Higher-order Masking and Shuffling for Software Implementations of Block Ciphers

Matthieu Rivain[1,2], Emmanuel Prouff[1], and Julien Doget[1,3,4]

[1] Oberthur Technologies, France
[2] University of Luxembourg, Luxembourg
[3] Université catholique de Louvain, Belgium
[4] University of Paris 8, France
{m.rivain,e.prouff,j.doget}@oberthur.com

Abstract. Differential Power Analysis (DPA) is a powerful side channel key recovery attack that efficiently breaks block ciphers implementations. In software, two main techniques are usually applied to thwart them: masking and operations shuffling. To benefit from the advantages of the two techniques, recent works have proposed to combine them. However, the schemes which have been designed until now only provide limited resistance levels and some advanced DPA attacks have turned out to break them. In this paper, we investigate the combination of masking and shuffling. We moreover extend the approach with the use of higher-order masking and we show that it enables to significantly improve the security level of such a scheme. We first conduct a theoretical analysis in which the efficiency of advanced DPA attacks targeting masking and shuffling is quantified. Based on this analysis, we design a generic scheme combining higher-order masking and shuffling. This scheme is scalable and its security parameters can be chosen according to any desired resistance level. As an illustration, we apply it to protect a software implementation of AES for which we give several security/efficiency trade-offs.

1 Introduction

Side Channel Analysis (SCA in short) exploits information that leaks from physical implementations of cryptographic algorithms. This leakage (*e.g.* the power consumption or the electro-magnetic emanations) may indeed reveal information on the secret data manipulated by the implementation. Among SCA attacks, two classes may be distinguished. The set of so-called *Profiling SCA* corresponds to a powerful adversary who has a copy of the attacked device under control and who uses it to evaluate the distribution of the leakage according to the processed values. Once such an evaluation is obtained, a maximum likelihood approach is followed to recover the secret data manipulated by the attacked device. The second set of attacks is the set of so-called *Differential Power Analysis* (DPA) [11]. It corresponds to a more realistic (and much weaker) adversary than the one considered in Profiling SCA, since the adversary is only able to observe the device behavior and has no *a priori* knowledge of the implementation details.

This paper only deals with the set of DPA as it includes a great majority of the attacks encountered *e.g.* by the smart card industry.

Block ciphers implementations are especially vulnerable to DPA attacks and research efforts have been stepped up to specify implementation schemes counteracting them. For software implementations, one identifies two main approaches: masking [2, 7] and shuffling [8]. However, some advanced DPA techniques exist that defeat these countermeasures [3, 15]. A natural approach to improve the DPA resistance is to mix masking and shuffling [8, 25, 26]. This approach seems promising since it enables to get the best of the two techniques. However, the schemes that have been proposed so far [8, 26] only focus on first-order masking which prevents them from reaching high resistance levels. This is all the more serious that advanced DPA attacks have turned out to be quite efficient in breaking them [25, 26].

In this paper, we conduct an analysis to quantify the efficiency of an attack that targets either a masked implementation or a shuffled implementation or a masked-and-shuffled implementation. Based on this analysis, we design a new scheme combining higher-order masking and shuffling to protect software implementations of block ciphers. This scheme is scalable and its parameters can be specified to achieve any desired resistance level. We apply it to protect a software implementation of AES and we show how to choose the scheme parameters to achieve a given security level with the minimum overhead.

2 Preliminaries

2.1 Masking and Shuffling Countermeasures

To protect cryptographic implementations against DPA, one must reduce the amount of information that leaks on sensitive intermediate variables during the processing. A variable is said to be *sensitive* if it is a function of the plaintext and a guessable part of the secret key (that is not constant with respect to the latter).

To thwart DPA attacks, countermeasures try to make leakages as independent as possible of sensitive variables. Nowadays, two main approaches are followed to achieve such a purpose in software: the *masking* and the *shuffling*. We briefly recall hereafter the two techniques.

The core idea behind masking is to randomly split every sensitive variable X into $d+1$ shares M_0,..., M_d in such a way that the relation $M_0 \star ... \star M_d = X$ is satisfied for a group operation $\star$ (*e.g.* the x-or or the modular addition). Usually, M_1,..., M_d (called *the masks*) are randomly picked up and M_0 (called *the masked variable*) is processed to satisfy $M_0 \star ... \star M_d = X$. The parameter d is usually called *the masking order*. When carefully implemented (namely when all the shares are processed at different times), d^{th}-order masking perfectly withstands any DPA exploiting less than $d+1$ leakage signals simultaneously. Although attacks exploiting $d+1$ leakages are always theoretically possible, in practical settings their complexity grows exponentially with d [2]. The design of efficient

higher-order masking schemes for block ciphers is therefore of great interest. However, even for small d, dealing with the propagation of the masks through the underlying scheme is an issue. For linear operations, efficient and simple solutions exist that induce an acceptable overhead irrespective of d. Actually, the issue is to protect the non-linear S-boxes computations. In the particular case $d = 1$, a straightforward solution called the *table re-computation* exists (see for instance [1, 14]). Straightforward generalizations of the method to higher orders d do not provide security *versus* higher-order DPA. Indeed, whatever the number of masks, an attack targeting two different masked input/output is always possible (see for instance [17]). To bypass this flaw, [23] suggests to re-compute a new table before every S-box computation. This solution is very costly in terms of timings and [5] shows the feasibility of third-order attacks, so the scheme is only secure for $d < 3$. An alternative solution for $d = 2$ has been proposed in [21] but the timing overhead is of the same order.

Shuffling consists in spreading the signal containing information about a sensitive variable X over t different signals $S_1, \ldots, S_t$ leaking at different times. This way, if the spread is uniform, then for every i the probability that S_i corresponds to the manipulation of X is $\frac{1}{t}$. As a consequence, the signal-to-noise ratio of the instantaneous leakage on X is reduced by a factor of t (see Sect. 3.2 for details). Applying shuffling is straightforward and does not relate to the nature (linear or non-linear) of the layer to protect. Moreover, shuffling is usually significantly less costly than higher-order masking when applied to non-linear layers.

Since higher-order masking is expensive and since first-order masking can be defeated with quite reasonable efforts [17], a natural idea is to use shuffling together with first-order masking. A few schemes have already been proposed in the literature [8, 26]. In [8], an 8-bit implementation of AES is protected using first-order masking and shuffling. The work in [26] extends this scheme to a 32-bit implementation with the possible use of instructions set extension. Furthermore, [26] proposes some advanced DPA attacks on such schemes whose practicability is demonstrated in [25]. These works show that combining first-order masking with shuffling is definitely not enough to provide a strong security level. A possible improvement is to involve higher-order masking. This raises two issues. First, a way to combine higher-order masking with shuffling must be defined (especially for S-boxes computations). Secondly, the security of such a scheme should be quantifiable. It would indeed be of particular interest to have a lower bound on the resistance of the overall implementation by choosing *a priori* the appropriate trade-off between masking and shuffling orders. In the rest of the paper, we address those two issues.

2.2 Notations and Leakage Model

We use the calligraphic letters, like $\mathcal{X}$, to denote finite sets (*e.g.* $\mathbb{F}_2^n$). The corresponding capital letter X is used to denote a random variable over $\mathcal{X}$, while the lowercase letter x - a particular element from $\mathcal{X}$. The expectation of X is denoted by $\mathrm{E}[X]$, its variance by $\mathrm{Var}[X]$ and its standard deviation by $\sigma[X]$. The

correlation coefficient [27] between X and Y is denoted by $\rho[X, Y]$. It measures the linear interdependence between X and Y and is defined by:

$$\rho[X, Y] = \frac{\mathrm{Cov}[X, Y]}{\sigma[X]\,\sigma[Y]} \ , \tag{1}$$

where $\mathrm{Cov}[X, Y]$, called *covariance of X and Y*, equals $\mathrm{E}[(X - \mathrm{E}[X])(Y - \mathrm{E}[Y])]$ or $\mathrm{E}[XY] - \mathrm{E}[X]\,\mathrm{E}[Y]$ equivalently.

In the next sections, we investigate the security of the combination of masking and shuffling towards DPA. Our analysis is conducted in the Hamming weight leakage model that we formally define hereafter. This model is very common for the analysis of DPA attacks [9, 20, 26] and it has been practically validated several times [15, 17].

Definition 1 (Hamming weight model). *The leakage signal S_i produced by the processing of a variable D_i satisfies:*

$$S_i = \delta_i + \beta_i \cdot \mathrm{H}(D_i) + N_i \ , \tag{2}$$

where δ_i denotes a constant offset, β_i is a real value, $\mathrm{H}(\cdot)$ denotes the Hamming weight function and N_i denotes a noise with mean 0 and standard deviation σ.

When several leakage signals S_i's are jointly considered, we shall make three additional assumptions: (1) the constant β_i is the same for the different S_i's (without loss of generality, we consider $\beta_i = 1$), (2) noises N_i's are mutually independent and (3) the noise standard deviation σ is the same for the different N_i's.

3 Analysis of Advanced DPA Attacks Against Masking and Shuffling

Higher-order DPA attacks aim at recovering information on a sensitive variable X by considering several non-simultaneous leakage signals. Let us denote by $\mathbf{S}$ the multivariate random variable corresponding to those signals. The attack starts by converting $\mathbf{S}$ into an univariate random variable by applying it a function g. Then, a *prediction function* f is defined according to some assumptions on the device leakage model. Eventually, every guess $\hat{X}$ on X is checked by estimating the correlation coefficient between the combined leakage signal $g(\mathbf{S})$ and the so-called *prediction* $f(\hat{X})$.

As argued in several works (see for instance [12, 13, 20, 23]), the absolute value of the correlation coefficient $\rho[f(X), g(\mathbf{S})]$ (corresponding to the correct key guess) is a sound estimator of the efficiency of a correlation based DPA characterized by the pair of functions (f, g). In [13, 24], it is even shown that the number of leakage measurements required for the attack to succeed can be approximated by $c \cdot \rho[f(X), g(\mathbf{S})]^{-2}$ where c is a constant depending on the number of key guesses and the required success rate. In the following, we exhibit in the Hamming weight model (see Sect. 2.2) explicit formulae of this coefficient

for advanced DPA attacks where the sensitive variable is either (1) protected by (higher-order) masking, or (2) protected by shuffling or (3) protected with a combination of the two techniques.

3.1 Defeating Masking: Higher-order DPA

When d^{th}-order masking is used, any sensitive variable X is split into $d+1$ shares $X \oplus \mathbf{M}$, M_1, ..., M_d, where $\mathbf{M}$ denotes the sum $\bigoplus_i M_i$. In the following, we shall denote $X \oplus \mathbf{M}$ by M_0. The processing of each share M_i respectively results in a leakage signal S_i. Since the M_i's are assumed to be mutually independent, every tuple of d signals or less among the S_i's is independent of X. Thus, to recover information about X, the joint distribution of all the $d+1$ signals must be considered. Higher-order DPA consists in combining the $d+1$ leakage signals by the mean of a so-called *combining function* $\mathsf{C}(\cdot, \cdots, \cdot)$. This enables the construction of a signal that is correlated to the sensitive variable X.

Several combining functions have been proposed in the literature. Two of them are commonly used: the *product combining* [2] which consists in multiplying the different signals and the *absolute difference combining* [15] which computes the absolute value of the difference between two signals. As noted in [5, Sect. 1], the latter can be extended to higher orders by induction. Other combining functions have been proposed in [9,16]. In a recent paper [20], the different combining functions are compared for second-order DPA in the Hamming weight model. An improvement of the product combining called *normalized product combining* is proposed and it is shown to be more efficient than the other combining functions[5]. In this paper, we therefore consider the normalized product combining generalized to higher orders:

$$\mathsf{C}\left(S_0, S_1 \cdots, S_d\right) = \prod_{i=0}^{d} \left(S_i - \mathrm{E}\left[S_i\right]\right) \ . \tag{3}$$

We shall denote by $C_d(X)$ the combined leakage signal $\mathsf{C}\left(S_0, S_1 \cdots, S_d\right)$ where the S_i's correspond to the processing of the shares $X \oplus \mathbf{M}$, M_1, ..., M_d in the Hamming weight model. The following lemma gives the expectation of $C_d(X)$ given $X = x$ for every $x \in \mathbb{F}_2^n$. The proof is given in the extended version of this paper [22].

Lemma 1. *Let $x \in \mathbb{F}_2^n$, then the expectation of $C_d(x)$ satisfies:*

$$\mathrm{E}\left[C_d(x)\right] = \left(-\frac{1}{2}\right)^d \left(\mathrm{H}(x) - \frac{n}{2}\right) \ . \tag{4}$$

Lemma 1 shows that the expectation of $C_d(x)$ is an affine function of the Hamming weight of x. According to the analysis in [20], this implies that the

[5] This assertion is true while considering a noisy model. In a fully idealized model, other combining may provide better results (see [20]).

Hamming weight of X maximizes the correlation. For the reasons given in [20], this function can therefore be considered as an optimal prediction for $C_d(X)$. Hence, the HO-DPA we focus here consists in estimating the correlation between the Hamming weight of the target variable $\mathrm{H}(X)$ and the combined leakage $C_d(X)$. The next proposition provides the exact value of this correlation. The proof is given in the extended version of this paper [22].

Proposition 1. *Let X be a random variable uniformly distributed over $\mathbb{F}_2^n$. The correlation between $\mathrm{H}(X)$ and $C_d(X)$ satisfies:*

$$\rho\left[\mathrm{H}(X), C_d(X)\right] = (-1)^d \frac{\sqrt{n}}{\left(n + 4\sigma^2\right)^{\frac{d+1}{2}}} \ . \tag{5}$$

Notation. The correlation coefficient in (5) shall be referred as $\rho(n, d, \sigma)$.

3.2 Defeating Shuffling: Integrated DPA

When shuffling is used, the signal containing information about the sensitive variable X is randomly spread over t different signals S_1, ..., S_t. As a result, the correlation between the prediction and one of these signals is reduced by a factor t compared to the correlation without shuffling. In [3], an *integrated DPA attack* (also called *windowing attack*) is proposed for this issue. The principle is to add the t signals all together to obtain an *integrated signal*. The correlation is then computed between the prediction and the integrated signal. The resulting correlation is reduced by a factor $\sqrt{t}$ instead of t without integration. This is formalized in the next proposition.

Proposition 2. *Let $(S_i)_{1 \leqslant i \leqslant t}$ be t random variables identically distributed and mutually independent. Let Y denote a signal S_j's whose index j is a random variable uniformly distributed over $\{1, \cdots, t\}$. Let X be a random variable that is correlated to Y and that is independent of the remaining S_i's. For every measurable function f, the correlation between $f(X)$ and $S_1 + \cdots + S_t$ satisfies:*

$$\rho\left[f(X), S_1 + \cdots + S_t\right] = \frac{1}{\sqrt{t}} \rho\left[f(X), Y\right] \ . \tag{6}$$

Proof. On one hand we have $\mathrm{Cov}\left[f(X), S_1 + \cdots + S_t\right] = \mathrm{Cov}\left[f(X), Y\right]$ and on the other hand we have $\sigma\left[S_1 + \cdots + S_t\right] = \sqrt{t}\ \sigma\left[Y\right]$. Relation (6) straightforwardly follows. ◇

3.3 Defeating Combined Masking and Shuffling: Combined Higher-order and Integrated DPA

When masking is combined with shuffling, any sensitive variable X is split into $d+1$ shares $X \oplus \mathbf{M}$, M_1, ..., M_d whose manipulations are randomly spread over t different times yielding t different signals S_i. The $(d+1)$-tuple of signals indices corresponding to the shares hence ranges over a subset I of the set

of $(d+1)$-combinations from $\{1, \cdots, t\}$. This subset depends on how the shuffling is performed (*e.g.* the shares may be independently shuffled or shuffled all together).

To bypass such a countermeasure, an adversary may combine integrated and higher-order DPA techniques. The most pertinent way to perform such a combined attack is to design a so-called *combined-and-integrated* signal by summing all the possible combinations of $d+1$ signals among S_1, ..., S_t [25, 26]. That is, the combined-and-integrated signal, denoted $IC_{d,I}(X)$, is defined by:

$$IC_{d,I}(X) = \sum_{(i_0,\ldots,i_d)\in I} \mathsf{C}(S_{i_0}, \cdots, S_{i_d}) \ . \tag{7}$$

We show in the following proposition that the correlation coefficients of the combined attacks relate on $\rho(n, d, \sigma)$. The proof is given in the extended version of this paper [22].

Proposition 3. *Let X, M_1, ..., M_d be a family of $d+1$ n-bit random variables uniformly distributed and mutually independent. Let I be a set of $(d+1)$-combinations from $\{1, \cdots, t\}$ and let $(i_0, \cdots, i_d)$ be a random vector uniformly distributed over I. Let $(D_i)_i$ be a family of random variables such that $(D_{i_0}, D_{i_i}, \cdots, D_{i_d}) = (X \oplus \bigoplus_i M_i, M_1, \cdots, M_d)$ and, for every $j \neq i_0, ..., i_d$, D_j is uniformly distributed and mutually independent of $(D_i)_{i\neq j}$. Let $(S_i)_i$ be a family of t signals following the Hamming weight model corresponding to the processing of the D_i's. Then we have:*

$$\rho\left[\mathrm{H}(X), IC_{d,I}(X)\right] = \frac{1}{\sqrt{\#I}}\rho(n, d, \sigma) \ .$$

4 A Generic Scheme Combining Higher-order Masking and Shuffling

In this section, we propose a generic scheme to protect block cipher implementations by combining higher-order masking and shuffling. First we introduce the general block cipher model and then we describe the proposed scheme. Afterward, we investigate the possible attack paths and we deduce a strategy for choosing the scheme parameters (*i.e.* the masking and shuffling orders, see Sect. 4.2).

4.1 Block Cipher Model

A block cipher is parameterized by a *master key* and it transforms a plaintext block into a ciphertext block through the repetition of key-dependent *round transformations*. We denote by p, and we call *state*, the temporary value taken by the ciphertext during the algorithm. In practice, the cipher is *iterative*, which means that it applies several times the same round transformation φ to the state.

This round transformation is parameterized by a *round key* k that is derived from the master key.

In our model, φ is composed of different operations: a key addition layer (by xor), a non-linear layer γ and a linear layer λ:

$$\varphi[k](p) = [\lambda \circ \gamma](p \oplus k) \ .$$

We assume that the non-linear layer applies the same non-linear transformation S, called *S-box*, on N independent n-bit parts p_i of the state: $\gamma(p) = \big(S(p_1), \cdots, S(p_N)\big)$. For efficiency reasons, the S-box is usually implemented by a look-up table. The linear layer λ is composed of L linear operations λ_i that operate on L independent l-bit parts $p_{i(l)}$ of the state: $\lambda(p) = \big(\lambda_1(p_{1(l)}), \cdots, \lambda_L(p_{L(l)})\big)$. We also denote by $l' \leqslant l$ the minimum number of bits of a variable manipulated during the processing of λ_i. For instance, the MixColumns layer of AES applies to columns of $l = 32$ bits but it manipulates some elements of $l' = 8$ bits. We further assume that the λ_i's are sufficiently similar to be implemented by one *atomic operation* that is an operation which has the same execution flow whatever the index i.

Remark 1. Linear and non-linear layers may involve different state indexing. In AES for instance, the state is usually represented as a 4×4 matrix of bytes and the non-linear layer usually operates on its elements p_1,...,p_{16} vertically (starting at the top) and from left to right. In this case, the operation λ_1 corresponding to the AES linear layer (that is composed of ShiftRows followed by MixColumns [6]) operates on $p_{1(32)} = (p_1, p_6, p_{11}, p_{16})$.

In the sequel, we shall consider that the key addition and the non-linear layer are merged in a *keyed substitution layer* that adds each key part k_i to the corresponding state part p_i before applying the S-box S.

4.2 Our Scheme

In this section, we describe a generic scheme to protect a round φ by combining higher-order masking and operations shuffling. Our scheme involves a d^{th}-order masking for an arbitrarily chosen d. Namely, the state p is split into $d+1$ shares m_0, ..., m_d satisfying:

$$m_0 \oplus \cdots \oplus m_d = p \ . \tag{8}$$

In practice, m_1, ..., m_d are random masks and m_0 is the masked state defined according to (8). In the sequel, we shall denote by $(m_j)_i$ (resp. $(m_j)_{i(l)}$) the i^{th} n-bit part (resp. the i^{th} l-bit part) of a share m_j. At the beginning of the ciphering the masks are initialized to zero. Then, each time a part of a mask is used during the keyed substitution layer computation, it is refreshed with a new random value (see below). For the reasons given in Sect. 2.1, our scheme uses two different approaches to protect the keyed substitution layer and the linear layer. These are described hereafter.

Protecting the keyed substitution layer. To protect the keyed substitution layer, we use a single d'^{th}-order masked S-box (for some $d' \leqslant d$) to perform all the S-box computations. As explained in Sect. 2.1, such a method is vulnerable to a second-order DPA attack targeting two masked inputs/outputs. To deal with this issue, we make use of a high level of shuffling in order to render such an attack difficult and to keep an homogeneous security level (see Sect. 4.4).

The input of S is masked with d' masks r_1, ..., $r_{d'}$ and its output is masked with d' masks s_1, ..., $s_{d'}$. Namely, a masked S-box S^* is computed that is defined for every $x \in \{0,1\}^n$ by:

$$S^*(x) = S\left(x \oplus \bigoplus_{j=1}^{d'} r_j\right) \oplus \bigoplus_{j=1}^{d'} s_j \ . \tag{9}$$

This masked S-box is then involved to perform all the S-box computations. Namely, when the S-box must be applied to a masked variable $(m_0)_i$, the d masks $(m_j)_i$ of this latter are replaced by the d' masks r_j which enables the application of S^*. The d' masks s_j of the obtained masked output are then switched for d new random masks $(m_j)_i$.

The high level shuffling is ensured by the addition of dummy operations. Namely, the S-box computation is performed t times: N times on a relevant part of the state and $t - N$ times on dummy data. For such a purpose, each share m_j is extended by a dummy part $(m_j)_{N+1}$ that is initialized by a random value at the beginning of the ciphering. The round key k is also extended by such a dummy part k_{N+1}. For each of the t S-box computations, the index i of the parts $(m_j)_i$ to process is read in a table T. This table of size t contains all the indices from 1 to N stored at random positions and its $t-N$ other elements equal $N+1$. Thanks to this table, the S-box computation is performed once on every of the N relevant parts and $t - N$ times on the dummy parts. The following algorithm describes the whole protected keyed substitution layer computation.

Algorithm 1 Protected keyed substitution layer

INPUT: the shares m_0, ..., m_d s.t. $\bigoplus m_i = p$, the round key $k = (k_1, \cdots, k_{N+1})$
OUTPUT: the shares m_0, ..., m_d s.t. $\bigoplus m_i = \gamma(p \oplus k)$

1. **for** $i_T = 1$ **to** t

// Random index pick-up
2. $\quad i \leftarrow T[i_T]$

// Masks conversion : $(m_0)_i \Leftarrow p_i \bigoplus_j r_j$
3. $\quad$ **for** $j = 1$ **to** d' **do** $(m_0)_i \leftarrow ((m_0)_i \oplus r_j) \oplus (m_j)_i$
4. $\quad$ **for** $j = d' + 1$ **to** d **do** $(m_0)_i \leftarrow (m_0)_i \oplus (m_j)_i$

// key addition and S-box computation: $(m_0)_i \Leftarrow S(p_i \oplus k_i) \oplus \bigoplus_j s_j$
5. $\quad (m_0)_i \leftarrow S^*\left((m_0)_i \oplus k_i\right)$

// Masks generation and conversion: $(m_0)_i \Leftarrow S(p_i \oplus k_i) \oplus \bigoplus_j (m_j)_i$

6. **for** $j = 1$ **to** d'
7. $(m_j)_i \leftarrow \mathsf{rand}()$
8. $(m_0)_i \leftarrow ((m_0)_i \oplus (m_j)_i) \oplus s_j$
9. **for** $j = d' + 1$ **to** d
10. $(m_j)_i \leftarrow \mathsf{rand}()$
11. $(m_0)_i \leftarrow (m_0)_i \oplus (m_j)_i$

12. **return** $(m_0, \cdots, m_d)$

Remark 2. In Steps 3 and 8, we used round brackets to underline the order in which the masks are introduced. A new mask is always introduced before removing an old mask. Respecting this order is mandatory for the scheme security.

Masked S-box computation. The look-up table for S^* is computed dynamically at the beginning of the ciphering by performing d' table re-computations such as proposed in [23]. This method has been shown to be insecure for $d' > 2$, or for $d' > 3$ depending on the table re-computation algorithm [5, App. A]. We will therefore consider that one can compute a masked S-box S^* with $d' \leqslant 3$ only. The secure computation of a masked S-box with $d' > 3$ is left to further investigations.

Indices table computation. Several solutions exist in the literature to randomly generate indices permutation over a finite set [10, 18, 19]. Most of them can be slightly transformed to design tables T of size $t \geqslant N$ containing all the indices 1 to N in a random order and whose remaining cells are filled with $N+1$. However, few of those solutions are efficient when implemented in low resources devices. In our case, since t is likely to be much greater than N, we have a straightforward algorithm which tends to be very efficient for $t \gg N$. This algorithm is given in Appendix A (Algorithm 3).

Protecting the linear layer. The atomic operations λ_i are applied on each part $(m_j)_{i(l)}$ of each share m_j in a random order. For such a purpose a table T' is constructed at the beginning of the ciphering that is randomly filled with all the pairs of indices $(j, i) \in \{0, \cdots, d\} \times \{1, \cdots, L\}$. The linear layer is then implemented such as described by the following algorithm.

Algorithm 2 Protected linear layer
INPUT: the shares m_0, ..., m_d s.t. $\bigoplus m_i = p$
OUTPUT: the shares m_0, ..., m_d s.t. $\bigoplus m_i = \lambda(p)$

1. **for** $i_{T'} = 1$ **to** $(d+1) \cdot L$
2. $(j, i) \leftarrow T'[i_{T'}]$ // Random index look-up
3. $(m_j)_{i(l)} \leftarrow \lambda_i\big((m_j)_{i(l)}\big)$ // Linear operation
4. **return** $(m_0, \cdots, m_d)$

Indices table computation. To implement the random generation of a permutation T' on $\{0, \cdots, d\} \times \{1, \cdots, L\}$, we followed the outlines of the method proposed in [4]. However, since this method can only be applied to generate permutations on sets with cardinality a power of 2 (which is not *a priori* the case for T'), we slightly modified it. The new version can be found in Appendix A (Algorithm 4).

4.3 Time Complexity

In the following we express the time complexity of each step of our scheme in terms of the parameters (t, d, d', N, L) and of constants a_i that depend on the implementation and the device architecture. Moreover, we provide practical values of these constants (in number of clock cycles) for an AES implementation protected with our scheme and running on a 8051-architecture.

Generation of T (see Appendix A). Its complexity $\mathcal{C}_T$ satisfies:

$$\mathcal{C}_T = t \times a_0 + N \times a_1 + f(N,t) \times a_2 \ ,$$

where $f(N,t) = t \sum_{i=0}^{N-1} \frac{1}{t-i}$. As argued in Appendix A, $f(N,t)$ can be approximated by $t \ln\left(\frac{t}{t-N}\right)$ for $t \gg N$.

Example 1. For our AES implementation, we got $a_0 = 6$, $a_1 = 7$ and $a_2 = 9$.

Generation of T'. Let q denote $\lceil \log_2((d+1)L) \rceil$. The complexity $\mathcal{C}_{T'}$ satisfies:

$$\mathcal{C}_{T'} = \begin{cases} q \times a_0 + 2^q \times (a_1 + q \times a_2) \text{ if } q = \log_2((d+1)L), \\ q \times a_0 + 2^q \times (a_1 + q \times a_2) + 2^q \times a_3 \text{ otherwise.} \end{cases}$$

Example 2. For our AES implementation, we got $a_0 = 3$, $a_1 = 15$ and $a_2 = 14$, $a_3 = 17$.

Generation the Masked S-box. Its complexity $\mathcal{C}_{MS}$ satisfies:

$$\mathcal{C}_{MS} = d' \times a_0 \ .$$

Example 3. For our AES implementation, we got $a_0 = 4352$.

Protected keyed Substitution Layer. Its complexity $\mathcal{C}_{SL}$ satisfies:

$$\mathcal{C}_{SL} = t \times (a_0 + d \times a_1 + d' \times a_2) \ .$$

Example 4. For our AES implementation, we got $a_0 = 55$, $a_1 = 37$ and $a_2 = 18$.

Protected Linear Layer. Its complexity $\mathcal{C}_{LL}$ satisfies:

$$\mathcal{C}_{LL} = (d+1)L \times a_0 \ .$$

Example 5. For our AES implementation, we got $a_0 = 169$.

4.4 Attack Paths

In this section, we list attacks combining higher-order and integrated DPA that may be attempted against our scheme. Section 3 is then involved to associate each attack with a correlation coefficient that depends on the leakage noise deviation σ, the block cipher parameters (n, N, l', L) and the security parameters (d, d', t). As argued, these coefficients characterize the attacks efficiencies and hence the overall resistance of the scheme.

Remark 3. In this paper, we only consider known plaintext attack *i.e.* we assume the different sensitive variables uniformly distributed. In a chosen plaintext attack, the adversary would be able to fix the value of some sensitive variables which could yield better attack paths. We do not take such attacks into account and let them for further investigations.

Every sensitive variable in the scheme is (1) either masked with d unique masks or (2) masked with d' masks shared with other sensitive variables (during the keyed substitution layer).

(1). In the first case, the $d+1$ shares appear during the keyed substitution layer computation and the linear layer computation. In both cases, their manipulation is shuffled.

(1.1). For the keyed substitution layer (see Algorithm 1), the $d+1$ shares all appear during a single iteration of the loop among t. The attack consists in combining the $d+1$ corresponding signals for each loop iteration and to sum the t obtained combined signals. Proposition 2 implies that this attack can be associated with the following correlation coefficient ρ_1:

$$\rho_1(t, d) = \frac{1}{\sqrt{t}} \rho(n, d, \sigma) \ . \tag{10}$$

(1.2). For the linear layer (see Algorithm 2), the $d+1$ shares appear among $(d+1) \cdot L$ possible operations. The attack consists in summing all the combinations of $d+1$ signals among the $(d+1) \cdot L$ corresponding signals. According to Proposition 3, this attack can be associated with the following correlation coefficient ρ_2:

$$\rho_2(L, d) = \frac{1}{\sqrt{\binom{(d+1) \cdot L}{d+1}}} \rho(l', d, \sigma) \ . \tag{11}$$

Remark 4. In the analysis above, we chose to not consider attacks combining shares processed in the linear layers together with shares processed in the keyed substitution layer. Actually, such an attack would yield to a correlation coefficient upper bounded by the maximum of the two correlations in (10) and (11).

(2). In the second case, the attack targets a d'^{th}-order masked variable occurring during the keyed substitution layer. Two alternatives are possible.

(2.1). The first one is to simultaneously target the masked variable (that appears in one loop iteration among t) and the d' masks that appear at fixed

times (*e.g.* in every loop iteration of Algorithm 1 or during the masked S-box computation). The attack hence consists in summing the t possible combined signals obtained by combining the masked variable signal (t possible times) and the d' masks signals (at fixed times). According to Proposition 3, this leads to a correlation coefficient ρ_3 that satisfies:

$$\rho_3(t, d') = \frac{1}{\sqrt{t}} \rho(n, d', \sigma) \ . \tag{12}$$

(2.2). The second alternative is to target two different variables both masked with the same sum of d' masks (for instance two masked S-box inputs or outputs). These variables are shuffled among t variables. The attack hence consists in summing all the possible combinations of the two signals among the t corresponding signals. According to Proposition 3, this leads to a correlation coefficient ρ_4 that satisfies:

$$\rho_4(t) = \frac{1}{\sqrt{t \cdot (t-1)}} \rho(n, 2, \sigma) \ . \tag{13}$$

4.5 Parameters Setting

The security parameters (d, d', t) can be chosen to satisfy an arbitrary resistance level characterized by an upper bound ρ^* on the correlation coefficients corresponding to the different attack paths exhibited in the previous section. That is, the parameters are chosen to satisfy the following inequality:

$$\max(|\rho_1|, |\rho_2|, |\rho_3|, |\rho_4|) \leqslant \rho^* \ . \tag{14}$$

Among the 3-tuples (d, d', t) satisfying the relation above, we select one among those that minimize the timing complexity (see Sect. 4.3).

5 Application to AES

We implemented our scheme for AES on a 8051-architecture. According to Remark 1, the ShiftRows and the MixColumns were merged in a single linear layer applying four times the same operation (but with different state indexings). The block cipher parameters hence satisfy: $n = 8$, $N = 16$, $l = 32$, $l' = 8$ and $L = 4$.

Remark 5. In [8], it is claimed that the manipulations of the different bytes in the MixColumns can be shuffled. However it is not clear how to perform such a shuffling in practice since the processing differs according to the byte index.

Table 1 summarizes the timings obtained for the different steps of the scheme for our implementation.

Remark 6. The unprotected round implementation has been optimized, in particular by only using variables stored in DATA memory. Because of memory constraints and due to the scalability of the code corresponding to the protected

Table 1. Timings for the different steps of the scheme for an AES implementation on a 8051-architecture.

T Generation	$\mathcal{C}_T = 112 + t\left(6 + 9\sum_{i=0}^{15} \frac{1}{t-i}\right)$
T' Generation	$\mathcal{C}_{T'} = 3q + 2^q(15 + 14q) \quad [+17 \times 2^q]$
Masked S-box Generation	$\mathcal{C}_{MS} = 4352d'$
Pre-computations	$\mathcal{C}_T + \mathcal{C}_{T'} + \mathcal{C}_{MS}$
Substitution Layer	$\mathcal{C}_{SL} = t(55 + 37d + 18d')$
Linear Layer	$\mathcal{C}_{LL} = 676(d+1)$
Protected Round	$\mathcal{C}_{SL} + \mathcal{C}_{LL} = 676(d+1) + t(55 + 37d + 18d')$
Unprotected Round	432

round, many variables have been in stored in XDATA memory which made the implementation more complex. This explains that, even for $d = d' = 0$ and $t = 16$ (*i.e.* when there is no security), the protected round is more time consuming than the unprotected round.

We give hereafter the optimal security parameters (t, d, d') for our AES implementation according to some illustrative values of the device noise deviation σ and of correlation bound ρ^*. We consider three noise deviation values: 0, $\sqrt{2}$ and $4\sqrt{2}$. In the Hamming weight model, these values respectively correspond to a signal-to-noise ratio (SNR) to $+\infty$, 1 and $\frac{1}{4}$. We consider four correlation bounds: 10^{-1}, 10^{-2}, 10^{-3}, and 10^{-4}. The security parameters and the corresponding timings for the protected AES implementation are given in Table 5. Note that all the rounds have been protected.

Table 2. Optimal parameters and timings according to SNR and ρ^*.

	SNR $= +\infty$			SNR $= 1$			SNR $= \frac{1}{4}$		
ρ^*	t	d d'	timings	t	d d'	timings	t	d d'	timings
10^{-1}	16	1 1	3.66×10^4	16	1 1	3.66×10^4	16	1 0	2.94×10^4
10^{-2}	20	3 3	8.57×10^4	20	2 2	6.39×10^4	16	1 1	3.66×10^4
10^{-3}	1954	4 3	5.08×10^6	123	3 3	3.13×10^5	16	2 2	5.75×10^4
10^{-4}	195313	5 3	5.75×10^8	12208	4 3	3.15×10^7	19	3 3	8.35×10^4

When SNR $= +\infty$, the bound $d' \leqslant 3$ implies an intensive use of shuffling in the keyed substitution layer. The resulting parameters for correlation bounds 10^{-3} and 10^{-4} imply timings that quickly become prohibitive. A solution to overcome this drawback would be to design secure table re-computation algorithms for $d' \geqslant 3$. Besides, these timings underline the difficulty of securing block ciphers implementations with pure software countermeasures. When the leakage signals are not very noisy (SNR $= 1$), timings clearly decrease (by a factor from

10 to 20). This illustrates, once again, the soundness of combining masking with noise addition. This is even clearer when the noise is stronger (SNR $= \frac{1}{4}$), where it can be noticed that the addition of dummy operations is almost not required to achieve the desired security level.

6 Conclusion

In this paper, we have conducted an analysis that quantifies the efficiency of advanced DPA attacks targeting masking and shuffling. Based on this analysis, we have designed a generic scheme combining higher-order masking and shuffling. This scheme generalizes to higher orders the solutions previously proposed in the literature. It is moreover scalable and its security parameters can be chosen according to any desired resistance level. As an illustration, we applied it to protect a software implementation of AES for which we gave several security/efficiency trade-offs.

References

1. M.-L. Akkar and C. Giraud. An Implementation of DES and AES, Secure against Some Attacks. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.
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. C. Clavier, J.-S. Coron, and N. Dabbous. Differential Power Analysis in the Presence of Hardware Countermeasures. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000.
4. J.-S. Coron. A New DPA Countermeasure Based on Permutation Tables. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *Security and Cryptography for Networks, 6th International Conference, SCN 2008*, volume 5229 of *Lecture Notes in Computer Science*, pages 278–292. Springer, 2008.
5. J.-S. Coron, E. Prouff, and M. Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 2007.
6. FIPS PUB 197. *Advanced Encryption Standard*. National Institute of Standards and Technology, Nov. 2001.
7. 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.
8. P. Herbst, E. Oswald, and S. Mangard. An AES Smart Card Implementation Resistant to Power Analysis Attacks. In J. Zhou, M. Yung, and F. Bao, editors, *Applied Cryptography and Network Security – ANCS 2006*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 2006.

9. M. Joye, P. Paillier, and B. Schoenmakers. On Second-order Differential Power Analysis. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.
10. D. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, third edition, 1988.
11. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
12. S. Mangard. Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness. In T. Okamoto, editor, *Topics in Cryptology – CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.
13. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks – Revealing the Secrets of Smartcards*. Springer, 2007.
14. 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.
15. T. Messerges. Using Second-order Power Analysis to Attack DPA Resistant Software. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.
16. E. Oswald and S. Mangard. Template Attacks on Masking—Resistance is Futile. In M. Abe, editor, *Topics in Cryptology – CT-RSA 2007*, volume 4377 of *Lecture Notes in Computer Science*, pages 243–256. Springer, 2007.
17. E. Oswald, S. Mangard, C. Herbst, and S. Tillich. Practical Second-order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2006.
18. J. Patarin. How to Construct Pseudorandom and Super Pseudorandom Permutation from one Single Pseudorandom Function. In R. Rueppel, editor, *Advances in Cryptology – EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1992.
19. J. Pieprzyk. How to Construct Pseudorandom Permutations from Single Pseudorandom Functions Advances. In I. Damgård, editor, *Advances in Cryptology – EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 140–150. Springer, 1990.
20. E. Prouff, M. Rivain, and R. Bévan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Comput.*, 58(6):799–811, 2009.
21. 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.
22. M. Rivain, E. Prouff, and J. Doget. Higher-order Masking and Shuffling for Software Implementations of Block Ciphers. Cryptology ePrint Archive, 2009. `http://eprint.iacr.org/`.
23. 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.
24. F.-X. Standaert, E. Peeters, G. Rouvroy, and J.-J. Quisquater. An Overview of Power Analysis Attacks Against Field Programmable Gate Arrays. *IEEE*, 94(2):383–394, 2006.

25. S. Tillich and C. Herbst. Attacking State-of-the-Art Software Countermeasures-A Case Study for AES. In E. Oswald and P. Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2008.
26. S. Tillich, C. Herbst, and S. Mangard. Protecting AES Software Implementations on 32-Bit Processors Against Power Analysis. In J. Katz and M. Yung, editors, *ACNS*, volume 4521 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2007.
27. L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer Texts in Statistics, 2005.

A Algorithms for Index Tables Generations

Generation of T. To generate T, we start by initializing all the cells of T to the value $N+1$. Then, for every $j \leqslant N$, we randomly generate an index $i < t$ until $T[i] = N+1$ and we move j into $T[i]$. The process is detailed hereafter.

Algorithm 3 Generation of T

INPUT: state's length N and shuffling order t
OUTPUT: indices permutation table T

1. **for** $i \leftarrow 0$ **to** $t-1$
2. **do** $T[i] \leftarrow N+1$ // Initialization of T
3. $j \leftarrow 1$
4. **for** $j \leftarrow 1$ **to** N
5. **do** $i \leftarrow rand(t)$ **while** $T[i] = N+1$ // Generate random index $i < t$
6. $T[i] = j$ and $j \leftarrow j+1$
7. **return** T

Complexity Anlysis of loop 4-to-6. The expected number $f(N,t)$ of iterations of the loop 4-to-7 in Algorithm 3 satisfies:

$$f(N,t) = t \cdot (H_t - H_{t-N}) \quad , \tag{15}$$

where for every r, H_r denotes the r^{th} *Harmonic number* defined by $H_r = \sum_{i=1}^{r} \frac{1}{i}$.

Let us argue about (15). For every $j \leqslant N$, the probability that the loop **do-while** ends up after i iterations is $\left(\frac{t-j}{t}\right) \cdot \left(\frac{j}{t}\right)^{i-1}$: at the j^{th} iteration of the **for** loop, the test $T[i] = N+1$ succeeds with probability $p_j = \left(\frac{j}{t}\right)$ and fails with probability $1 - p_j = \left(\frac{t-j}{t}\right)$. One deduces that for every $j \leqslant N$, the expected number of iterations of the loop **do-while** is $\sum_{i \in \mathbb{N}} i \cdot {p_j}^{i-1} \cdot (1-p_j)$. We eventually get that the number of iterations $f(N,t)$ satisfies $f(N,t) = \sum_{j=0}^{N-1} \sum_{i \in \mathbb{N}} i \cdot \left({p_j}^{i-1} - {p_j}^{i}\right)$, that is $f(N,t) = \sum_{j=0}^{N-1} \sum_{i \in \mathbb{N}} i \cdot {p_j}^{i-1} - \sum_{j=0}^{N-1} \sum_{i \in \mathbb{N}} (i+1) \cdot {p_j}^{i} + \sum_{j=0}^{N-1} \sum_{i \in \mathbb{N}} {p_j}^{i}$. As the two first sums in the right-hand side of the previous equation are equal, one deduces that $f(N,t)$ equals $\sum_{j=0}^{N-1} \sum_{i \in \mathbb{N}} {p_j}^{i}$ that is $\sum_{j=0}^{N-1} \frac{1}{1-p_j}$. Eventually, since p_j equals $\frac{j}{t}$, we get $f(N,t) = \sum_{j=0}^{N-1} \frac{t}{t-j}$ which is equivalent with (15).

Since H_r tends towards $\ln(r) + \gamma$, where γ is the Euler-Mascheroni constant, we can approximate $H_t - H_{t-N}$ by $\ln(t) - \ln(t-N)$. We eventually get the following relation for $t \gg N$:

$$f(N,t) \approx t \cdot \ln\left(\frac{t}{t-N}\right).$$

Generation of T'. In view of the previous complexity, generating a permutation with the same implementation as for T is not pertinent (in this case $t = N$). To generate the permutation T', we follow the outlines of the method proposed in [4]. However, since this method can only be applied to generate permutations on sets with cardinality a power of 2 (which is not a priori the case for T'), we slightly modified it. Let 2^q be the smallest power of 2 which is greater than $(d+1)L$. Our algorithm essentially consists in designing a q-bit random permutation T' from a fixed q-bit permutation π and a family of q random values in $\mathbb{F}_2^q$ (Steps 1 to 6 in Algorithm 4). Then, if $(d+1)L$ is not a power of 2, the table T' is transformed into a permutation over $\{0,\cdots,d\} \times \{1,\cdots,L\}$ by deleting the elements which are strictly greater than $(d+1)L-1$. The process is detailled in pseudo-code hereafter.

Algorithm 4 Generation of T'

INPUT: parameters (d, L) and a n'-bit permutation π with $q = \lceil \log_2((d+1)L) \rceil$
OUTPUT: indices permutation table T'

```
1.  for i ← 0 to q − 1
2.      do alea_i ← rand(q)                    // Initialization of aleas
3.  for j ← 0 to 2^q − 1
4.      do T'[j] ← π[j]
5.      for i ← 0 to q − 1
6.          do T'[j] ← π[T'[j] ⊕ alea_i]       // Process the i^th index
7.  if q ≠ (d + 1)L
8.      then for j ← 0 to (d + 1)L − 1
9.               do i ← j
10.              while T'[i] ≥ (d + 1)L
11.                  do i ← i + 1
12.              T'[j] ← T'[i]
13.  return T'
```

With Algorithm 4, it is not possible to generate all the permutations over $\{0,\cdots,d\} \times \{1,\cdots,L\}$. In our context, we assume that this does not introduce any weakness in the scheme.

Complexity Anlysis of loop 8-to-12 The number of iterations of loop 8-to-12 in Algorithm 4 in the worst case is 2^q.