by considering a set of several instructions instead of a single one. The number d of instructions targeted by the attack is called *order* of the DPA.

The most common way of thwarting DPA involves random values (called *masks*) to de-correlate the leakage signal from the sensitive data which are manipulated [1, 4, 6]. If every sensitive variable is protected with a single mask, the implementation may thwart first order DPA but it can be theoretically attacked by a second order DPA targeting both the mask and the masked value. To thwart second order DPA (and more generally d-th order DPA), every sensitive variable must be masked with 2 (resp. d) random values. This implies that the timing-memory cost of the existing masking countermeasures increases greatly with the order of the DPA they aim to counteract. For applications where time constraints are very strong (such as contact-less applications), it may be considered as sufficient to mask all the sensitive data with a small number of random values generated at the beginning of the algorithm execution. The performances of many DPA countermeasures have been studied in this model. However recent results (see [13]) showed that second order DPA represents a real practical threat, especially when the same value is used to mask several intermediate results throughout the algorithm. Therefore, as noticed in [5, 15], the masks must be re-generated as often as possible and the analysis of the efficiency of a countermeasure has to take this fact into account. In this model, it appears that only a few among the existing countermeasures are still efficient in a low resource context and there is therefore still a need for investigating new and various countermeasures that thwart first order DPA efficiently when the masks change frequently.

Due to the very large variety of Side Channel Attacks reported against cryptosystems and devices, sensitive applications (*e.g.* Banking, GSM or Identity Card) cannot make use of countermeasures with *ad hoc* security but need countermeasures which are provably secure against a precisely modeled adversary. Recently, new notions and tools have been introduced to evaluate the security of an implementation against Side Channel Attacks [17, 18, 20]. They allow for the formal validation or the formal invalidation of the resistance of a countermeasure under some realistic assumptions on the behavior of the device and on the power of the adversary.

In this paper, we focus on DPA against block cipher algorithms. The most critical part when securing implementations of such algorithms against DPA is to protect their non-linear operations (*i.e.* the calls to the SBoxes). In the next section, we recall the methods which have been proposed in the Literature. Then, we introduce in Sect. 3 a new and simple countermeasure which counteracts first order DPA against SBox implementations whatever the algebraic structure of the SBox. When the masks change frequently, we argue that the new method has a good efficiency compared to the other generic methods. In Sect. 4, we analyze the security of our proposal by following the methodology described in [20]. In Appendix A, we apply our method to protect the implementation of the AES SBox and we compare its efficiency and its security with the ones of other existing countermeasures.

2 Secure Implementation of Non-linear Functions in the Literature

2.1 State of the Art of the Generic Methods

To counteract DPA attacks, one usually tries to make the power consumption signal as independent as possible of the sensitive data manipulated by the algorithm. Goubin and Patarin proposed in [6] a general solution, called *duplication method*, to protect the implementation of an algorithm. In this approach, every sensitive variable x is split into d blocks r_1, ..., r_d and every cryptographic primitive F that manipulates x is associated to d functions F_1, ..., F_d and to a simple transformation σ (*e.g.* a simple bitwise addition) such that $F(x) = \sigma(F_1[r_1, ..., r_d], \cdots, F_d[r_1, ..., r_d])$. Goubin and Patarin showed that an implementation protected by duplication method thwarts first order DPA if for every x (resp. every $F[x]$) the d blocks r_1, ..., r_d (resp. $F_1[r_1, ..., r_d]$, ... , $F_d[r_1, ..., r_d]$) are never manipulated at the same time.

Another approach consists in masking all the sensitive internal data with random values. Depending on the kind of operations performed by the linear parts of the algorithm, the mask values are introduced by modular addition, bitwise addition or multiplication. After selecting the masking operation $\star$, the implementation of a cryptographic function F is rendered resistant to first order DPA by solving the following problem:

Problem 1. Knowing $x \star r$, r and s, compute $F(x) \star s$ such that every value of the power consumption signal is independent of x.

If the function F is linear for the law $\star$, then solving the above problem is a simple task. Indeed, since $F(x \star r)$ equals $F(x) \star F(r)$, we have $F(x) \star s = F(x \star r) \star F(r) \star s$. If F is non-linear for the law $\star$ (which is the case when F is a SBox), then designing an implementation solving Problem 1 is much more difficult. Several kinds of methods have been proposed in the Literature and we recall two of them hereafter.

The first one, called *re-computation method* [1,11], involves the computation of a table corresponding to the masked SBox and the generation of one or several random value(s). In its most elementary version, two random values r and s are generated and the table $T^\star$ representing the function $F^\star : x \mapsto F(x \star r) \star s$ is computed from F and stored in the RAM of the device. Then, each time the masked value $F[x] \star s$ has to be computed from the masked input $x \star r$, the table $T^\star$ is accessed. For such a method, the number of tables to be recomputed during the execution of the algorithm equals the number of different input/output masks which is allowed.

Remark 1. The re-computation method is a particular case of the duplication method where d equals 2, where the sensitive value x is split into $r_1 = x \star r$

and $r_2 = r$ and where the functions F_1 and F_2 equal $r_1 \mapsto S[r_1] \star s$ and $r_2 \mapsto s$ respectively.

The second kind of methods, that we call here *SBox secure calculation*, has been essentially applied to protect AES implementations [4, 7, 18, 19, 21] due to the strong algebraic structure of the AES SBox. The SBox outputs are not directly obtained by accessing a table but are computed *on-the-fly* by using a mathematical representation F of the SBox. Then, each time the masked value $F[x] \star s$ must be computed from the pair $(x \star r, r)$, an algorithm performing F and parameterized by the 3-tuple $(x \star r, r, s)$ is executed. The computation of F is split into elementary operations (bitwise addition, bitwise multiplication, addition, multiplication, ...) and/or is performed in spaces of small dimensions (*e.g.* 4) by accessing one or several look-up table(s) (see [15]). The security of such a method is achieved by protecting each elementary operation and each memory transfer.

When the same pair of input/output masks is used throughout the algorithm, the latter is said to be protected in the *single-mask protection mode*. In such a mode, a new pair of input/output masks (r, s) is generated at each execution of the algorithm and every computation $F(x)$ performed during the execution is protected with this single pair. When the algorithm is protected in this mode, SBox secure calculation methods are often much more costly than the re-computation methods since they essentially replace a single access to a table by numerous logical operations and memory transfers. This difference between the performances of the two methods decreases when the number of different masks generated to protect the SBox calculations increases. In the *multi-mask protection mode*, the pair of masks (r, s) is re-generated each time a calculation $F(x)$ must be protected (and thus many times per algorithm execution). In such a context, the SBox secure calculation methods become more appropriate and induce a smaller timing/memory overhead than the re-computation methods. Indeed, when re-computation methods are used in the multi-mask protection mode, a new table must be computed from F after each re-generation of masks (*i.e.* before every computation $F(x)$).

As discussed in the previous paragraph, the choice between the first and the second category of methods highly depends on the protection mode, single-mask or multi-mask, in which the algorithm is implemented. We compare the two modes in the next section.

2.2 Single-mask Protection Mode *versus* Multi-mask Protection Mode

When it is only required to thwart first order DPA, then implementing the algorithm in the single mask protection mode is sufficient. However recent results (*e.g.* [13]) show that second order DPA can represent a real practical threat when the amount of information leaking in the two consumption points targeted by the attacker is sufficiently high to make the effects of the de-synchronization and of the noise negligible. More generally, the analyses of second order DPA published in [8,13,23] illustrate that the complexities of the various second order DPA are very different and show that some of them must be considered when implementing an algorithm for secure applications. In fact, as already put forward in [5,15], second order DPA are especially effective when the same pair of masks is used to protect all the inputs/outputs of the cryptographic primitives (*e.g.* all the inputs/outputs $(x, F(x))$ of the SBoxes) involved in the algorithm. Indeed, the beginning and the end dates of the execution of these primitives are quite easy to localize in the power consumption curves. Consequently, when all the inputs (resp. all the outputs) of the primitives are masked with the same value, then an attacker can precisely isolate two consumption points manipulating the same masks and is therefore able to unmask a sensitive data. A straightforward solution to make this particular class of second order DPA difficult to perform in practice consists in re-generating the masks as often as possible. Even if such a solution does not ensure that the algorithm thwarts all kinds of second order DPA, it allows to counteract those among the most efficient ones.

For the reasons detailed above, we think that there is a practical interest to introduce an intermediate resistance level between the *first order DPA-resistance*, in which every first order DPA is counteracted, and the *second order DPA-resistance*, in which every second order DPA is also counteracted. We call this intermediate level, *first order DPA-resistance in the multi-mask protection mode*. In this new level, the pair of masks (r, s) is re-generated each time a calculation $F(x)$ must be protected (and thus many times per algorithm execution). Since the implementation of an algorithm perfectly thwarting second-order DPA requires a great timing and memory overhead and since an implementation thwarting only first order DPA does no longer provide enough security, we think that the *first order DPA-resistance in the multi-mask protection mode* nowadays offers the better security/efficiency trade-off.

Analyzing the security of an implementation of an SBox for the new resistance level is equivalent to study the first order DPA resistance of the SBox implementation. The difference appears when it comes to investigate the efficiency of the countermeasure. For instance, a technic efficient in the single-mask mode (*e.g.* the re-computation method) can become much more costly in the multi-mask mode.

In the next section, we present a simple SBox secure calculation method which resolves Problem 1 and we compare its performances with other generic methods in the multi-mask protection mode.

3 The New S-Box Secure Calculation Method

3.1 Our Proposal

Let x denote a sensitive variable, let r and s be an input mask and an output mask and let F denote an SBox function mapping $\mathbb{F}_2^n$ into $\mathbb{F}_2^m$. The core idea of our proposal is to compute $F((x \oplus r) \oplus a) \oplus s$ for every value a, storing the result in a register R_0 if a equals r and in a second register R_1 otherwise.

Let *compare* : $x, y \mapsto compare(x, y)$ be the function returning 0 if $x = y$ and 1 otherwise, we depict our proposal in the following algorithm.

Algorithm 1 Computation of a masked S-Box output from a masked input

INPUT: a masked value $\tilde{x} = x \oplus r$, an input mask r, an output mask s, a look-up table for F

OUTPUT: the masked S-Box output $F(x) \oplus s$

```
1. for $a = 0$ to $2^n - 1$ do
2.     $cmp \leftarrow compare(a, r)$
3.     $R_{cmp} \leftarrow F(\tilde{x} \oplus a) \oplus s$
4. return $R_0$
```

Remark 2. Many microprocessors implement the function *compare* by a single instruction. Thus, we will assume in the rest of the paper that this function is an elementary operation.

To verify the correctness of Algorithm 1, it can be checked that Step 3 performs the following operation:

$$\begin{cases} R_0 \leftarrow F(\tilde{x} \oplus a) \oplus s & \text{if } a = r \ , \\ R_1 \leftarrow F(\tilde{x} \oplus a) \oplus s & \text{otherwise } . \end{cases} \tag{1}$$

Hence, R_0 contains the value $F(\tilde{x} \oplus r) \oplus s = F(x) \oplus s$ when the loop is completed.

The security of the new method highly depends on the assumption that the leakage generated by a register transfer is the same whatever is the register. This assumption is commonly accepted and it is the security core of SPA countermeasures used to protect asymmetric cryptosystems (see for instance [9]).

As every variable manipulated during the execution of the algorithm is masked by a random value, it can be proven (in a similar way as done in Sect. 4) that it thwarts DPA in both Hamming Weight and Hamming Distance models. Nevertheless, Algorithm 1 has a potential security weakness because it involves dummy operations (*i.e.* operations which do not impact the value returned by the algorithm)[3]. This flaw could be exploited by an attacker who would disrupt the Step 3 operation (for instance by fault injection [2]) during a chosen loop iteration $a = a_0$. By checking if Algorithm 1 output is erroneous or not, the attacker would be able to detect when the mask r equals the known loop index a_0. Finally, for the power consumption measurements corresponding to the cases $r = a_0$, the attacker would be able to unmask $\tilde{x}$ and to perform a classical first order DPA.

To circumvent the flaw, dummy operations must be avoided. When F is *balanced* [4], *i.e.* when $\#F^{-1}(y)$ equals 2^{n-m} for every $y \in \mathbb{F}_2^m$, then we propose

[3] Attacks exploiting dummy operations have been mainly applied to attack SPA-resistant implementations of RSA (see [24] for an example of such attacks).

[4] For security reasons, functions F involved in cryptographic applications are always balanced.

in the following a slightly modified version of Algorithm 1 where both registers R_0 and R_1 are involved in the computation of the output value.

Algorithm 2 Computation of a masked S-Box output from a masked input

INPUT: a masked value $\tilde{x} = x \oplus r$, an input mask r, an output mask s, a look-up table for F

OUTPUT: the masked S-Box output $F(x) \oplus s$

```
1. $R_0 \leftarrow s$
2. $R_1 \leftarrow s$
3. for $a = 0$ to $2^n - 1$ do
4.     $cmp \leftarrow compare(a, r)$
5.     $R_{cmp} \leftarrow R_{cmp} \oplus F(\tilde{x} \oplus a)$
6. $cmp \leftarrow compare(R_0, R_1)$
7. return $R_0 \oplus (cmp \times R_1)$
```

According to (1), register R_0 contains the value $F(x) \oplus s$ at the end of the loop. Moreover it can be checked that the content of R_1 equals $s \oplus \bigoplus_{\substack{a \in \mathbb{F}_2^n \\ a \neq r}} F(\tilde{x} \oplus a)$. As F is balanced, the summation $\bigoplus_{\substack{a \in \mathbb{F}_2^n \\ a \neq r}} F(\tilde{x} \oplus a)$ equals $F(\tilde{x} \oplus r)$ that is $F(x)$ since we have $\tilde{x} = x \oplus r$. Indeed, the summation $\bigoplus_{a \in \mathbb{F}_2^n} F(x)$ can be rewritten $\bigoplus_{y \in \mathbb{F}_2^m} (\bigoplus_{a \in \mathbb{F}_2^n;\ F(a)=y} y)$. As F is assumed to be balanced, each term $\bigoplus_{a \in \mathbb{F}_2^n;\ F(a)=y} y$ corresponds to 2^{n-m} times the sum of the vector y with itself. Thus, each term $\bigoplus_{a \in \mathbb{F}_2^n;\ F(a)=y} y$ equals the null vector if $n > m$ and equals y if $n = m$. This implies that $\bigoplus_{a \in \mathbb{F}_2^n} F(a)$ equals 0 if $n > m$ and equals $\bigoplus_{y \in \mathbb{F}_2^m} y$ if $n = m$. Since the sum of all the elements of a space equals the zero vector, one deduces that $\bigoplus_{a \in \mathbb{F}_2^n} F(a)$ is also equal to 0 if n and m are equal. Consequently, when F is balanced, the equality $R_1 = s \oplus F(\tilde{x} \oplus r) = F(x) \oplus s$ holds at the end of the loop.

Finally, Step 6 aims at verifying that the contents of the two registers R_0 and R_1 are equal. Then, Step 7 returns either the expected result if no perturbation occurred or an erroneous result otherwise. This simple improvement of Algorithm 1 ensures that the attacker is no longer able to determine when the index a_0 of the targeted loop iteration equals the input mask r.

Algorithm 1 requires $2^n \times 3$ logical operations (2 x-or operations and 1 comparison per loop iteration) and 2^n memory transfers (1 table look-up per loop operation). For Algorithm 2, two assignments (Steps 1 and 2), one comparison (Step 6), one multiplication and one x-or operation (Step 7) are added.

Since changing the input and output masks at each execution of Algorithm 2 has no impact on its performances, our proposal is efficient in the multi-mask protection mode. Moreover, it is generic in the sense that it can be applied to any balanced SBox F without any assumption on the algebraic structure of F. In what follows, we compare the performances of our proposal with the ones of other generic methods.

3.2 Comparison With Other Generic Methods

We focus here on two well-known elementary masked SBox computation methods. The first one is the table re-computation method recalled in Sect. 2. The second one, that we call here the *global look-up table method*, uses a large look-up table addressed with the mask and the masked value.

Re-computation method. Let T denote a 2^n bytes table allocated in RAM[5]. When the block cipher algorithm is protected in the multi-mask protection mode, a new pair of input/output masks is generated each time an SBox output is computed and the following sequence of operations is performed:

```
1. for $x = 0$ to $2^n - 1$ do
2.     $T[x] \leftarrow F(x \oplus r) \oplus s$
3. return $T[\tilde{x}]$
```

This algorithm requires $2^n \times 2$ logical operations (2 x-or operations per loop iteration) and $2^n \times 2 + 1$ memory transfers (1 read operation and 1 write operation per loop iteration and one access to the re-computed table). The computational cost of the table re-computation method is approximatively the same as for Algorithm 2. However, it also requires the allocation of 2^n bytes of RAM which can be problematic in a low resource context, especially when several SBoxes need to be protected.

Global look-up table method. Let $T^\star$ denote the look-up table associated to the function $(x, y) \mapsto F(x \oplus y) \oplus y$. To compute $F(x) \oplus r$ from $x \oplus r$ and r, the global look-up table method performs a single operation: the table look-up $T^\star[\tilde{x}, r]$. Its timing performances are ideal since it requires only one memory transfer. However, the size 2^{2n} of the look-up table $T^\star$ makes an application of the method difficult in a low resource context. For instance, if n is greater than or equal to 7, the amount of ROM required is definitively too great (at least 16 KB!). When n is lower than or equal to 6, the feasibility of the method depends on the amount of ROM of the device and on the number of different SBoxes which must be protected. The method can become interesting when it comes to protect SBoxes mapping $\mathbb{F}_2^4$ into itself (as it is the case for FOX [10] where three such SBoxes are involved) or when the SBox calculus can be performed in spaces of dimensions lower than or equal to 4 (as it is the case for the AES SBox - see Appendix A -).

From a security point of view, the global look-up table method has a flaw since it manipulates the mask r and the masked data $\tilde{x}$ at the same time. Indeed $\tilde{x}$ and r are concatenated to address the look-up table $T^\star$ and thus, the value $\tilde{x}||r$

[5] To make the description easier, we assume that every element of $\mathbb{F}_2^m$ is stored on one byte.

is transferred through the bus. Since the variables $\tilde{x}||r$ and x are statistically dependent, the leakage on $\tilde{x}||r$ is potentially exploitable by a first order DPA.

Table 1 summarizes the costs of the three previously considered methods according to the number of register logical operations (RLO), the number of memory transfers (MT), the memory size (bytes in RAM) and the code size (bytes in ROM).

Method	Masking Mode	Pre-computation	SBox calculation	RAM	ROM
Table recomp.	Single-masking	$2^{n+1}MT + 2^{n+1}RLO$	$1MT$	2^n	2^n
Table recomp.	Multi-masking	0	$(2^{n+1}+1)MT + 2^{n+1}RLO$	2^n	2^n
Global LUT	Multi-masking	0	$1MT$	0	2^{2n}
Algo. 2	Multi-masking	0	$2^n MT + (3 \times 2^n + 5)RLO$	0	2^n

Table 1. Comparison of methods solving Problem 1 for the bitwise addition.

4 Security Analysis

To study the security of our proposal we will use some basic notions of information Theory. We recall them in the next section.

4.1 Preliminaries

We use the calligraphic letters, like $\mathcal{X}$, to denote finite sets. The corresponding large letter X is then used to denote a random variable over $\mathcal{X}$, while the lowercase letter x - a particular element from $\mathcal{X}$. The probability of the event $(X = x)$ is denoted $\mathrm{P}[X = x]$. The *entropy* $H(X)$ of a random variable X aims at measuring the amount of information provided by an observation of X and satisfies $H(X) = -\sum_{x\in\mathcal{X}} \mathrm{P}[X = x]\log(\mathrm{P}[X = x])$. The *conditional entropy* of X given Y, denoted by $H(X|Y)$, equals $-\sum_{y\in\mathcal{Y}} \mathrm{P}[Y = y]\sum_{x\in\mathcal{X}} P[X = x|Y = y]\log(P[X = x|Y = y])$. To quantify the amount of information that Y reveals about X, the notion of *mutual information* is usually involved. The mutual information of X and Y is the value $\mathcal{I}(X, Y)$ defined by $\mathcal{I}(X, Y) = H(X) - H(X|Y)$. The random variables X and Y are *independent* if and only if $\mathcal{I}(X, Y)$ equals 0. Moreover, the mutual information is always positive or null and it satisfies the following property.

Property 1. Let X and Y be two random variables respectively defined over $\mathcal{X}$ and $\mathcal{Y}$. For every function f defined over $\mathcal{Y}$, we have $\mathcal{I}(X, f(Y)) \leq \mathcal{I}(X, Y)$.

For our security analysis, we shall also use the following proposition.

Proposition 1. *Let X and Y be two random variables defined over $\mathcal{X}$ and let Z be a random variable defined over $\mathcal{Z}$. If Z is mutually independent of X and Y and has a uniform distribution over $\mathcal{Z}$, then for every measurable function f defined from $\mathcal{X}^2$ into $\mathcal{Z}$, we have $\mathcal{I}(X, Z \oplus f(X, Y)) = 0$.*

As a consequence of Proposition 1, we have $\mathcal{I}(X, Z \oplus X) = 0$ when X and Z satisfy the conditions of Proposition 1.

4.2 Evaluation Methodology

To evaluate the security of our proposal, we follow the outlines of the methodology depicted in [20]. This methodology holds in five steps: specify the target implementation, specify the target secret, define the adversary model, evaluate the information leakage and define a metric to evaluate the security.

The target implementation is Algorithm 2 running on a smart card.

The target secret is the un-masked value x corresponding to the masked input $\tilde{x}$ of Algorithm 2.

The adversary model. We assume that the attacker can query the targeted cryptographic primitive with an arbitrary number of plaintexts and obtain the corresponding physical observations, but cannot choose its queries in function of the previously obtained observations (such a model is called *non-adaptive known plaintext model* in [20]). We also assume that the attacker has access to the power consumption and electromagnetic emanations of the device and applies a first order DPA attack but is not able to perform HODPA.

The effectiveness of the *prediction* made by the adversary is strongly related to the amount of information provided by the physical observations. In our analysis, we assume that the attacker knows how information leaks from the device and straightforwardly makes its prediction based on the leakage model (such a prediction is called *device profiled prediction* in [20]). Moreover, the physical observations are assumed to be perfect *i.e.* matching exactly the leakage model (which is a very favorable situation from the attacker's viewpoint).

For our analysis, we choose to consider a general leakage model that can be used for both power consumption or electromagnetic emanations.

The leakage model. Different models coexist to quantify the leakage of CMOS circuits with respect to the data handled but two of them are predominantly used: the *Hamming Weight* model (where the leakage is related to the Hamming Weight of the data handled) and the *Hamming Distance* model (where the leakage is related to the Hamming Distance between the previous and the current data handled in a register or transmitted through a bus - see [3]). In the Hamming Distance model, the leakage of the bit-transitions $0 \to 1$ and $1 \to 0$ are assumed to be equal, which makes the leakage analysis much simpler. However this assumption, which is adequate when trying to attack an unprotected implementation, is not relevant to model a strong opponent against a secure implementation. Indeed, in practice CMOS gates leak differently when charging or discharging the load capacitance (especially in the case of electromagnetic emanations [17]). Hence, as mentioned in [17,20], a more accurate leakage model must be defined to model an attacker who is able to observe these differences.

Definition 1. *[17] In the Hamming Distance Extended (HDE) model, the leakage $\mathcal{L}_{HDE}(s,y)$ related to a variable y that replaces an initial state s, satisfies*

$$\mathcal{L}_{HDE}(s,y) = N_{0\to 1}(s,y) \times P_{0\to 1} + N_{1\to 0}(s,y) \times P_{1\to 0} + \beta \ , \tag{2}$$

where $N_{0\to 1}(s,y)$ (resp. $N_{1\to 0}(s,y)$) denotes the number of transitions $0 \to 1$ (resp. $1 \to 0$) from s to y, $P_{0\to 1}$ (resp.$P_{1\to 0}$) denotes the average energy consumed by a transition $0 \to 1$ (resp. $1 \to 0$) and where β denotes some noise.

Denoting by δ the normalized difference $\frac{P_{0\to 1}-P_{1\to 0}}{P_{0\to 1}}$ and by ε the leakage $P_{0\to 1}$, we have $P_{1\to 0} = \varepsilon(1-\delta)$ and Relation (2) can be rewritten:

$$\mathcal{L}_{HDE}(s,y) = \varepsilon\left(1-\frac{\delta}{2}\right)\mathrm{HW}(s\oplus y) + \varepsilon\,\frac{\delta}{2}\left(\mathrm{HW}(y)-\mathrm{HW}(s)\right) + \beta \ . \tag{3}$$

From Relation (3), it can be verified that the HDE model includes the Hamming Weight (HW) and the Hamming Distance (HD) models. Indeed, when there is no difference between the transitions $0 \to 1$ and $1 \to 0$ (*i.e.* when $\delta = 0$), we have $\mathcal{L}_{HDE}(s,y) = \varepsilon HW(s\oplus y) + \beta$ and the HDE model is equivalent to the HD model. On the other hand, when the initial state s is constant, equal to 0, then we have $\mathcal{L}_{HDE}(s,y) = \varepsilon HW(y) + \beta$ and the HDE model is equivalent to the HW model. The HDE model is also appropriate to quantify electro-magnetic emanations leaking in the *signed distance model* which assumes that $P_{1\to 0}$ equals $-P_{0\to 1}$ [17]. In this case, we have $\delta = 2$ and the leakage satisfies $\mathcal{L}_{HDE}(s,y) = \varepsilon(HW(y) - HW(s)) + \beta$.

Once the behavior of the device and the attacker capacities are modeled, a method can be deduced from Standaert *et al.* [20] to prove the security of a countermeasure.

Evaluation of the security. Let us denote by X, Y and IS the random variables respectively corresponding to the sensitive data targeted by the attacker, the data manipulated at the date of a leakage and the initial state replaced by Y. In the theoretical model depicted by the four steps above, it has been proved in [18] and [20] that a first order DPA does not succeed in extracting information about X if and only if X and $\mathcal{L}_{HDE}(IS,Y)$ are independent for every pair (IS,Y) appearing during the execution of the algorithm.

In the following section, we evaluate $\mathcal{I}(X,\mathcal{L}_{HDE}(IS,Y))$ for every pair (IS,Y) appearing during the execution of Algorithm 2.

4.3 Proof of Security

If IS is an operation code, a constant memory address or a system variable which is independent of the intermediate results of the algorithm, then Property

1 and the positivity of the mutual information imply the following inequality:

$$0 \leq \mathcal{I}(X, \mathcal{L}_{HDE}(IS, Y)) \leq \mathcal{I}(X, Y) \ . \tag{4}$$

In such a case, proving $\mathcal{I}(X, Y) = 0$ is sufficient to prove $\mathcal{I}(X, \mathcal{L}_{HDE}(IS, Y)) = 0$.

If IS corresponds to an intermediate result of the algorithm, then Property 1 and the positivity of the mutual information imply the following inequality:

$$0 \leq \mathcal{I}(X, \mathcal{L}_{HDE}(IS, Y)) \leq \mathcal{I}(X, (IS, Y)) \ , \tag{5}$$

where (IS, Y) denotes the random variable that has the joint distribution of IS and Y. In this case, proving that $\mathcal{I}(X, (IS, Y))$ equals 0 is sufficient to prove that $\mathcal{I}(X, \mathcal{L}_{HDE}(IS, Y))$ equals 0.

To show that $\mathcal{I}(X, \mathcal{L}_{HDE}(IS, Y))$ equals 0 for every pair (IS, Y) appearing during the execution of Algorithm 2, we decompose our security proof into two steps, depending on the nature of IS. In a first step, we show that for every intermediate variable Y manipulated by Algorithm 2, the mutual information $\mathcal{I}(X, Y)$ equals 0. According to Inequality (4), this will prove that X and $\mathcal{L}_{HDE}(IS, Y)$ are independent when IS is assumed to be an operation code, a constant memory address or a system variable: we shall say in this case that there is no *variables leakage*. In a second time, we show that for every transition occurring between two intermediate results Y_1 and Y_2, the mutual information $\mathcal{I}(X, (Y_1, Y_2))$ equals 0. According to Inequality (5), this will prove that X and $\mathcal{L}_{HDE}(IS, Y)$ are independent when IS corresponds to an intermediate result of the algorithm: we shall say in this case that there is no *transitions leakage*.

Variables leakage. We decompose Algorithm 2 into several elementary operations each manipulating an intermediate result computed from the sensitive variable X, the input mask R and the output mask S. Since the random variables R and S correspond to randomly generated values, we can assume that they have a uniform distribution and that X, R and S are mutually independent.

Let $sum_a(X, R)$ denotes the sum $\bigoplus_{\substack{j=0 \\ j \neq R}}^{a} F(X \oplus R \oplus j)$ and let tmp denote the register used to store the intermediate results at Step 5. Table 2 lists the intermediate values that occur during an execution of Algorithm 2.

As R and S have a uniform distribution and as X, R and S are mutually independent, one straightforwardly deduces from Proposition 1 that all the intermediate results listed in Table 2 are independent of X.

Transitions leakage. We consider hereafter the transitions between intermediate results that occur either on the bus or in the registers R_0, R_1, cmp and tmp. For the bus, we consider transitions that appear when memory addresses and data transit either on the same bus (here denoted *S-BUS* for single bus) or on different bus (an address bus denoted *A-BUS* and a data bus denoted *D-BUS*).

Let ad_F denote the memory address of the look-up table F. In Table 3, we list the successive bus or register transitions occurring during an execution of Algorithm 2. We only list the transitions involving the sensitive data X.

Step	Instruction	Intermediate results
5.	$tmp \leftarrow \tilde{x}$	$X \oplus R$
	$tmp \leftarrow tmp \oplus a$	$X \oplus R \oplus a$
	$tmp \leftarrow F(tmp)$	$F(X \oplus R \oplus a)$
	$R_{cmp} \leftarrow R_{cmp} \oplus tmp$	$S \oplus \begin{cases} 0 & \text{if } R = a \\ sum_{a-1}(X,R) & \text{otherwise} \end{cases}$
		$S \oplus \begin{cases} F(X) & \text{if } R = a \\ sum_{a}(X,R) & \text{otherwise} \end{cases}$
6.	$cmp \leftarrow compare(R_0, R_1)$	$F(X) \oplus S$
7.	**return** $R_0 \oplus (cmp \times R_1)$	$F(X) \oplus S$

Table 2. Intermediate results manipulated during Algorithm 2.

Step	Operation	Target	Initial State IS	New State Y
5	$tmp \leftarrow \tilde{x}$	tmp	$F(X \oplus R \oplus (a-1))$	$X \oplus R$
5	$tmp \leftarrow tmp \oplus a$	tmp	$X \oplus R$	$X \oplus R \oplus a$
5	$tmp \leftarrow F(tmp)$	A-BUS	$ad_F + (X \oplus R \oplus (a-1))$	$ad_F + (X \oplus R \oplus a)$
5	$tmp \leftarrow F(tmp)$	D-BUS	$F(X \oplus R \oplus (a-1))$	$F(X \oplus R \oplus a)$
5	$tmp \leftarrow F(tmp)$	S-BUS	$F(X \oplus R \oplus (a-1))$	$ad_F + (X \oplus R \oplus a)$
5	$tmp \leftarrow F(tmp)$	S-BUS	$ad_F + (X \oplus R \oplus a)$	$F(X \oplus R \oplus a)$
5	$tmp \leftarrow F(tmp)$	tmp	$X \oplus R \oplus a$	$F(X \oplus R \oplus a)$
5	$R_{cmp} \leftarrow R_{cmp} \oplus tmp$	R_{cmp}	$S \oplus \begin{cases} 0 & \text{if } R = a \\ sum_{a-1}(X,R) & \text{otherwise} \end{cases}$	$S \oplus \begin{cases} F(X) & \text{if } R = a \\ sum_{a}(X,R) & \text{otherwise} \end{cases}$
7	$R_1 \leftarrow R_1 \oplus cmp$	cmp	$F(X \oplus S)$	$F(X \oplus S)$

Table 3. Transitions between intermediate results occurring during Algorithm 2.

For all except the fifth row of Table 3, Proposition 1 and Property 1 straightforwardly imply that $\mathcal{I}(X,(IS,Y))$ equals 0. For the fifth row, which corresponds to the update of R_{cmp}, let us study (IS,Y) for $R = a$ and $R \neq a$. If R equals a, then (IS,Y) can be written $(S, S \oplus F(X \oplus R \oplus a))$ where S is uniformly distributed over $\mathbb{F}_2^m$ and is independent of the pair (X,R). If R differs from a, then (IS,Y) can be written $(S \oplus sum_{a-1}(X,R), S \oplus sum_a(X,R))$ that is $(S', S' \oplus F(X \oplus R \oplus a))$ after denoting $S \oplus sum_{a-1}(X,R)$ by S'. It can be verified that S' has a uniform distribution over $\mathbb{F}_2^m$. Thus, due to Proposition 1, S' is independent of (X,R). One deduces that (IS,Y) is equivalent to a random variable $(U, U \oplus F(X \oplus R \oplus a))$ where U is uniformly distributed over $\mathbb{F}_2^m$ and independent of the pair (X,R). Then, from Property 1, we get $\mathcal{I}(X,(IS,Y)) \leq \mathcal{I}(X,(U, X \oplus R))$ and Proposition 1 implies that $\mathcal{I}(X,(IS,Y))$ equals 0.

We showed in this section that the sensitive variable X is independent of every intermediate result Y and every transition $IS \rightarrow Y$ that occurs during the execution of Algorithm 2. As argued in Sect. 4.2, this implies that there is no mutual information between the sensitive variable and the instantaneous power consumption leakages. We can therefore conclude that Algorithm 2 is secure against first order DPA in the HDE model.

5 Conclusion

In this paper we have presented a new masking scheme for software SBox implementations that requires no RAM allocation. Since our method does not rely on specific SBox properties, it is generic and can thus be applied to protect any symmetric cryptosystem. We have argued that a first order DPA countermeasure must be efficient not only in the single-mask protection mode but also in the multi-mask mode. In this mode, we have shown that our countermeasure is as efficient as the other classical generic methods and does not require RAM allocation. We have evaluated our solution within the framework recently introduced by Standaert *et al.* in [17, 20], proving its security against first order DPA under realistic assumptions about the attacker and the device behaviors. Finally, we have applied our method to AES and we have compared its efficiency with other secure implementations. Based on this analysis, we think that the timing and memory overhead of our countermeasure are suitable for practical implementations of the AES algorithm when a protection in multi-mask protection mode is required.

Acknowledgements

We would like to thank Christophe Giraud and Emmanuelle Dottax for their fruitful comments and suggestions on this paper.

References

1. M.-L. Akkar and C. Giraud. An Implementation of DES and AES, Secure against Some Attacks. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, vol. 2162 of *LNCS*, pages 309–318.
2. D. Boneh, R. DeMillo, and R. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. *Journal of Cryptology*, 14(2):101–119, 2001.
3. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, vol. 3156 of *LNCS*, pages 16–29. Springer, 2004.
4. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology – CRYPTO '99*, vol. 1666 of *LNCS*, pages 398–412. Springer, 1999.
5. J. Golić and C. Tymen. Multiplicative Masking and Power Analysis of AES. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, vol. 2523 of *LNCS*, pages 198–212. Springer, 2002.
6. L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In *Cryptographic Hardware and Embedded Systems – CHES '99*, vol. 1717 of *LNCS*, pages 158–172. Springer, 1999.
7. S. Gueron, O. Parzanchevsky, and O. Zuk. Masked Inversion in GF(2^n) Using Mixed Field Representations and its Efficient Implementation for AES. In *Embedded Cryptographic Hardware: Methodologies and Architectures*, pages 213–228. Nova Science Publishers, 2004.

8. M. Joye, P. Paillier, and B. Schoenmakers. On Second-Order Differential Power Analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, vol. 3659 of *LNCS*, pages 293–308. Springer, 2005.
9. M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In *Cryptographic Hardware and Embedded Systems – CHES 2002*, vol. 2523 of *LNCS*, pages 291–302. Springer, 2002.
10. P. Junod and S. Vaudenay. FOX: a new family of block ciphers. In *Selected Areas in Cryptography – SAC 2004*, vol. 3357 of *LNCS*, pages 114–129. Springer, 2004.
11. T. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In *Fast Software Encryption – FSE 2000*, vol. 1978 of *LNCS*, pages 150–164. Springer, 2000.
12. T. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant software. In *Cryptographic Hardware and Embedded Systems – CHES 2000*, vol. 1965 of *LNCS*, pages 238–251. Springer, 2000.
13. E. Oswald, S. Mangard, C. Herbst, and S. Tillich. Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In *Topics in Cryptology – CT-RSA 2006*, vol. 3860 of *LNCS*. Springer, 2006.
14. E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen. A Side-Channel Analysis Resistant Description of the AES S-box. In *Fast Software Encryption – FSE 2005*, vol. 3557 of *LNCS*, pages 413–423. Springer, 2005.
15. E. Oswald and K. Schramm. An Efficient Masking Scheme for AES Software Implementations. In *WISA 2005*, vol. 3786 of *LNCS*, pages 292–305. Springer, 2006.
16. E. Oswald, Stefan, and N. Pramstaller. Secure and Efficient Masking of AES – A Mission Impossible ? Cryptology ePrint Archive, Report 2004/134, 2004.
17. E. Peeters, F.-X. Standaert, and J.-J. Quisquater. Power and Electromagnetic Analysis: Improved Model, Consequences and Comparisons. In *Integration, the VLSI Journal*. Elsevier, Spring 2006. To appear.
18. E. Prouff, C. Giraud, and S. Aumonier. Provably Secure S-Box Implementation Based on Fourier Transform. In *Cryptographic Hardware and Embedded Systems – CHES 2006*, vol. 4249 of *LNCS*, pages 216–230. Springer, 2006.
19. A. Rudra, P. K. Bubey, C. S. Jutla, V. Kumar, J. Rao, and P. Rohatgi. Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. In *Cryptographic Hardware and Embedded Systems – CHES 2001*, vol. 2162 of *LNCS*, pages 171–184. Springer, 2001.
20. F.-X. Standaert, T. G. Malkin, and M. Yung. Side-Channel Resistant Ciphers: Model, Analysis and Design. Cryptology ePrint Archive, Report 2006/139, 2006.
21. E. Trichina. Combinatorial Logic Design for AES SubByte Transformation on Masked Data. Cryptology ePrint Archive, Report 2003/236, 2003.
22. E. Trichina and L. Korkishko. Secure and Efficient AES Software Implementation for Smart Cards. In *Information Security Applications, 5th International Workshop, WISA 2004*, vol. 3325 of *LNCS*, pages 425–439. Springer, 2004.
23. J. Waddle and D. Wagner. Toward Efficient Second-order Power Analysis. In *Cryptographic Hardware and Embedded Systems – CHES 2004*, vol. 3156 of *LNCS*, pages 1–15. Springer, 2004.
24. S.-M. Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

A Application to AES

We recall that the AES SBox is composed of two parts: a non-linear function and an affine mapping. In the following we focus on the non-linear part, which will be denoted here by F. Let $p(x)$ denotes the irreducible polynomial $x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1 \in \mathbb{F}_2[x]$. The function F is defined in $\mathbb{F}_2[x]/p(x)$ by $F(a) = 0$ if $a = 0$ and by $F(a) = a^{-1}$ otherwise.

At first, we applied the method depicted in Algorithm 2 for $n = 8$ to protect the SBox access of the AES algorithm. We implemented the solution on a classical 8051 chip running at 8 Mhz and we studied the performances of the implementation. Clearly, the timing of the resulting AES algorithm was not interesting (around 115 ms and $1,8$ KB of ROM) compared to 5ms for an implementation without countermeasures.

Secondly, we represented $\mathbb{F}_{2^8}$ has an extension of $\mathbb{F}_{2^4}$, allowing us to perform the computations in $\mathbb{F}_{2^4}$ instead of $\mathbb{F}_{2^8}$ (such a method is usually called *composite field approach*). We chose the two irreducible polynomials $p'(x) = x^2 + x + \{e\}$ and $p''(x) = x^4 + x + 1$ in $\mathbb{F}_4[x]$ and $\mathbb{F}_2[x]$ respectively and we denoted by *map* the field isomorphism which takes an element a of $\mathbb{F}_2[x]/p(x)$ as input and outputs the pair $(a_h, a_l) \in (\mathbb{F}_2[x]/p''(x))^2$ corresponding to the coefficients of the linear polynomial $(a_h x + a_l) \in \mathbb{F}_{2^4}[x]/p'(x)$. Moreover, we denoted by $Inv_{\mathbb{F}_{2^4}}$ the function which corresponds to the inverse function over $\mathbb{F}_2[x]/(x^4 + x + 1)\backslash\{0\}$ and which maps 0 into itself. In the following, we depict the different steps of our computation:

Algorithm 3 Inversion of a masked element $\widetilde{a} = a \oplus m_a$ in $\mathbb{F}_{2^8}$

```
INPUT: $(\widetilde{a} = a \oplus m_a, m_a) \in {\mathbb{F}_{2^8}}^2$
OUTPUT: $(\widetilde{a^{-1}} = a^{-1} \oplus m'_a, m'_a)$

1. Pick up three 4-bit random $m_d$, $m'_h$ and $m'_l$
2. $(m_h, m_l) \in \mathbb{F}_{2^4}^2 \leftarrow map(m_a)$
3. $(\widetilde{a_h}, \widetilde{a_l}) \in \mathbb{F}_{2^4}^2 \leftarrow map(\widetilde{a})$    $[(\widetilde{a_h}, \widetilde{a_l}) = (a_h \oplus m_h, a_l \oplus m_l)]$
4. $\widetilde{d} \leftarrow \widetilde{a_h}^2 \otimes \{e\} \oplus \widetilde{a_h} \otimes \widetilde{a_l} \oplus \widetilde{a_l}^2 \oplus m_d \oplus \widetilde{a_h} \otimes m_l$    $[\widetilde{d} = d \oplus m_d]$
       $\oplus \widetilde{a_l} \otimes m_h \oplus m_h^2 \otimes \{e\} \oplus m_l^2 \oplus m_h \otimes m_l$
5. $\widetilde{d^{-1}} \leftarrow$ Algorithm 2$(\widetilde{d}, m_d, m_{d^{-1}}, Inv_{\mathbb{F}_{2^4}})$    $[\widetilde{d^{-1}} = d^{-1} \oplus m_{d^{-1}}]$
6. $\widetilde{a'_h} \leftarrow \widetilde{a_h} \otimes \widetilde{d^{-1}} \oplus m'_h \oplus m_h \otimes \widetilde{d^{-1}} \oplus m_{d^{-1}} \otimes \widetilde{a_h} \oplus m_{d^{-1}} \otimes m_h$    $[\widetilde{a'_h} = a'_h \oplus m'_h]$
7. $\widetilde{a'_l} \leftarrow \widetilde{a_l} \otimes \widetilde{d^{-1}} \oplus m'_l \oplus \widetilde{a'_h} \oplus \widetilde{d^{-1}} \otimes m_l \oplus \widetilde{a_l} \otimes m_{d^{-1}} \oplus m'_h \oplus m_l \otimes m_{d^{-1}}$ $[\widetilde{a'_l} = a'_l \oplus m'_l]$
8. $m'_a \leftarrow map^{-1}(m'_h, m'_l)$
9. $\widetilde{a^{-1}} \leftarrow map^{-1}(\widetilde{a'_h}, \widetilde{a'_l})$    $[\widetilde{a^{-1}} = a^{-1} \oplus m'_a]$
10. **return** $(\widetilde{a^{-1}}, m'_a)$
```

For this version, the timing of the resulting AES algorithm are very interesting and the input and output masks can be changed at each execution of the algorithm.

In the following table, we have listed the timing/memory performances of our proposal and the ones of other methods proposed in the Literature. As the performances have been measured for a particular implementation on a particular architecture, the table above does not aim at arguing that a method is better than another but aims at enlightening the main particularities (timing performances and ROM/RAM requirements) of each method.

Method	Timings (ms)	RAM (bytes)	ROM (bytes)	Multi-masking
Straightforward implementation	5	0	1150	-
Re-computation Methods in the single-mask mode				
Re-computation Method in $\mathbb{F}_{2^8}$ [11]	$\times 1.42$	+256	+49%	not allowed
Re-computation Method in $\mathbb{F}_{2^4}$ [11]	$\times 2.60$	+16	+150%	not allowed
Re-computation Methods in the multi-mask mode				
Re-computation Method in $\mathbb{F}_{2^8}$ [11]	$\times 50, 60$	+256	+49%	allowed
Re-computation Method in $\mathbb{F}_{2^4}$ [11]	$\times 5.86$	+16	+150%	allowed
Secure SBox computation methods based on the composite field approach				
Oswald *et al.* [14, 16]	$\times 5.20$	0	+173%	allowed
This paper (Algo. 3)	$\times 5.30$	0	+150%	allowed
Prouff *et al.* [18]	$\times 6.40$	0	+147%	allowed
Methods with security under discussion				
Oswald and Schramm [15]	$\times 2.40$	0	+200%	allowed
Trichina *et al.* [22]	$\times 4.20$	+256	+165%	not allowed

Table 4. Comparison of several methods to protect AES against DPA.

The AES implementations listed above only differ in their approaches to protect the SBox access. The linear steps of the AES have been implemented in the same way and the internal sensitive data have been masked by bitwise addition of a random value. We chose to protect only rounds 1 to 3 and 8 to 10, assuming that the diffusion properties of the AES algorithm make DPA attacks impossible to mount on inner rounds 4 to 7 (this implies that the SBox calculations made in rounds 4 to 7 are performed by simply accessing the table representation of F which is stored in ROM).

In the single mask mode, the re-computation method in $\mathbb{F}_{2^8}$ has the best timing performances but at least 256 bytes of RAM must be allocated to store the re-computed SBox table. As RAM is a sensitive resource in the area of embedded devices, we implemented a second version which follows the outlines of the composite field approach and then applies the re-computation method in $\mathbb{F}_{2^4}$. Because only 16 bytes of RAM are required to store the table re-computed from the $Inv_{\mathbb{F}_{2^4}}$ function, the new implementation requires much less RAM than the version in $\mathbb{F}_{2^8}$ and the timing performances are suitable for practical applications.

As the multi-mask protection mode offers better security with respect to power analysis attacks [5,15], we tested the re-computation method in this mode. As expected, the re-computation method in $\mathbb{F}_{2^8}$ no longer gives full satisfaction in this mode (requiring 253 ms for one AES execution). The timing performances of the re-computation method in $\mathbb{F}_{2^4}$ are acceptable in the multi-mask protection mode, however they are close to (and even slightly greater than) the performances of the SBox secure calculation methods. Moreover, 16 bytes of RAM are required.

The SBox secure calculation methods of Oswald *et al.*, Prouff *et al.* and our proposed approach only differ in the ways of securely computing the value $d^{-1} \oplus m_{d^{-1}}$ from $\tilde{d}$, m_d and $m_{d^{-1}}$ (*i.e.* to securely perform the fifth Step of Algorithm 3):

- In [14,16], the inversion is performed by going down to $\mathbb{F}_4$ and its complexity approximatively equals the one of Algorithm 3 excluding the 5th Step which is replaced by a square operation (since the inversion operation in $\mathbb{F}_4$ is equivalent to squaring). For our implementation of [14, 16], the number of cycles required for the fifth step is 267.
- In [18], the inversion is essentially performed by computing a Fourier transform on $\mathbb{F}_2^4$. For our implementation of [18], the number of cycles required for the fifth step is 468.
- For the new solution presented here, the fifth step essentially corresponds to the computation of $y^{-1} \oplus m_{d^{-1}}$ for every $y \in \mathbb{F}_2^4$. For our implementation, the number of cycles required by the fifth step is 270.

The three methods can be used in the multi-mask protection mode without decreasing the performances of the implementation and they offer approximatively the same (good) level of security related to first order DPA attacks. The execution timings of AES implementations based on our proposal or on Oswald *et al.*'s method are very close and their RAM requirements are almost equal. The additional time required by the Prouff *et al.* method is slightly greater, however the code seems to be shorter (2844 bytes of ROM *versus* 2881 and 3144 bytes of ROM for our method and the Oswald *et al.*'s method).

The methods proposed by Trichina [22] and Oswald-Shramm [15] have good timing performances but are not perfectly resistant to first order DPA attacks.

- In the method of Trichina *et al.*, a *primitive element* of $\mathbb{F}_{2^8}$ is computed and every non-zero element of $\mathbb{F}_{2^8}$ is expressed as a power of that element. To resolve Problem 1 for the bitwise operation, Trichina *et al.* use pre-computed discrete logarithm and exponentiation tables to realize the SBox operation. As argued in [15], the method has a faulty behavior when some intermediate values are null and to correct the method without introducing a flaw with respect to first order attacks seems to be an issue.
- The method proposed by Oswald and Shramm offers the best timing performances. As for Algorithm 3, it is based on the composite field approach but steps 4 to 7 are replaced by a sequence of table look-ups and bitwise additions. The table look-ups have been render resistant to first order DPA attacks by applying the global look-up table method (which is recalled in Sect. 3). For example, the computation of $d^{-1} \oplus m_{d^{-1}}$ (Step 5) is performed by accessing the table T_{inv} associated to the function $((d \oplus m_d), m_d) \in (\mathbb{F}_{2^4})^2 \mapsto (d^{-1} \oplus m_d) \in \mathbb{F}_{2^4}$. As argued in Sect. 3, the global look-up table method has a flaw with respect to first order DPA attacks. Indeed, to address the T_{inv} table the value $(d \oplus m_d)||m_d$ is manipulated, which results in a power consumption that leaks information on the sensitive value d. For instance, it can be checked that $\mathcal{I}(d, H((d \oplus m_d)||m_d))$ is not null, which

results in an information leakage in the Hamming Weight model. Moreover, the input and output masks being equal, this method has also a potential flaw in the Hamming Distance model. Indeed, if a transition occurs between the index $(d \oplus m_d)||m_d$ and the value $d^{-1} \oplus m_d$ accessed in the look-up table (which is very likely in a single bus architecture), the mask m_d is canceled and information leaks about d and/or d^{-1}.