Higher-Order Side Channel Security and Mask Refreshing

Jean-Sébastien Coron[1], Emmanuel Prouff[2], Matthieu Rivain[3(✉)], and Thomas Roche[2]

[1] Tranef, Paris, France
jscoron@tranef.com
[2] ANSSI, Paris, France
{emmanuel.prouff,thomas.roche}@ssi.gouv.fr
[3] CryptoExperts, Paris, France
matthieu.rivain@cryptoexperts.com

Abstract. Masking is a widely used countermeasure to protect block cipher implementations against side-channel attacks. The principle is to split every sensitive intermediate variable occurring in the computation into $d + 1$ shares, where d is called the *masking order* and plays the role of a security parameter. A masked implementation is then said to achieve d^{th}-order security if any set of d (or less) intermediate variables does not reveal key-dependent information. At CHES 2010, Rivain and Prouff have proposed a higher-order masking scheme for AES that works for any arbitrary order d. This scheme, and its subsequent extensions, are based on an improved version of the shared multiplication processing published by Ishai *et al.* at CRYPTO 2003. This improvement enables better memory/timing performances but its security relies on the refreshing of the masks at some points in the algorithm. In this paper, we show that the method proposed at CHES 2010 to do such mask refreshing introduces a security flaw in the overall masking scheme. Specifically, we show that it is vulnerable to an attack of order $\lceil d/2 \rceil + 1$ whereas the scheme is supposed to achieve d^{th}-order security. After exhibiting and analyzing the flaw, we propose a new solution which avoids the use of mask refreshing, and we prove its security. We also provide some implementation trick that makes our proposed solution, not only secure, but also faster than the original scheme.

1 Introduction

In the late nineties, attacks called *Side Channel Analysis* (SCA for short) have been exhibited against cryptosystems implemented in embedded devices. Since the seminal works [7,8], they have been refined and, in particular, the initial principle has been generalized in order to exploit several leakage points simultaneously. This led to the introduction of the *higher-order SCA* concept.

S. Moriai (Ed.): FSE 2013, LNCS 8424, pp. 410–424, 2014.
DOI: 10.1007/978-3-662-43933-3_21,

Those attacks are based on leakage observations resulting from the handling of several (say d) intermediate variables during the cryptosystem processing. One way to make them ineffective is to randomize the algorithm such that the probability distribution of any vector of less than d observations is independent of the key. To perform this randomization, a standard technique is to apply Boolean masking [2]. It consists in replacing the manipulation of every secret-dependent intermediate variable x (called *sensitive variable*) by that of $d+1$ *shares* $x_0, \ldots, x_d$ satisfying $x_0 \oplus x_1 \oplus \cdots \oplus x_d = x$. Usually, the d shares $x_1, \ldots, x_d$ (called *the masks*) are randomly picked up and the last one x_0 (called *the masked variable*) is processed such that it satisfies the previous equality. When d random masks are involved per sensitive variable, the masking is said to be *of order* d and every d-tuple of intermediate variables of the computation is statistically independent of any sensitive variable. In fact, only attacks exploiting the leakages related to $d+1$ intermediate variables may succeed in retrieving sensitive information. Since the efficiency of such an attack becomes impractical as d increases [2], the masking order is usually considered as a sound criterion to refer to the robustness against SCA.

When applying the principle of masking to secure a block cipher implementation, a so-called *masking scheme* must be designed to operate on the masks and the masked data. It must ensure that the final shares enable the recovery of the expected ciphertext, while satisfying the d^{th}-order *security property* for the chosen order d. When satisfied, the latter property guarantees that no attack of order lower than or equal to d is possible. The main difficulty in designing a Boolean masking scheme lies in masking the non-linear parts of the cipher, the so-called s-boxes.

The first scheme achieving d^{th}-order security for an arbitrary chosen d has been designed by Ishai, Sahai and Wagner in [5]. The here called *ISW scheme* consists in masking the Boolean representation of an algorithm which is composed of logical operations NOT and AND. Securing a NOT for any order d is straightforward since $x = \bigoplus_i x_i$ implies $\text{NOT}(x) = \text{NOT}(x_0) \oplus x_1 \cdots \oplus x_d$. The main contribution of [5] is a method to secure the AND operation for any arbitrary order d (the description of this scheme is recalled in Sect. 2). A direct application of ISW scheme to secure an s-box in software would consist in taking the Boolean representation of the s-box and to process every logical operation successively in a masked way. Since the Boolean representation of common s-boxes involves a huge number of logical operations, a direct application of [5] is often not possible in practice. A solution to deal with this efficiency issue has first been proposed by Rivain and Prouff in [10] for the AES (it will be called RP Scheme in the sequel), and then extended to any block cipher by Carlet *et al.* in [1]. Those works start from the observation that the ISW scheme can be extended to secure a multiplication over any finite field. The core idea is then to represent the s-box to protect as a polynomial function over a finite field and to secure the polynomial evaluation thanks to the ISW scheme.

Since the processing of affine transformations is performed by operating on each share separately, proving its d^{th}-order security is straightforward as noticed

in [5,10]. Actually, only the secure processing of the multiplication $a \times b$ from the sharings $(a_0, a_1, \cdots, a_d)$ and $(b_0, b_1, \cdots, b_d)$ is challenging. In [5], the ISW scheme is shown to be secure at the order $d/2$, meaning that a d^{th}-order security is obtained from a masking of order $2d$. As the complexity of ISW is quadratic in the number of shares, such a doubling of the masking order makes the resulting implementation roughly 4 times slower. To avoid such a penalty, the authors of [10] additionally assume that the multiplication operands sharings involve mutually independent random masks and they prove that ISW scheme is actually d^{th}-order secure under the latter condition.[1] To satisfy this independence when a multiplication of the form $a \times g(a)$ occurs in a secure s-box processing, with g being a linear function, the authors of [10] suggest to refresh the mask of $g(a)$ before performing the secure multiplication $a \times g(a)$. To do so, they use a so-called *mask-refreshing* procedure which from the sharing of $g(a)$, computes a new sharing of with fresh random values. Such a procedure is mandatory in [10] as well as in the subsequent schemes [1,6] to deduce the d^{th}-order security of the whole s-box processing from that of the secure multiplications.

Our Contribution. In this paper, we show that the actual proposal of mask-refreshing procedure in [10] fails in reaching its goal and introduces a security flaw in the overall masking scheme. In fact, even if both the mask-refreshing procedure and the ISW multiplication are secure at order d, their composition is insecure and it is defeated by an attack of order $\lceil d/2 \rceil + 1$. After exhibiting and analyzing the flaw, we propose a secure solution which avoids the use of mask refreshing, and we prove its d^{th}-order security. Our solution consists in adapting ISW scheme to directly process, from a sharing of a, the multiplications of the form $a \times g(a)$ with g being a linear function. We also provide an improvement that allows to avoid costly multiplications over $\mathbb{F}_{2^n}$ in this context. As a consequence, the resulting shared multiplication for $a \times g(a)$ is not only secure but also more efficient than the original scheme proposed in [10].

Paper Organisation. In Sect. 2, we recall the ISW scheme and the existing solutions to mask a full s-box computation at any order d. The flaw from the composition of the mask-refreshing procedure and the secure multiplication is exhibited and analyzed in Sect. 3. Then, we describe our new algorithm and prove its d^{th}-order security in Sect. 4. Eventually, implementation results are provided in Sect. 5 to report on the efficiency of our improved algorithm when plugged in RP Scheme.

2 Higher-Order Masking Schemes for S-Boxes

This section presents the different schemes published in the literature to mask an s-box processing at any order d. We first recall the ISW scheme [5] which is the starting point of the different solutions. Then we detail the RP Scheme for

[1] Specifically [10] requires that every $2d$-tuple composed of d elements from $(a_i)_i$ and of d elements from $(b_i)_i$ is uniformly distributed and independent of any sensitive variable.

the AES s-box [10]. Eventually, we briefly recall the improvement proposed by Kim *et al.* [6], and the extension put forward by Carlet *et al.* [1].

Ishai-Sahai-Wagner's Scheme. Let a and b be binary values from $\mathbb{F}_2$ and let $(a_i)_{0\leqslant i\leqslant d}$ and $(b_i)_{0\leqslant i\leqslant d}$ be d^{th}-order sharings of a and b respectively. To securely compute a sharing of $c = a \times b$ from $(a_i)_i$ and $(b_i)_i$, the ISW method works as follows:[2]

1. For every $0 \leqslant i < j \leqslant d$, pick up a random bit $r_{i,j}$.
2. For every $0 \leqslant i < j \leqslant d$, compute $r_{j,i} = (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$.
3. For every $0 \leqslant i \leqslant d$, compute $c_i = a_i b_i \oplus \bigoplus_{j\neq i} r_{i,j}$.

It can be checked that the obtained shares form a sound encoding of c. Namely, we have: $\bigoplus_{i=0}^{d} c_i = \big(\bigoplus_{i=0}^{d} a_i\big)\big(\bigoplus_{i=0}^{d} b_i\big) = ab = c$. In [5] it is moreover shown that the above computation achieves security at order $d/2$.

Rivain and Prouff's Scheme. In [10], the authors proposed to use the ISW scheme to secure a multiplication $c = a \times b$ over $\mathbb{F}_{2^n}$ for any n greater than 1. For completeness sake, we recall the obtained algorithm hereafter, where the multiplication over $\mathbb{F}_{2^n}$ is denoted $\odot$.

Algorithm 1. SecMult

Input: shares a_i satisfying $\bigoplus_i a_i = a$, shares b_i satisfying $\bigoplus_i b_i = b$
Output: shares c_i satisfying $\bigoplus_i c_i = a \odot b$

for $i = 0$ **to** d **do**
  for $j = i + 1$ **to** d **do**
    $r_{i,j} \leftarrow^{\$} \mathbb{F}_{2^n}$
    $r_{j,i} \leftarrow (r_{i,j} \oplus a_i \odot b_j) \oplus a_j \odot b_i$
  end for
end for
for $i = 0$ **to** d **do**
  $c_i \leftarrow a_i \odot b_i$
  for $j = 0$ **to** d, $j \neq i$ **do** $c_i \leftarrow c_i \oplus r_{i,j}$
end for
return $(c_0, c_1, \ldots, c_d)$

As shown in [10], for Algorithm 1 to be secure at order d, the masks $(a_i)_{i\geqslant 1}$ and $(b_i)_{i\geqslant 1}$ in input must be mutually independent. When this condition is not satisfied, the authors suggest to refresh the masks of one of the operands prior to the secure multiplication processing. For such a purpose, they suggest to apply the following mask-refreshing procedure.

[2] The use of brackets indicates the order in which the operations are performed, which is mandatory for security of the scheme.

Algorithm 2. RefreshMasks

Input: shares $(z_i)_i$ satisfying $\bigoplus_i z_i = z$
Output: shares $(z'_i)_i$ satisfying $\bigoplus_i z'_i = z$
1: $(z'_0, z'_1, \ldots, z'_d) \leftarrow (z_0, z_1, \ldots, z_d)$
2: **for** $i = 1$ **to** d **do**
3: $\quad r_i \leftarrow^{\$} \mathbb{F}_{2^n}$
4: $\quad z'_0 \leftarrow z'_0 \oplus r_i$
5: $\quad z'_i \leftarrow z'_i \oplus r_i$
6: **end for**
7: **return** $(z'_0, z'_1, \ldots, z'_d)$

In [10], Algorithms 1 and 2 are eventually involved to secure the whole exponentiation to the power 254 over $\mathbb{F}_{256}$ (that is the non-linear part of the AES s-box). We recall the complete exponentiation algorithm hereafter:

Algorithm 3. SecExp254

Input: shares x_i satisfying $\bigoplus_i x_i = x$
Output: shares y_i satisfying $\bigoplus_i y_i = x^{254}$
1: **for** $i = 0$ **to** d **do** $z_i \leftarrow x_i^2$ // $\bigoplus_i z_i = x^2$
2: $(z_0, z_1, \ldots, z_d) \leftarrow \mathsf{RefreshMasks}(z_0, z_1, \ldots, z_d)$
3: $(y_0, y_1, \ldots, y_d) \leftarrow \mathsf{SecMult}\big((x_0, x_1, \ldots, x_d), (z_0, z_1, \ldots, z_d)\big)$ // $\bigoplus_i y_i = x^3$
4: **for** $i = 0$ **to** d **do** $w_i \leftarrow y_i^4$ // $\bigoplus_i w_i = x^{12}$
5: $(w_0, w_1, \ldots, w_d) \leftarrow \mathsf{RefreshMasks}(w_0, w_1, \ldots, w_d)$
6: $(y_0, y_1, \ldots, y_d) \leftarrow \mathsf{SecMult}\big((y_0, y_1, \ldots, y_d), (w_0, w_1, \ldots, w_d)\big)$ // $\bigoplus_i y_i = x^{15}$
7: **for** $i = 0$ **to** d **do** $y_i \leftarrow y_i^{16}$ // $\bigoplus_i y_i = x^{240}$
8: $(y_0, y_1, \ldots, y_d) \leftarrow \mathsf{SecMult}\big((y_0, y_1, \ldots, y_d), (w_0, w_1, \ldots, w_d)\big)$ // $\bigoplus_i y_i = x^{252}$
9: $(y_0, y_1, \ldots, y_d) \leftarrow \mathsf{SecMult}\big((y_0, y_1, \ldots, y_d), (z_0, z_1, \ldots, z_d)\big)$ // $\bigoplus_i y_i = x^{254}$

Kim-Hong-Lim's Improvement. In [6], Kim *et al.* propose an alternative to Rivain-Prouff's scheme based on the so-called *tower-field representation* of the AES s-box from [11]. The exponentiation is performed by representing the field $\mathbb{F}_{256}$ as a quadratic extension of $\mathbb{F}_{16}$. In such a way, the AES s-box can be computed with 5 multiplications over $\mathbb{F}_{16}$ rather than 4 multiplications over $\mathbb{F}_{256}$. Even if the number of field multiplications is greater than in the orginal scheme, multiplications over $\mathbb{F}_{16}$ can be tabulated, which eventually leads to a significant timing improvement.

Carlet-Goubin-Prouff-Quisquater-Rivain's Scheme. In [1], Carlet *et al.* extend [10] to design a higher-order masking scheme for any nonlinear function from $\{0,1\}^n$ to $\{0,1\}^m$ with $m \leqslant n$ and n small (typically $n \in \{4, 6, 8\}$). Their approach is to express such an s-box as a sequence of affine functions over $\mathbb{F}_2^n$ and multiplications over $\mathbb{F}_{2^n}$. Such a strategy is always possible since any

function from $\{0,1\}^n$ to $\{0,1\}^m$ (with $m \leqslant n$) can be represented by a polynomial $\bigoplus_{i=0}^{2^n-1} \alpha_i x^i$ in $\mathbb{F}_{2^n}[x]$ and the α_i can be obtained from the s-box look-up table by applying Lagrange Interpolation Theorem. As for the secure exponentiation (Algorithm 3), the secure evaluation algorithms proposed in [1] involve a mask-refreshing procedure to change the sharing of some intermediate results before applying the multiplication scheme. Once again, each sub-routine of the evaluation procedure (affine transformation, multiplication processing and mask refreshing) is provably d^{th}-order secure and the security of the whole function evaluation is essentially deduced from those local securities.

3 A Flaw from the Mask-Refreshing Procedure

Even though Algorithms 1 and 2 are both secure against d^{th}-order SCA when considered separately, we show hereafter that their sequential application, as done in Steps 2–3 and Steps 5–6 of Algorithm 3, is not. Namely, we exhibit a tuple of $\lceil d/2 \rceil + 1$ intermediate variables that jointly depend on the sensitive input. Hence, for $d > 1$, this flaw invalidates the claim that the schemes proposed in [1,6,10] achieve d^{th}-order security.

To exhibit the flaw, we assume that the attacked s-box evaluation procedure contains the following sequence:

$$
\begin{aligned}
&(z_0, z_1, \cdots, z_d) \leftarrow (g(x_0), g(x_1), \ldots, g(x_d)),\\
&(z'_0, z'_1, \cdots, z'_d) \leftarrow \mathsf{RefreshMasks}((z_0, z_1, \cdots, z_d)),\\
&(y_0, y_1, \cdots, y_d) \leftarrow \mathsf{SecMult}((x_0, x_1, \cdots, x_d), (z'_0, z'_1, \cdots, z'_d)),
\end{aligned}
$$

with $(x_i)_i$, being a sharing of some sensitive variable x and with g being some $\mathbb{F}_2$-linear function. Two examples of occurrence of this sequence can be found in Algorithm 3: from Step 1 to Step 3 (with the function g corresponding to a squaring over $\mathbb{F}_{256}$), and from Step 4 to Step 6 (with the function g corresponding to a raising to the 4 over $\mathbb{F}_{256}$). The sequence above also occurs in the schemes proposed in [1,6]

For the sake of clarity, we only consider the case where d is even (in the odd case an extra intermediate variable would be needed). The flaw arises from a particular intermediate variable of the mask refreshing combined with $d/2$ intermediate variables of the multiplication. Namely, the targeted intermediate variables are:

– the variable z'_0 just after the fourth step during the $(d/2)^{\text{th}}$ iteration of the loop in RefreshMasks (Algorithm 2), denoted ℓ_0 hereafter, and which satisfies

$$
\begin{aligned}
\ell_0 &= z_0 \oplus \bigoplus_{i=1}^{d/2} r_i\\
&= z \oplus \bigoplus_{i=1}^{d} z_i \oplus \bigoplus_{i=1}^{d/2} r_i\\
&= z \oplus \bigoplus_{i=1}^{d/2} (z_i \oplus r_i \oplus z_{d/2+i})\\
&= g(x) \oplus \bigoplus_{i=1}^{d/2} \big(g(x_i) \oplus r_i \oplus g(x_{d/2+i})\big),
\end{aligned}
\tag{1}
$$

– the product $z'_i \odot x_j$ arising at Step 4 of SecMult (Algorithm 1 called on $(x_i)_i$ and $(z'_i)_i$) for every $i \in \{1, 2, \ldots, d/2\}$ and $j = i + d/2$, denoted ℓ_i hereafter, and which satisfies

$$\ell_i = z'_i \odot x_{d/2+i} = (z_i \oplus r_i) \odot x_{d/2+i} = (g(x_i) \oplus r_i) \odot x_{d/2+i}. \quad (2)$$

In a nutshell, the intermediate variable $\ell_i = (g(x_i) \oplus r_i) \odot x_{d/2+i}$ is statistically dependent on the sum $g(x_i) \oplus r_i \oplus g(x_{d/2+i})$ involved to mask $z = g(x)$ in the expression of ℓ_0 (see (1)). Therefore, the $(d/2)$-tuple $(\ell_i)_i$ defined in (2) provides information on the $d/2$ masks $g(x_i) \oplus r_i \oplus g(x_{d/2+i})$ and this information can be used to partially unmask ℓ_0. In other terms, the family of $d/2 + 1$ intermediate variables $\ell_0, \ell_1, \ldots, \ell_{d/2}$ depends on the sensitive variable x.

In the context of side-channel attacks, the physical leakage of an implementation does not reveal the exact values of the intermediate variables but a noisy function of them. That is why, and according to the methodology described in [12], we analyze hereafter the quantity of information about x that an attacker can expect to retrieve from noisy leakages on the ℓ_i, and we report the results of some standard side-channel attack simulations in this context.

Information Theoretic Evaluation. To estimate the sensitive information leakage corresponding to the identified flaw, we conduct hereafter an information theoretic analysis for $d = 2$ (i.e. data are split in 3 shares). For comparison purpose, we also conduct it for the sensitive information leakage corresponding to a first-order and second-order Boolean masking. To this purpose we consider that the leakage related to a variable manipulation corresponds to the Hamming weight of the variable (denoted $\mathrm{HW}(\cdot)$) affected by an independent Gaussian noise.

Let $(B_i)_{i=1,2,3}$, denote three mutually independent random variables with zero mean and standard deviation σ and let $(M_i)_{i=1,2,3}$ be three mutually independent random variables with uniform distribution over $\mathbb{F}_{256}$. For the three considered cases, we computed the mutual information $I(X; \boldsymbol{L})$ between the targeted sensitive variable $X \in \mathbb{F}_{256}$ and the vector of leakages $\boldsymbol{L} = (L_i)_i$ defined as follows depending on the case:

1. For the flaw described in this paper with $d = 2$ and g being the squaring in $\mathbb{F}_{256}$, we have $\boldsymbol{L} = (L_1, L_2)$ such that:

$$L_1 = \mathrm{HW}(X^2 \oplus M_1^2 \oplus M_2^2 \oplus M_3) + B_1,$$
$$L_2 = \mathrm{HW}(M_2 \odot (M_1^2 \oplus M_3)) + B_2.$$

2. For the classical third-order leakage on a second-order Boolean masking, we have $\boldsymbol{L} = (L_1, L_2, L_3)$ where

$$L_1 = \mathrm{HW}(X \oplus M_1 \oplus M_2) + B_1,$$
$$L_2 = \mathrm{HW}(M_1) + B_2,$$
$$L_3 = \mathrm{HW}(M_2) + B_3.$$

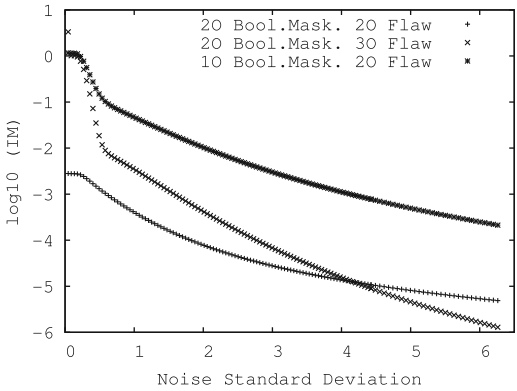

Fig. 1. Mutual information $I(X; \boldsymbol{L})$ over an increasing σ.

3. For the classical second-order leakage on a first-order Boolean masking, we have $\boldsymbol{L} = (L_1, L_2)$ where

$$L_1 = \mathrm{HW}(X \oplus M_1) + B_1,$$
$$L_2 = \mathrm{HW}(M_1) + B_2.$$

The results are given in Fig. 1. We see that, for $d = 2$ and $g = (\cdot)^2$, the flaw described in this paper delivers only small information about the sensitive variable. This can be explained by the algebraic complexity of the relation between the sensitive variable and the two leakages. However, the progression of the information leakage as σ grows is comparable to that of the classical second-order leakage. This means that the security of the flawed second-order masking scheme tends towards the security of a first-order masking scheme as the amount of noise increases.

Attack Simulations. We have analyzed above the information leakage resulting from the exhibited flaw in comparison to unflawed first-order masking and second-order masking. However, we did not discuss the capacity of an attacker to exploit this information leakage using classical side-channel attack techniques. To fill this gap we applied two classical side-channel distinguishers on simulated traces using the introduced leakage model for different values of noise standard deviations. Specifically we launched a second-order Correlation Power Analysis (CPA for short) and a second-order Mutual Information Analysis (MIA for short). The second-order CPA was performed by means of the centered product combining function and its associated optimal prediction function as described in [9], whereas the second-order MIA used an histogram-based bivariate pdf estimation [4]. None of these attacks reached a success rate greater than 20 % when applied on the exhibited second-order flaw for a number of leakage measurements up to one million, even when the noise component was null (i.e. $\sigma = 0$). These results show that although the information leakage is comparable to a

classical leakage of order $\lceil d/2 \rceil + 1$ over an increasing noise, it seems difficult to turn it into an efficient key-recovery using common side-channel attack tools. It is however not excluded that a more powerful attacker using advanced side-channel techniques (e.g. multivariate and/or profiling attacks) could properly exploit the exhibited information leakage.

4 A Secure Solution

In the previous section we have exhibited a flaw of order $\lceil d/2 \rceil + 1$ in the d^{th}-order masking scheme proposed in [10] (and its extensions [1,6]). This flaw arises from the mask-refreshing procedure involved in the secure computation of the multiplications of the form $x \odot g(x)$ (where g is a linear function). Although the resulting information leakage is small and seems difficult to exploit by standard side-channel attack techniques, it is asymptotically stronger than the information from a proper d^{th}-order secure masking scheme. In order to avoid such a security flaw, we propose in this section a new solution for the secure masked processing of multiplications of the form $x \odot g(x)$.

Let a and b be two sensitive variables such that $b = g(a)$ for a $\mathbb{F}_2$-linear function g. When such a relation stands for a and b in Algorithm 3, their corresponding sharings $(a_i)_i$ and $(b_i)_i$ before the call to the mask-refreshing procedure satisfy $b_i = g(a_i)$ for every $i \in [0; d]$. By exploiting this property, the idea is to modify the secure multiplication algorithm in such a way that the masks refreshing is not longer needed.

Before introducing our solution, let us introduce the function f defined from $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$ to $\mathbb{F}_{2^n}$ by

$$f(x, y) = x \odot g(y) \oplus g(x) \odot y,$$

where $\odot$ denotes the multiplication over $\mathbb{F}_{2^n}$. It can be checked that the $\mathbb{F}_2$-linearity of g implies the $\mathbb{F}_2$-bilinearity of f. That is, for every $x, y, r \in \mathbb{F}_{2^n}$, f satisfies:

$$f(x, y) = f(x \oplus r, y) \oplus f(r, y) = f(x, y \oplus r) \oplus f(x, r). \tag{3}$$

When b_i equals $g(a_i)$ for every i, the value $r_{j,i}$ computed at Step 4 of the ISW Scheme (Algorithm 1) satisfies:

$$r_{j,i} = a_i b_j \oplus a_j b_i \oplus r_{i,j} = f(a_i, a_j) \oplus r_{i,j},$$

where $r_{i,j}$ is a freshly generated random value. One can then compute $r_{j,i}$ by evaluating f on (a_i, a_j) and by adding the random value $r_{i,j}$. However, $f(a_i, a_j)$ cannot be directly computed since it would leak on two different shares of a at the same time. To avoid such a leakage we use an additional fresh random value, denoted $r'_{i,j}$, to split the computation of $f(a_i, a_j)$ into the computation of $f(a_i, a_j \oplus r'_{i,j})$ and $f(a_i, r'_{i,j})$. That is, the variable $r_{j,i}$ is computed as:

$$r_{j,i} = \big(r_{i,j} \oplus f(a_i, r'_{i,j})\big) \oplus f(a_i, a_j \oplus r'_{i,j}),$$

where the brackets indicate the order in which the operations are processed. Doing so, we avoid any joint leakage on a_i and a_j. We give hereafter the algorithmic description of our solution.

Algorithm 4. Secure evaluation of $h : x \mapsto x \odot g(x)$

Input: shares $(a_i)_i$ such that $\bigoplus_i a_i = a$
Output: shares c_i satisfying $\bigoplus_i c_i = a \odot g(a)$ for some $\mathbb{F}_2$-linear function g

1: **for** $i = 0$ **to** d **do**
2: **for** $j = i + 1$ **to** d **do**
3: $r_{i,j} \leftarrow^{\$} \mathbb{F}_{2^n}$
4: $r'_{i,j} \leftarrow^{\$} \mathbb{F}_{2^n}$
5: $t \leftarrow r_{i,j} \oplus a_i \odot g(r'_{i,j})$
6: $t \leftarrow t \oplus (r'_{i,j} \odot g(a_i))$ // $t = r_{i,j} \oplus f(a_i, r'_{i,j})$
7: $t \leftarrow t \oplus (a_i \odot g(a_j \oplus r'_{i,j}))$
8: $t \leftarrow t \oplus ((a_j \oplus r'_{i,j}) \odot g(a_i))$ // $t = r_{i,j} \oplus f(a_i, a_j)$
9: $r_{j,i} \leftarrow t$
10: **end for**
11: **end for**
12: **for** $i = 0$ **to** d **do**
13: $c_i \leftarrow a_i \odot g(a_i)$
14: **for** $j = 0$ **to** d, $j \neq i$ **do** $c_i \leftarrow c_i \oplus r_{i,j}$
15: **end for**

In Algorithm 4, the computation of $r_{j,i}$ involves four additions and four multiplications over $\mathbb{F}_{2^n}$. When n is small enough (e.g. $n = 4$), the multiplication over $\mathbb{F}_{2^n}$ can be tabulated in a look-up table with 2^{2n} n-bit elements. However for greater values of n (e.g. $n = 6$, $n = 8$), the size of such a table becomes prohibitive and other strategies must be considered to implement the multiplication. A typical choice is to use so-called log/alog tables (see for instance [3]), but the resulting multiplication is less efficient. We show hereafter how the bilinearity of f can be exploited to prevent such an efficiency loss.

Let us first introduce the function h mapping $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^n}$ and satisfying $h(x) = x \odot g(x)$. The $\mathbb{F}_2$-linearity of g then implies the following relation between f and h:

$$f(x, y) = h(x \oplus y) \oplus h(x) \oplus h(y),$$

for every $x, y \in \mathbb{F}_{2^n}$. Then (3) gives:

$$f(x, y) = h(x \oplus r \oplus y) \oplus h(x \oplus r) \oplus h(y \oplus r) \oplus h(r),$$

for every $x, y, r \in \mathbb{F}_{2^n}$. Our approach is then to store a look-up table for h and to compute $r_{j,i}$ as:

$$r_{j,i} = \big(\big(\big(r_{i,j} \oplus h(a_i \oplus r'_{i,j})\big) \oplus h(r'_{i,j} \oplus a_j)\big) \oplus h(a_i \oplus r'_{i,j} \oplus a_j)\big) \oplus h(r'_{i,j}).$$

We give the algorithmic description of our improved solution in Algorithm 5. Note that the use of brackets in Step 8 indicates the order in which the operations are processed.

Algorithm 5. Secure evaluation of $h : x \mapsto x \odot g(x)$

Input: shares $(a_i)_i$ such that $\bigoplus_i a_i = a$, a look-up table for $h : x \mapsto x \odot g(x)$
Output: shares c_i satisfying $\bigoplus_i c_i = a \odot g(a)$ for some $\mathbb{F}_2$-linear function g

for $i = 0$ **to** d **do**
  for $j = i + 1$ **to** d **do**
    $r_{i,j} \leftarrow^{\$} \mathbb{F}_{2^n}$
    $r'_{i,j} \leftarrow^{\$} \mathbb{F}_{2^n}$
    $t \leftarrow r_{i,j}$
    $t \leftarrow t \oplus h(a_i \oplus r'_{i,j})$
    $t \leftarrow t \oplus h(a_j \oplus r'_{i,j})$
    $t \leftarrow t \oplus h\big((a_i \oplus r'_{i,j}) \oplus a_j\big)$
    $t \leftarrow t \oplus h(r'_{i,j})$ // $t = f(a_i, a_j) \oplus r_{i,j}$
    $r_{j,i} \leftarrow t$
  end for
end for
for $i = 0$ **to** d **do**
  $c_i \leftarrow h(a_i)$
  for $j = 0$ **to** d, $j \neq i$ **do** $c_i \leftarrow c_i \oplus r_{i,j}$
end for

In Sect. 5, we provide implementation results to compare the above solution when plugged in the RP Scheme (Steps 1 to 3 and Steps 4 to 6 in Algorithm 3) to the original scheme using mask refreshing. We see that the scheme using our new solution is not only secure, but also faster than the original scheme.

The only drawback of the new scheme is to require more random generations. Specifically it needs $d(d+1)$ random field elements *versus* d (mask refreshing) plus $d(d+1)/2$ (secure multiplication) for the original scheme. However, the mask refreshing procedure involved in the original scheme is flawed and it is not clear whether it could be patched with less than $d(d+1)/2$ random field elements.

Security Proof. We prove hereafter that our solution achieves d^{th}-order security. Namely we show that any d-tuple of intermediate variables of Algorithm 4 is independent of the sensitive variable a. The (very similar) proof for Algorithm 5 is given in appendix.

Our proof consists in constructing a strict subset I of indices in $[0; d]$ such that the distribution of any d-tuple $(v_1, v_2, \ldots, v_d)$ of intermediate variables of Algorithm 4 can be perfectly simulated from $a_{|I} := (a_i)_{i \in I}$. This will prove the d^{th}-order security since, by definition, $a_{|I}$ is independent of a as long as the cardinality of I is strictly smaller than d.

By construction, it can first be checked that an intermediate variable v_h of Algorithm 4 necessarily belongs to one of the five following categories:

1. a_i, $g(a_i)$ and $a_i \odot g(a_i)$
2. $r'_{i,j}$, $g(r'_{i,j})$, $a_i \odot g(r'_{i,j})$ and $r'_{i,j} \odot g(a_i)$,
3. $a_j \oplus r'_{i,j}$, $g(a_j \oplus r'_{i,j})$, $a_i \odot g(a_j \oplus r'_{i,j})$ and $(a_j \oplus r'_{i,j}) \odot g(a_i)$

4. $r_{i,j}$, $f(a_i, r'_{i,j}) \oplus r_{i,j}$, $f(a_i, r'_{i,j}) \oplus a_i \odot g(a_j \oplus r'_{i,j}) \oplus r_{i,j}$ and $f(a_i, a_j) \oplus r_{i,j}$
5. $a_i \odot g(a_i) \oplus \bigoplus_{j=0}^{j_0}(f(a_j, a_i) \oplus r_{j,i})$ with $j_0 \leqslant i-1$ and
   $a_i \odot g(a_i) \oplus \bigoplus_{j=0}^{i-1}(f(a_j, a_i) \oplus r_{j,i}) \oplus \bigoplus_{j=i+1}^{j_0} r_{i,j}$ with $i < j_0 \leqslant d$

For the sake of clarity we use the notations $r_{i,j}$ and $r_{j,i}$ in the above list only for fresh random values (i.e. the $r_{i,j}$ are always such that $i < j$ and the $r_{j,i}$ are always such that $j < i$).

To construct the set I, we proceed as follows. Initially, I is empty. First, for every v_h in category 1 or 5, we add i to I. Then, for the remaining v_h (in categories 2, 3 and 4), we add j to I if i is already in I and we add i to I otherwise.

Now that the set I has been determined – and note that since there are at most d intermediate variables v_h, the cardinality of I can be at most d – we show how to perfectly simulate the d-tuple $(v_1, v_2, \ldots, v_d)$ using only the components of $a_{|I}$. First, we assign a random value to every $r_{i,j}$ and $r'_{i,j}$ entering in the computation of any v_h (as done in Steps 3 and 4 of Algorithm 4). Then every intermediate variable v_h is simulated as follows.

1. If v_h is in category 1, then $i \in I$ and v_h is directly computed from a_i.
2. If v_h is in category 2, then $i \in I$ and v_h is directly computed from a_i and $r'_{i,j}$.
3. If v_h is in category 3, then $i \in I$ and two possible cases occur:
   - if $j \in I$, then v_h can be directly assigned from a_i, a_j and $r'_{i,j}$,
   - if $j \notin I$, then $r'_{i,j}$ does not enter in the expression of any other v_h (otherwise j would be in I), and $a_j \oplus r'_{i,j}$ is randomly distributed and mutually independent of the variables in $\{v_1, v_2, \ldots, v_d\} \backslash \{v_h\}$. Hence v_h can be assigned to either r, $g(r)$, $a_i \odot g(r)$, or $r \odot g(a_i)$, where r is a fresh random value (and $r'_{i,j}$ does not need to be assigned to a random value at the beginning of the simulation).
4. If v_h is in category 4, then $i \in I$ and two possible cases occur:
   - if $j \in I$, then v_h can be directly assigned from a_i, a_j, $r_{i,j}$ and $r'_{i,j}$,
   - if $j \notin I$, then $r_{i,j}$ does not enter in the expression of any other v_h (otherwise j would be in I), and v_h is randomly distributed and mutually independent of the variables in $\{v_1, v_2, \ldots, v_d\} \backslash \{v_h\}$. Hence v_h can be assigned to a fresh random value (and $r_{i,j}$ does not need to be assigned to a random value at the beginning of the simulation).
5. If v_h is in category 5, then $i \in I$ and the firm term $a_i \odot g(a_i)$ is hence directly computed from a_i, whereas the second term $\bigoplus_{j=i+1}^{j_0} r_{i,j}$ is directly deduced from the $r_{i,j}$'s. Eventually, every element $f(a_j, a_i) \oplus r_{j,i}$ in the sum $\bigoplus_{j=0}^{i-1}(f(a_j, a_i) \oplus r_{j,i})$ is assigned as follows:
   - if $j \in I$ then $f(a_j, a_i) \oplus r_{j,i}$ is directly assigned from a_j, a_i and $r_{i,j}$,
   - if $j \notin I$ then $r_{j,i}$ does not enter in the expression of any other v_h (otherwise j would be in I), and $f(a_j, a_i) \oplus r_{j,i}$ is randomly distributed and mutually independent of the variables in the set $\{v_1, v_2, \ldots, v_d\} \backslash \{v_h\}$. Hence $f(a_j, a_i) \oplus r_{j,i}$ can be assigned to a fresh random value (and $r_{j,i}$ does not need to be assigned to a random value at the beginning of the simulation).

Table 1. Timings (clock cycles) for a masked implementation of the AES s-box w.r.t. the masking order d.

	d = 1	d = 2	d = 3
Rivain and Prouff scheme [10]	533	832	1905
Improved scheme (this paper)	407	622	1237

Table 2. Timings (clock cycles) for a masked implementation of a multiplication of the form $x \odot g(x)$ where g is $\mathbb{F}_2$-linear and with masking order $d \in \{1, 2, 3\}$.

	d = 1	d = 2	d = 3
Algorithm 1	110	252	346
Algorithm 5	41	107	204

5 Implementation Results

In this section, we give implementation results to compare, the original RP Scheme with our new proposal. We implemented the RP scheme (Algorithm 3) using Algorithm 5 for multiplications of the form $x \odot g(x)$ (i.e. for Steps 2–3 and Steps 4–6 in Algorithm 3) with the appropriate look-up tables (for h being $x \mapsto x \odot x^2$ and $x \mapsto x \odot x^4$ respectively). Codes were written in assembly language for an `8051` based 8-bit architecture with bit-addressable memory.
Table 1 lists the timing performances of the two versions of the scheme for $d \in \{1, 2, 3\}$. In Table 2, we also report on the timing performances of a secure multiplication of the form $x \odot g(x)$ when processed either with our new algorithm (Algorithm 5) or with the original ISW scheme (Algorithm 1). We see that our improved method achieves a significant gain in timings. Regarding memory, the RAM consumption is similar for both implementations, while our new secure multiplication requires more ROM for the storage of the look-up table. For the RP scheme, our new solution implies a 600-byte overhead in ROM to store the two look-up tables ($x \mapsto x \odot x^2$ and $x \mapsto x \odot x^4$) and for Algorithm 5 source code.

A Security Proof for Algorithm 5

Similarly to what has been done in Sect. 4 for Algorithm 4, we show here that any d-tuple of intermediate variables of Algorithm 5 is independent of the sensitive variable a.

Our proof consists in constructing a set I of indices in $[0; d]$ with cardinality lower than or equal to d and such that the distribution of any d-tuple $(v_1, v_2, \ldots, v_d)$ of intermediate variables of Algorithm 5 can be perfectly simulated from $a_{|I} := (a_i)_{i \in I}$. This will prove the d^{th}-order security, by definition, since $a_{|I}$ is independent of a as long as the cardinality of I is strictly smaller than d.

Let us first enumerate five possible categories for the intermediate variables of Algorithm 5:

1. a_i or $h(a_i)$;
2. $r'_{i,j}$, $a_i \oplus r'_{i,j}$, $h(r'_{i,j})$ or $h(a_i \oplus r'_{i,j})$;
3. $a_j \oplus r'_{i,j}$, $a_i \oplus r'_{i,j} \oplus a_j$, $h(a_j \oplus r'_{i,j})$ or $h(a_i \oplus r'_{i,j} \oplus a_j)$;
4. $r_{i,j}$, $r_{i,j} \oplus h(a_i \oplus r'_{i,j})$, $r_{i,j} \oplus h(a_i \oplus r'_{i,j}) \oplus h(a_j \oplus r'_{i,j})$, $r_{i,j} \oplus h(a_i \oplus r'_{i,j}) \oplus h(a_j \oplus r'_{i,j}) \oplus h(a_i \oplus r'_{i,j} \oplus a_j)$, or $r_{i,j} \oplus h(a_i \oplus r'_{i,j}) \oplus h(a_j \oplus r'_{i,j}) \oplus h(a_i \oplus r'_{i,j} \oplus a_j) \oplus h(r'_{i,j})$;
5. $h(a_i) \oplus \bigoplus_{j=0}^{j_0}(f(a_j, a_i) \oplus r_{j,i})$ with $j_0 \leqslant i-1$, or $h(a_i) \oplus \bigoplus_{j=0}^{i-1}(f(a_j, a_i) \oplus r_{j,i}) \oplus \bigoplus_{j=i+1}^{j_0} r_{i,j}$ with $j_0 \leqslant d$

For the sake of clarity we use the notations $r_{i,j}$ and $r_{j,i}$ in the above list only for fresh random values (i.e. the $r_{i,j}$ are always such that $i < j$ and the $r_{j,i}$ are always such that $j < i$).

To construct the set I, we proceed as follows. Initially, I is empty and all the v_h's are unassigned. First, for every v_h of category 1 or 5, we add i to I. Then, for every v_h of category 2, 3 or 4, if i is already in I, then we add j to I, otherwise we add i to I.

Now that the set I has been determined – and note that since there are at most d intermediate variables v_h, the cardinality of I can be at most d – we show how to complete a perfect simulation of the d-tuple $(v_1, v_2, \ldots, v_d)$ using only the values of $a_{|I}$. First, we assign a random value to every $r_{i,j}$ and $r'_{i,j}$ entering in the computation of any v_h (as done in steps 3 and 4 of Algorithm 5). Then every intermediate variable v_h is simulated as follows.

1. If v_h is of category 1, then $i \in I$ and v_h is directly assigned from a_i.
2. If v_h is of category 2, then $i \in I$ and v_h is directly assigned from a_i and $r'_{i,j}$.
3. If v_h is of category 3, then $i \in I$ and two possible cases occur:
   - if $j \in I$, then v_h can be directly assigned from a_i, a_j and $r'_{i,j}$,
   - if $j \notin I$, then $r'_{i,j}$ does not enter in the expression of any other v_h (otherwise j would be in I). Therefore $a_j \oplus r'_{i,j}$ (or $a_i \oplus r'_{i,j} \oplus a_j$) is randomly distributed and mutually independent of variables in $\{v_1, v_2, \ldots, v_d\} \backslash \{v_h\}$. Hence v_h can be assigned to either r or $h(r)$, where r is a fresh random value (and $r'_{i,j}$ does not need to be assigned to a random value at the beginning of the simulation).
4. If v_h is of category 4, then $i \in I$ and two possible cases occur:
   - if $j \in I$, then v_h can be directly assigned from a_i, a_j, $r_{i,j}$ and $r'_{i,j}$,
   - if $j \notin I$, then $r_{i,j}$ does not enter in the expression of any other v_h (otherwise j would be in I), and v_h is randomly distributed and mutually independent of variables in $\{v_1, v_2, \ldots, v_d\} \backslash \{v_h\}$. Hence v_h can be assigned to a fresh random value (and $r_{i,j}$ does not need to be assigned to a random value at the beginning of the simulation).
5. If v_h is of category 5, then $i \in I$, $h(a_i)$ is directly assigned from a_i, and $\bigoplus_{j=i+1}^{j_0} r_{i,j}$ is directly assigned from the $r_{i,j}$'s. Then for the sum $\bigoplus_{j=0}^{i-1}(f(a_j, a_i) \oplus r_{j,i})$, every $f(a_j, a_i) \oplus r_{j,i}$ is assigned as follows:

- if $j \in I$, then $f(a_j, a_i) \oplus r_{j,i}$ is directly assigned from a_j, a_i and $r_{i,j}$,
- if $j \notin I$, then $r_{j,i}$ does not enter in the expression of any other v_h (otherwise $j \in I$), and $f(a_j, a_i) \oplus r_{j,i}$ is randomly distributed and mutually independent of variables in $\{v_1, v_2, \ldots, v_d\} \backslash \{v_h\}$. Hence $f(a_j, a_i) \oplus r_{j,i}$ can be assigned to a fresh random value (and $r_{j,i}$ does not need to be assigned to a random value at the beginning of the simulation).

References

1. Carlet, C., Goubin, L., Prouff, E., Quisquater, M., Rivain, M.: Higher-order masking schemes for S-Boxes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 366–384. Springer, Heidelberg (2012)
2. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)
3. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer, Heidelberg (2002)
4. Gierlichs, B., Batina, L., Preneel, B., Verbauwhede, I.: Revisiting Higher-Order DPA Attacks: Multivariate Mutual Information Analysis. Cryptology ePrint Archive, report 2009/228 (2009). http://eprint.iacr.org/
5. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
6. Kim, H., Hong, S., Lim, J.: A fast and provably secure higher-order masking of AES S-Box. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 95–107. Springer, Heidelberg (2011)
7. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
8. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
9. Prouff, E., Rivain, M., Bévan, R.: Statistical analysis of second order differential power analysis. IEEE Trans. Comput. **58**(6), 799–811 (2009)
10. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
11. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A compact rijndael hardware architecture with S-Box optimization. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 239–254. Springer, Heidelberg (2001)
12. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)