Provably Secure Higher-Order Masking of AES*

Matthieu Rivain[1] and Emmanuel Prouff[2]

[1] CryptoExperts
matthieu.rivain@cryptoexperts.com
[2] Oberthur Technologies
e.prouff@oberthur.com

Abstract. Implementations of cryptographic algorithms are vulnerable to Side Channel Analysis (SCA). To counteract it, masking schemes are usually involved which randomize key-dependent data by the addition of one or several random value(s) (the *masks*). When dth-order masking is involved (*i.e.* when d masks are used per key-dependent variable), the complexity of performing an SCA grows exponentially with the order d. The design of generic dth-order masking schemes taking the order d as security parameter is therefore of great interest for the physical security of cryptographic implementations. This paper presents the first generic dth-order masking scheme for AES with a provable security and a reasonable software implementation overhead. Our scheme is based on the hardware-oriented masking scheme published by Ishai *et al.* at Crypto 2003. Compared to this scheme, our solution can be efficiently implemented in software on any general-purpose processor. This result is of importance considering the lack of solution for $d \geqslant 3$.

1 Introduction

Side Channel Analysis 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 data manipulated by the implementation. Some of these data are *sensitive* in the sense that they are related to the secret key, and the leaking information about them enables efficient key-recovery attacks [7, 19].

Due to the very large variety of side channel attacks reported against cryptosystems and devices, important efforts have been done to design countermeasures with provable security. They all start from the assumption that a cryptographic device can keep at least some secrets and that only computation leaks [25]. Based on these assumptions, two main approaches have been followed. The first one consists in designing new cryptographic primitives inherently resistant to side channel attacks. In [25], a very powerful side channel adversary is considered who has access to the whole internal state of the ongoing computation. In such a model, the authors show that if a *physical* one-way permutation exists which does not leak any information, then it can be used in the pseudo-random number generator (PRNG) construction proposed in [4] to give a PRNG provably secure against the aforementioned side channel adversary. Unfortunately, no such leakage-resilient one-way permutation is known at this day. Besides, the obtained construction is quite inefficient since each computation of the one-way permutation produces one single random bit. To get more practical constructions, further works focused on designing primitives secure against a *limited* side channel adversary [13]. The definition of such a limited adversary is inspired by the *bounded retrieval model* [10, 22] which assumes that the device leaks a limited amount of information about its internal state for each elementary computation. In such a setting, the block cipher based PRNG construction proposed in [30] is provably secure assuming that the underlying cipher is *ideal*. Other

* Full version of the paper published in the proceedings of CHES 2010.

constructions were proposed in [13,31] which do not require such a strong assumption but are less efficient [40]. The main limitations of these constructions is that they do not enable the choice of an initialization vector (otherwise the security proofs do not hold anymore) which prevents their use for encryption with synchronization constraints or for challenge-response protocols [40]. Moreover, as they consist in new constructions, these solutions do not allow for the protection of the implementation of standard algorithms such as DES or AES [14,15].

The second approach to design countermeasures provably secure against side channel attacks consists in applying *secret sharing schemes* [2,39]. In such schemes, the sensitive data is randomly split into several shares in such a way that a chosen number (called the *threshold*) of these shares is required to retrieve any information about the data. When the SCA threat appeared, secret sharing was quickly identified as a pertinent protection strategy [6, 17] and numerous schemes (often called *masking schemes*) were published that were based on this principle (see for instance [1,3,18,23,26,29,34,38]). Actually, this approach is very close to the problem of defining Multi Party Communication (MPC) schemes (see for instance [9,12]) but the resources and constraints differ in the two contexts (*e.g.* MPC schemes are often based on a *trusted dealer* who does not exist in the SCA context). A first advantage of this approach is that it can be used to secure standard algorithms such as DES and AES. A second advantage is that *dth-order masking schemes*, for which sensitive data are split into $d + 1$ shares (the threshold being $d + 1$), are sound countermeasures to SCA in *realistic leakage model*. This fact has been formally demonstrated by Chari *et al.* [6] who showed that the complexity of recovering information by SCA on a bit shared into several pieces grows exponentially with the number of shares. As a direct consequence of this work, the number of shares (or equivalently of masks) in which sensitive data are split is a sound security parameter of the resistance of a countermeasures against SCA.

The present paper deals with the problem of defining an efficient masking scheme to protect the implementation of the AES block cipher [11]. Until now, most of works published on this subject have focused on first-order masking schemes where sensitive variables are masked with a single random value (see for instance [1, 3, 23, 26, 29]). However, this kind of masking have been shown to be efficiently breakable in practice by *second-order SCA* [24,27,42]. To counteract those attacks, *higher-order masking schemes* must be used but a very few have been proposed. A first method has been introduced by Ishai *et al.* [18] which enables to protect an implementation at any chosen order. Unfortunately, it is not suited for software implementations and it induces a prohibitive overhead for hardware implementations. A scheme devoted to secure the software implementation of AES at any chosen order has been proposed by Schramm and Paar [38] but it was subsequently shown to be secure only in the second-order case [8]. Alternative second-order masking schemes with provable security were further proposed in [34], but no straightforward extension of them exist to get efficient and secure masking scheme at any order. Actually, at this day, no method exists in the literature that enables to mask an AES implementation at any chosen order $d \geqslant 3$ with a practical overhead; the present paper fills this gap.

2 Preliminaries on Higher-Order Masking

2.1 Basic Principle

When higher-order masking is involved to secure the physical implementation of a cryptographic algorithm, every sensitive variable x occurring during the computation is randomly split into $d+1$ shares $x_0, \ldots, x_d$ in such a way that the following relation is satisfied for a group operation $\perp$:

$$x_0 \perp x_1 \perp \cdots \perp x_d = x \ . \tag{1}$$

In the rest of the paper, we shall consider that $\perp$ is the exclusive-or (XOR) operation denoted by $\oplus$. 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 (1). When d random masks are involved per sensitive variable the masking is said to be *of order d*.

Assuming that the masks are uniformly distributed, masking renders every intermediate variable of the computation statistically independent of any sensitive variable. As a result, classical side channel attacks exploiting the leakage related to a single intermediate variable are not possible anymore. However, a dth-order masking is always theoretically vulnerable to $(d+1)$*th-order SCA* which exploits the leakages related to $d+1$ intermediate variables at the same time [24, 37, 38]. Indeed, the leakages resulting from the $d+1$ shares (i.e. the masked variable and the d masks) are jointly dependent on the sensitive variable. Nevertheless, such attacks become impractical as d increases, which makes higher-order masking a sound countermeasure.

2.2 Soundness of Higher-Order Masking

The soundness of higher-order masking was formally demonstrated by Chari *et al.* in [6]. They assume a simplified but still realistic leakage model where a bit b is masked using d random bits $x_1, \ldots, x_d$ such that the masked bit is defined as $x_0 = b \oplus x_1 \oplus \cdots \oplus x_d$. The adversary is assumed to be provided with observations of $d+1$ leakage variables L_i, each one corresponding to a share x_i. For every i, the leakage is modelled as $L_i = x_i + N_i$ where the noises N_i's are assumed to have Gaussian distributions $\mathcal{N}(\mu, \sigma^2)$ and to be mutually independent. Under this leakage model, they show that the number of samples q required by the adversary to distinguish the distribution $(L_0, \ldots, L_d | b = 0)$ from the distribution $(L_0, \ldots, L_d | b = 1)$ with a probability at least α satisfies:

$$q \geqslant \sigma^{d+\delta} \tag{2}$$

where $\delta = 4 \log \alpha / \log \sigma$. This result encompasses all the possible side-channel distinguishers and hence formally states the resistance against every kind of side channel attack. Although the model is simplified, it could probably be extended to more common leakage models such as the Hamming weight/distance model. The point is that if an attacker observes noisy side channel information about $d+1$ shares corresponding to a variable masked with d random masks, the number of samples required to retrieve information about the unmasked variable is lower bounded by an exponential function of the masking order whose base is related to the noise standard deviation. This formally demonstrates that higher-order masking is a sound countermeasure especially when combined with noise. Many works also made this observation in practice for particular side channel distinguishers (see for instance [37, 38, 41]).

2.3 Higher-Order Masking Schemes

When dth-order masking is involved in protecting a block cipher implementation, a so-called *dth-order masking scheme* (or simply a *masking scheme* if there is no ambiguity on d) must be designed to enable computation on masked data. In order to be complete and secure, the scheme must satisfy the two following properties:

- *completeness*: at the end of the computation, the sum of the d shares must yield the expected ciphertext (and more generally each masked transformation must result in a set of shares whose sum equal the correct intermediate result),
- *dth-order SCA security*: every tuple of d or less intermediate variables must be independent of any sensitive variable.

If the dth-order security property is satisfied, then no attack of order lower than $d + 1$ is possible and we benefit from the security bound (2).

Most block cipher structures (*e.g.* AES or DES) alternate several rounds composed of a key addition, one or several linear transformation(s), and a non-linear transformation. The main difficulty in designing masking schemes for such block ciphers lies in masking the nonlinear transformations. Many solutions have been proposed to deal with this issue but the design of a dth-order secure scheme for $d > 1$ has quickly been recognized as a difficult problem by the community. As mentioned above, only three methods exist in the literature that have been respectively proposed by Ishai, Sahai and Wagner [18], by Schramm and Paar [38] (secure only for $d \leqslant 2$) and by Rivain, Dottax and Prouff [34] (dedicated to $d = 2$). Among them, only [18] can be applied to secure a non-linear transformation at any order d. This scheme is recalled in the next section.

2.4 The Ishai-Sahai-Wagner Scheme

In [18], Ishai *et al.* propose a higher-order masking scheme (referred to as ISW in this paper) enabling to secure the hardware implementation of any *circuit* at any chosen order d. They describe a way to transform the circuit to protect into a new circuit (dealing with masked values) such that no subset of d of its *wires* reveals information about the unmasked values[3]. For such a purpose, they assume without loss of generality that the circuit to protect is exclusively composed of NOT and AND gates. Securing a NOT for any order d is straightforward since $x = \bigoplus_i x_i$ implies $\mathrm{NOT}(x) = \mathrm{NOT}(x_0) \oplus x_1 \cdots \oplus x_d$. The main difficulty is therefore to secure the AND gates. To answer this issue, Ishai *et al.* suggest the following elegant solution.

Secure logical AND. Let a an b be two bits and let c denote $\mathrm{AND}(a, b) = ab$. Let us assume that a and b have been respectively split into $d+1$ shares $(a_i)_{0 \leqslant i \leqslant d}$ and $(b_i)_{0 \leqslant i \leqslant d}$ such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$. To securely compute a $(d + 1)$-tuple $(c_i)_{0 \leqslant i \leqslant d}$ s.t. $\bigoplus_i c_i = c$, Ishai *et al.* perform the following steps:

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 $c_i = a_i b_i \oplus \bigoplus_{j \neq i} r_{i,j}$.

[3] Considering wires as intermediate variables, this is equivalent to the security property given in Section 2.3.

Remark 1. The use of brackets indicates the order in which the operations are performed, which is mandatory for security of the scheme.

The completeness of the solution follows from:

$$\bigoplus_i c_i = \bigoplus_i \left(a_i b_i \oplus \bigoplus_{j \neq i} r_{i,j}\right) = \bigoplus_i \left(a_i b_i \oplus \bigoplus_{j>i} r_{i,j} \oplus \bigoplus_{j<i} (r_{j,i} \oplus a_i b_j \oplus a_j b_i)\right)$$
$$= \bigoplus_i \left(a_i b_i \oplus \bigoplus_{j<i} (a_i b_j \oplus a_j b_i)\right) = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) \ .$$

In [18] it is shown that the AND computation above is secure against any attack of order lower than or equal to $d/2$. In Section 4, we give a tighter security proof: we show that the scheme is actually dth-order secure.

Practical issues. Although the ISW scheme is an important theoretical result, its practical application suffers few issues. Firstly, it induces an important overhead in silicon area for the masked circuit. Indeed, every single AND gate is encoded using $(d+1)^2$ AND gates plus $2d(d+1)$ XOR gates, and it requires the generation of $d(d+1)/2$ random bits at every clock cycle. As an illustration, masking the compact circuit for the AES S-box described in [5] would multiply its size (in terms of number of gates) by 7 for $d = 2$, by 14 for $d = 3$ and by 22 for $d = 4$ (without taking the random bits generation into account). Secondly, masking at the hardware level is sensitive to glitches, which induces first-order flaws although in theory every internal wire carries values that are independent of the sensitive variables [20,21]. Preventing glitches in masked circuits imply the addition of synchronizing elements (*e.g.* registers or latches) which still significantly increases the circuit size (see for instance [32]).

Since software implementations of masking schemes do not suffer area overhead and are not impacted by the presence of glitches at the hardware level, a straightforward approach to deal with the practical issues discussed above could be to implement the ISW scheme in software. Namely, we could represent each non-linear transformation S to protect by a tuple of Boolean functions $(f_i)_i$ usually called *coordinate functions* of S, and evaluate the f_i's with the ISW scheme by processing the AND and XOR operations with CPU instructions. However, this approach is not practical since the timing overhead would clearly be prohibitive. The present paper follows a different approach: we generalize the ISW scheme to secure any finite field multiplication rather than a simple multiplication over $\mathbb{F}_2$ (*i.e.* a logical AND). We apply this idea to design a secure higher-order masking scheme for the AES and we show that its software implementation induces a reasonable overhead.

3 Higher-Order Masking of AES

The AES block cipher iterates a round transformation composed of a key addition, a linear layer and a nonlinear layer which applies the same substitution-box (S-box) to every byte of the internal state. As previously explained, the main difficulty while designing a masking scheme for such a cipher is the masking of the nonlinear transformation, which in that case lies in the masking of the S-box. Our method for masking the AES S-box is presented in the next section, afterward the masking of the whole cipher is described.

In what follows, we shall consider that a random generator is available which on an invocation $\mathsf{rand}(n)$ returns n unbiased random bits.

3.1 Higher-Order Masking of the AES S-box

The AES S-box S is defined as the right-composition of an affine transformation Af over $\mathbb{F}_2^8$ with the power function $x \mapsto x^{254}$ over the field $\mathbb{F}_{2^8} \equiv \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$. Since the affine transformation is straightforward to mask, our scheme mainly consists in a method for masking the power function at any order d. Our solution consists in a secure computation of the exponentiation to the power 254 over $\mathbb{F}_{2^8}$. Such an approach has already been described by Blömer *et al.* for $d = 1$ [3]. The core idea is to apply an exponentiation algorithm (*e.g.* the square-and-multiply algorithm) on the first-order masked input while ensuring the mask correction step by step. Compared to Blömer *et al.* 's solution, our exponentiation algorithm is able to operate on dth-order masked inputs and it achieves dth-order SCA security for any value of d. To perform such a secure exponentiation, we define hereafter some methods to securely compute a squaring and a multiplication over $\mathbb{F}_{2^8}$ at the dth order.

Masking the field squaring. Since we operate on a field of characteristic 2, the squaring is a linear operation and we have $x_0^2 \oplus x_1^2 \oplus \cdots \oplus x_d^2 = x^2$. Securely computing a squaring can hence be carried out by squaring every share separately. More generally, for every natural integer j, raising x to the power 2^j can be done securely by raising each x_i to the 2^j separately.

Masking the field multiplication. For the usual field multiplication we use the ISW scheme recalled in Section 2.4. Even if it has been described to securely compute a logical AND (that is a multiplication over $\mathbb{F}_2$), it can actually be transposed to secure a multiplication over any field of characteristic 2: variables over $\mathbb{F}_2$ are replaced by variables over $\mathbb{F}_{2^n}$, binary multiplications (*i.e.* ANDs) are replaced by multiplications over $\mathbb{F}_{2^n}$ and binary additions (*i.e.* XORs) are replaced by addition over $\mathbb{F}_{2^n}$ (that are n-bit XORs). This keep unchanged the completeness of the scheme recalled in Section 2.4. The whole secure multiplication over $\mathbb{F}_{2^n}$ is depicted in the following algorithm.

Algorithm 1 SecMult - dth-order secure multiplication over $\mathbb{F}_{2^n}$

INPUT: shares a_i satisfying $\bigoplus_i a_i = a$, shares b_i satisfying $\bigoplus_i b_i = b$
OUTPUT: shares c_i satisfying $\bigoplus_i c_i = ab$

1. **for** $i = 0$ **to** d **do**
2. **for** $j = i + 1$ **to** d **do**
3. $r_{i,j} \leftarrow \mathsf{rand}(n)$
4. $r_{j,i} \leftarrow (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$
5. **for** $i = 0$ **to** d **do**
6. $c_i \leftarrow a_i b_i$
7. **for** $j = 0$ **to** d, $j \neq i$ **do** $c_i \leftarrow c_i \oplus r_{i,j}$

Masking the power function. Now we have a secure squaring and a secure multiplication over $\mathbb{F}_{2^8}$ it remains to specify an exponentiation algorithm. It is clear from Algorithm 1 that the running time of a secure multiplication is huge compared to that of a secure squaring. A secure squaring indeed requires $d + 1$ squarings while a secure multiplication requires $(d+1)^2$ field multiplications, $2d(d+1)$ XORs and the generation of $d(d+1)/2$ random 8-bit values. Our goal is therefore to design an exponentiation algorithm using the least possible multiplications which are not squares. It can be checked that an exponentiation to the power

254 requires at least 4 such multiplications. The exponentiation algorithm presented hereafter achieves this lower bound and requires few additional squares. It involves three intermediate variables denoted y, z and w (note that x and y may be associated to the same memory address).

Algorithm 2 Exponentiation to the 254

INPUT: x
OUTPUT: $y = x^{254}$

1. $z \leftarrow x^2$	$[z = x^2]$
2. $y \leftarrow zx$	$[y = x^2 x = x^3]$
3. $w \leftarrow y^4$	$[w = (x^3)^4 = x^{12}]$
4. $y \leftarrow yw$	$[y = x^3 x^{12} = x^{15}]$
5. $y \leftarrow y^{16}$	$[y = (x^{15})^{16} = x^{240}]$
6. $y \leftarrow yw$	$[y = x^{240} x^{12} = x^{252}]$
7. $y \leftarrow yz$	$[y = x^{252} x^2 = x^{254}]$

As we will argue in Section 4, , for the dth-order security to hold, it is important that the masks $(a_i)_{i \geqslant 1}$ and $(b_i)_{i \geqslant 1}$ in input of the SecMult algorithm are mutually independent. That is why we shall refresh the masks at some points during the secure exponentiation by calling a procedure RefreshMasks[4]. The whole exponentiation to the power 254 over $\mathbb{F}_{2^8}$ secure against dth-order SCA is depicted in the following algorithm.

Algorithm 3 SecExp254 - dth-order secure exponentiation to the 254 over $\mathbb{F}_{2^8}$

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. RefreshMasks$(z_0, z_1, \ldots, z_d)$	
3. $(y_0, y_1, \ldots, y_d) \leftarrow$ SecMult$\big((z_0, z_1, \ldots, z_d), (x_0, x_1, \ldots, x_d)\big)$	$[\bigoplus_i y_i = x^3]$
4. **for** $i = 0$ **to** d **do** $w_i \leftarrow y_i^4$	$[\bigoplus_i w_i = x^{12}]$
5. RefreshMasks$(w_0, w_1, \ldots, w_d)$	
6. $(y_0, y_1, \ldots, y_d) \leftarrow$ 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. $(y_0, y_1, \ldots, y_d) \leftarrow$ SecMult$\big((y_0, y_1, \ldots, y_d), (w_0, w_1, \ldots, w_d)\big)$	$[\bigoplus_i y_i = x^{252}]$
9. $(y_0, y_1, \ldots, y_d) \leftarrow$ SecMult$\big((y_0, y_1, \ldots, y_d), (z_0, z_1, \ldots, z_d)\big)$	$[\bigoplus_i y_i = x^{254}]$

For completeness, we describe the RefreshMasks algorithm hereafter.

Algorithm 4 RefreshMasks

INPUT: shares x_i satisfying $\bigoplus_i x_i = x$
OUTPUT: shares x_i satisfying $\bigoplus_i x_i = x$

```
1. for i = 1 to d do
2.     tmp ← rand(8)
3.     x_0 ← x_0 ⊕ tmp
4.     x_i ← x_i ⊕ tmp
```

Algorithm 3 involves of $8d(d+1) + 4d$ XORs, $4(d+1)^2$ multiplications (over $\mathbb{F}_{2^8}$), $d+1$ squares, $d+1$ raising to the 4 and $d+1$ raising to the 16. It uses $3(d+1) + d(d+1)/2$ bytes

[4] Note that the masks resulting from the SecMult algorithm are independent of the input masks.

Table 1. Complexity of SecExp254.

order	nb. XORs	nb. mult.	nb. $\hat{}2^j$	nb. rand. bytes	memory (bytes)
1	20	16	6	6	7
2	56	36	9	16	12
3	108	64	12	20	18
4	176	100	15	48	25
5	260	144	18	70	33
d	$8d^2+12d$	$4d^2+8d+4$	$3d+3$	$2d^2+4d$	$\frac{1}{2}d^2+\frac{7}{2}d+3$

of memory[5] and it requires the generation of $2d(d+1)+2d$ random bytes (see illustrative values in Table 1). In comparison, the 2nd-order countermeasures previously published [34,38] require at least 512 look-ups and 512 XORs and have a memory consumption of at least 256 bytes (see [33, 35] for a detailed comparison).

Masking the full S-box. The affine transformation is straightforward to mask. After recalling that the additive part of Af equals `0x63`, it can be checked that we have:

$$Af(x_0) \oplus Af(x_1) \oplus \cdots \oplus Af(x_d) = \begin{cases} Af(x) & \text{if } d \text{ is even,} \\ Af(x) \oplus \texttt{0x63} & \text{if } d \text{ is odd.} \end{cases}$$

Masking the affine transformation hence simply consists in applying it to every input share separately and, in case of an even d, in adding `0x63` to one of the share afterward. The full S-box computation secure against dth-order SCA is summarized in the following algorithm.

Algorithm 5 SecSbox

INPUT: shares x_i satisfying $\bigoplus_i x_i = x$
OUTPUT: shares y_i satisfying $\bigoplus_i y_i = \mathrm{S}(x)$

1. $(y_0, \ldots, y_d) \leftarrow \mathsf{SecExp254}(x_0, \ldots, x_d)$
2. **for** $i = 0$ **to** d **do** $y_i \leftarrow Af(y_i)$
3. **if** ($d \bmod 2 = 1$) **then** $y_0 \leftarrow y_0 \oplus \texttt{0x63}$

Implementation aspects. Multiplications over $\mathbb{F}_{2^8}$ are typically implemented in software using log/alog tables (see for instance [11]). Note that for security reasons, such an implementation must avoid conditional branches in order to ensure a constant operation flow. The squaring and raisings to the 4 and 16 may be looked-up. Different time-memory tradeoffs are possible. If not much ROM is available, the squaring can be implemented using logical shifts and XORs (see for instance [11]), and the raising to the 2^j, $j \in \{2, 4\}$, can then be simply processed by j sequential squarings. Otherwise, depending on the amount of ROM available, one can either use one, two or three look-up table(s) to implement the raisings to 2^j, $j \in \{1, 2, 4\}$.

Remark 2. For the implementations presented in Section 5, we chose to implement the squaring by a look-up table, getting the raising to the 4 (resp. 16) by accessing this table sequentially 2 (resp. 4) times.

[5] $3(d+1)$ bytes for the shares y_i's, z_i's and w_i's (Algorithm 3), and $d(d+1)/2$ for the intermediate variables $r_{i,j}$'s (Algorithm 1).

Our scheme may also be implemented in hardware. The sensitive part is the implementation of the SecMult algorithm (see Algorithm 1) which may be subject to glitches and which should incorporate synchronizing elements. In particular, the evaluation of the c_i shares should not start before the evaluation of all the $r_{i,j}$'s has been fully completed. Another approach would be to enhance the software implementation of the scheme with special purpose hardware instructions. For instance, the multiplication, squaring and raisings to powers 4 and 16 over $\mathbb{F}_{2^8}$ could be added to the instructions set of the processor.

3.2 Higher-Order Masking of the Whole Cipher

In the previous section, we have shown how the AES S-box can be masked at any chosen order d. We now detail the dth-order masking scheme for the whole AES block cipher.

The AES block cipher [11] operates on a 4×4 array of bytes called the state and denoted $\mathbf{s} = (s_{l,j})_{1 \leqslant l,j \leqslant 4}$. The state is initialized by the plaintext value and holds the ciphertext value at the end of the encryption. Each round of AES is composed of four stages: AddRoundKey, SubBytes, ShiftRows, and MixColumns (except the last round that omits the MixColumns). AES is composed of either 10, 12 or 14 rounds, depending on the key length (the longer the key, the higher the number of rounds) plus a final AddRoundKey stage. The round keys involved in the different rounds are derived from the secret key through a key expansion process.

In what follows, we describe how to mask an AES computation at the dth order. We will assume that the secret key has been previously masked and that its $d+1$ shares are provided as input to the algorithm (otherwise a straightforward first-order attack would be possible). At the beginning of the computation, the state (holding the plaintext) is split into $d+1$ states $\mathbf{s}_0$, $\mathbf{s}_1$, ..., $\mathbf{s}_d$ satisfying:

$$\mathbf{s} = \mathbf{s}_0 \oplus \mathbf{s}_1 \oplus \cdots \oplus \mathbf{s}_d \ .$$

This is done by generating d random states $\mathbf{s}_i \leftarrow \mathsf{rand}(16 \times 8)$ and by computing $\mathbf{s}_0 \leftarrow \mathbf{s} \oplus \bigoplus_{i \geqslant 1} \mathbf{s}_i$. At the end of the AES computation, the state (holding the ciphertext) is recovered by $\mathbf{s} \leftarrow \bigoplus_i \mathbf{s}_i$.

In the next sections, we describe how to perform the different AES transformations on the state shares in order to guarantee the completeness as well as the dth-order security.

Masking AddRoundKey. The AddRoundKey stage at round r consists in adding (by XOR) the rth round key $\mathbf{k}^r$ to the state. The masked key expansion (see description hereafter) provides $d+1$ shares $(\mathbf{k}_i^r)_i$ for every round key $\mathbf{k}^r$. To securely process the addition of $\mathbf{k}^r$, one simply adds each of its share to one share of the state and the completeness holds from:

$$\mathbf{s} \oplus \mathbf{k}^r = (\mathbf{s}_0 \oplus \mathbf{k}_0^r) \oplus (\mathbf{s}_1 \oplus \mathbf{k}_1^r) \oplus \cdots \oplus (\mathbf{s}_d \oplus \mathbf{k}_d^r) \ .$$

Masking SubBytes. The SubBytes transformation consists in applying the AES S-box S to each byte of the state:

$$\mathsf{SubBytes}(\mathbf{s}) = (\mathrm{S}(s_{l,j}))_{1 \leqslant l,j \leqslant 4} \ .$$

In order to mask this transformation, we apply the secure S-box computation described in Section 3.1 to the $(d+1)$-tuple of byte shares $((\mathbf{s}_0)_{l,j}, (\mathbf{s}_1)_{l,j}, \ldots, (\mathbf{s}_d)_{l,j})$ for every row-coordinate $l \in [1,4]$ and for every column-coordinate $j \in [1,4]$.

Masking ShiftRows and MixColumns. The ShiftRows and MixColumns transformations compose the linear layer of AES. In the ShiftRows transformation, the bytes in the last three rows of the state are cyclically shifted over different numbers of bytes (1 for the second row, 2 for the third row and 3 for the fourth row). The MixColumns transformation operates on the state column-by-column. Each column is treated as a four-term polynomial over $\mathbb{F}_2[x]/(x^8 + x^4+x^3+x+1)$ and is multiplied modulo x^4+1 with a fixed polynomial $a(x) = 3x^3+x^2+x+2$. Since they are both linear with respect to the XOR operation, masking these transformations is straightforward. One just apply them to every state share separately and the completeness holds from:

$$\mathsf{ShiftRows}(\mathbf{s}) = \bigoplus_{i=0}^{d} \mathsf{ShiftRows}(\mathbf{s}_i) \ ,$$

and:

$$\mathsf{MixColumns}(\mathbf{s}) = \bigoplus_{i=0}^{d} \mathsf{MixColumns}(\mathbf{s}_i).$$

Masking the key expansion. The AES key expansion generates a $4 \times 4(\mathrm{Nr} + 1)$ array of bytes $\mathbf{w}$, called the key schedule, where Nr is the number of rounds (which depends on the key-length). Let $\mathbf{w}_{*,j}$ denotes the jth column of $\mathbf{w}$. Each group of 4 columns $(\mathbf{w}_{*,4r-3}, \mathbf{w}_{*,4r-2}, \mathbf{w}_{*,4r-1}, \mathbf{w}_{*,4r})$ forms a round key $\mathbf{k}^r$ that is XORed to the state during the rth AddRoundKey stage. The first Nk columns of the key schedule are filled with the key bytes (where the key byte-length is 4Nk) and the next ones are derived according to the process described hereafter.

Let SubWord be the transformation that takes a four-byte input column and applies the AES S-box to each byte. Let RotWord be the transformation that takes a 4-byte column as input and performs a cyclic shift of one byte from bottom to top. Finally, let $\mathtt{Rcon}_j$ denotes the constant 4-bytes column $(\{02\}^{j-1}, 0, 0, 0)^T$, where $\{02\}^{j-1}$ is the $(j-1)$th power of x in the field $\mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$. The jth column of the key schedule $\mathbf{w}_{*,j}$ is defined as:

$$\mathbf{w}_{*,j} = \mathbf{w}_{*,j-\mathrm{Nk}} \oplus \mathbf{t}$$

with:

$$\mathbf{t} = \begin{cases} \mathsf{RotWord}(\mathsf{SubWord}(\mathbf{w}_{*,j-1})) \oplus \mathtt{Rcon}_{j/\mathrm{Nk}} & \text{if } (j \bmod \mathrm{Nk} = 0), \\ \mathsf{SubWord}(\mathbf{w}_{*,j-1}) & \text{if } (\mathrm{Nk} = 8) \text{ and } (j \bmod \mathrm{Nk} = 4), \\ \mathbf{w}_{*,j-1} & \text{otherwise.} \end{cases}$$

In order to securely process the key expansion at the dth-order, the key schedule $\mathbf{w}$ is split into $d+1$ schedules $\mathbf{w}_0$, $\mathbf{w}_1$, ..., $\mathbf{w}_d$. The first columns of each schedule shares are filled with the key shares at the beginning of the ciphering. Each time a new schedule column $\mathbf{w}_{*,j}$ must be computed, its $d+1$ shares $(\mathbf{w}_0)_{*,j}$, $(\mathbf{w}_1)_{*,j}$, ..., $(\mathbf{w}_d)_{*,j}$ are computed as:

$$(\mathbf{w}_i)_{*,j} = (\mathbf{w}_i)_{*,j-\mathrm{Nk}} \oplus \mathbf{t}_i$$

where the $\mathbf{t}_i$'s denote the 4-bytes shares of $\mathbf{t}$ that are securely computed from the 4-bytes shares of $\mathbf{w}_{*,j-1}$. Such a secure computation can be easily deduced from the methods described above. The SubWord transformation is processed by applying the secure S-box computation described in Section 3.1 to the byte shares $(\mathbf{w}_0)_{l,j}$, $(\mathbf{w}_1)_{l,j}$, ..., $(\mathbf{w}_d)_{l,j}$ for each row-coordinate

$l \in [1,4]$. Since RotWord is linear with respect to the XOR, it is applied (when involved) to every share separately. Finally, when $\texttt{Rcon}_{j/\mathrm{Nk}}$ must be added to $\mathbf{t}$, it is added to one of its share (*e.g.* $\mathbf{t}_0$).

The whole dth-order secure key expansion process is summarized in the following algorithm.

Algorithm 6 dth-order secure AES key expansion
INPUT: key shares $\mathbf{k}_i$ satisfying $\bigoplus_i \mathbf{k}_i = \mathbf{k}$
OUTPUT: shares $\mathbf{w}_i$ satisfying $\bigoplus_i \mathbf{w}_i = \mathbf{w}$

1. **for** $j = 1$ **to** Nk **do**
2. **for** $i = 0$ **to** d **do** $(\mathbf{w}_i)_{*,j} \leftarrow (\mathbf{k}_i)_{*,j}$
3. **for** $j = \mathrm{Nk}+1$ **to** $4(\mathrm{Nr}+1)$ **do**
4. **for** $i = 0$ **to** d **do** $\mathbf{t}_i \leftarrow (\mathbf{w}_i)_{*,j-1}$
5. **if** $\big((j \bmod \mathrm{Nk} = 0)$ or $(\mathrm{Nk} = 8)$ and $(j \bmod \mathrm{Nk} = 4)\big)$ **then**
6. **for** $l = 1$ **to** 4 **do** $\big((\mathbf{t}_0)_l, (\mathbf{t}_1)_l, \ldots, (\mathbf{t}_d)_l\big) \leftarrow \mathsf{SecSbox}\big((\mathbf{t}_0)_l, (\mathbf{t}_1)_l, \ldots, (\mathbf{t}_d)_l\big)$
7. **if** $(j \bmod \mathrm{Nk} = 0)$ **then**
8. **for** $i = 0$ **to** d **do** $\mathbf{t}_i \leftarrow \mathsf{RotWord}(\mathbf{t}_i)$
9. $\mathbf{t}_0 \leftarrow \mathbf{t}_0 \oplus \texttt{Rcon}_{j/\mathrm{Nk}}$
10. **for** $i = 0$ **to** d **do** $(\mathbf{w}_i)_{*,j-1} \leftarrow (\mathbf{w}_i)_{*,j-\mathrm{Nk}} \oplus \mathbf{t}_i$

Remark 3. Note that the key expansion can be executed on-the-fly during the AES computation in order to avoid the storage of all the round keys.

Masking the whole AES: algorithmic description. Algorithm 7 summarizes the whole AES computation secure against dth-order SCA.

Algorithm 7 dth-order secure AES computation
INPUT: plaintext $\mathbf{p}$, key shares $\mathbf{k}_i$ satisfying $\bigoplus_i \mathbf{k}_i = \mathbf{k}$
OUTPUT: ciphertext $\mathbf{c}$

1. $\mathbf{s}_0 \leftarrow \mathbf{p}$

*** State masking ***
2. **for** $i = 0$ **to** d **do**
3. $\mathbf{s}_i \leftarrow \mathsf{rand}(16 \times 8)$
4. $\mathbf{s}_0 \leftarrow \mathbf{s}_0 \oplus \mathbf{s}_i$

*** All but last rounds ***
5. **for** $r = 1$ **to** $\mathrm{Nr} - 1$ **do**
6. **for** $i = 0$ **to** d **do** $\mathbf{s}_i \leftarrow \mathbf{s}_i \oplus \mathbf{k}_i^r$
7. **for** $l = 1$ **to** 4, $j = 1$ **to** 4 **do**
8. $\big((\mathbf{s}_0)_{l,j}, (\mathbf{s}_1)_{l,j}, \ldots, (\mathbf{s}_d)_{l,j}\big) \leftarrow \mathsf{SecSbox}\big((\mathbf{s}_0)_{l,j}, (\mathbf{s}_1)_{l,j}, \ldots, (\mathbf{s}_d)_{l,j}\big)$
9. **for** $i = 0$ **to** d **do** $\mathbf{s}_i \leftarrow \mathsf{MixColumns}\big(\mathsf{ShiftRows}(\mathbf{s}_i)\big)$

*** Last round ***
10. **for** $i = 0$ **to** d **do** $\mathbf{s}_i \leftarrow \mathbf{s}_i \oplus \mathbf{k}_i^{\mathrm{Nr}}$
11. **for** $l = 1$ **to** 4, $j = 1$ **to** 4 **do**
12. $\big((\mathbf{s}_0)_{l,j}, (\mathbf{s}_1)_{l,j}, \ldots, (\mathbf{s}_d)_{l,j}\big) \leftarrow \mathsf{SecSbox}\big((\mathbf{s}_0)_{l,j}, (\mathbf{s}_1)_{l,j}, \ldots, (\mathbf{s}_d)_{l,j}\big)$
13. **for** $i = 0$ **to** d **do** $\mathbf{s}_i \leftarrow \mathsf{ShiftRows}(\mathbf{s}_i)$
14. **for** $i = 0$ **to** d **do** $\mathbf{s}_i \leftarrow \mathbf{s}_i \oplus \mathbf{k}_i^{\mathrm{Nr}+1}$

*** State unmasking ***
15. $\mathbf{c} \leftarrow \mathbf{s}_0$
16. **for** $i = 1$ **to** d **do** $\mathbf{c} \leftarrow \mathbf{c} \oplus \mathbf{s}_i$

4 Security Analysis

In this section, we give a formal security proof for our scheme. After describing the security model, we pay particular attention to the secure field multiplication algorithm SecMult (*i.e.* the generalized ISW scheme) which is the sensitive part of our scheme. We improve the security proof given in [18] for the ISW scheme and we show that it achieves dth-order security rather than $(d/2)$th-order security. Afterward, we prove the security of the whole AES computation (Algorithm 7).

4.1 Security Model

We consider a *randomized encryption algorithm* $\mathcal{E}$ taking a plaintext p and a (randomly shared) secret key k as inputs[6] and performing a deterministic encryption of p under the secret key k while randomizing its internal computations by means of an external random number generator (RNG). The RNG outputs are assumed to be perfectly random (uniformly distributed, mutually independent and independent of the plaintext and of the secret key). Any variable that can be expressed as a deterministic function of the plaintext and the secret key, which is not constant with respect to the secret key, is called a *sensitive variable* with the exception of the ciphertext $\mathcal{E}_k(p)$ or any deterministic function of it. Note that every intermediate variable computed during an execution of $\mathcal{E}$ (except the plaintext and the ciphertext) can be expressed as a deterministic function of a sensitive variable and of the RNG outputs.

We shall consider the plaintext, the secret key and the intermediate variables of $\mathcal{E}$ as random variables. The distributions of the intermediate variables are induced by the algorithm inputs (p and k) distributions and by the uniformity of the RNG outputs. The joint distribution of all the intermediate variables of $\mathcal{E}$ thus depends on (p, k). On the other hand, some subsets of intermediate variables may be jointly independent of (p, k). This leads us to the following formal definition of *dth-order SCA security*.

Definition 1. *A randomized encryption algorithm is said to achieve* dth-order SCA security *if every d-tuple of its intermediate variables is independent of any sensitive variable.*

Equivalently, an encryption algorithm achieves dth-order SCA security if any d-tuple of its intermediate variables, except the plaintext and the ciphertext (or any function of one of them), is independent of the algorithm inputs (p, k).

Before proving the security of our scheme, we need to introduce a few additional notions. A $(d+1)$*-family of shares* is a family of $d+1$ intermediate variables $(x_i)_{0 \leqslant i \leqslant d}$ such that every d-tuple of x_i's is uniformly distributed and independent of any sensitive variable and $\bigoplus_{0 \leqslant i \leqslant d} x_i$ is a sensitive variable. Two $(d+1)$-families of shares $(x_i)_i$ and $(y_i)_i$ are said to be

[6] The secret key k is assumed to be split into $d+1$ shares k_0, k_1, ..., k_d such that $\bigoplus_i k_i = k$ and every d-tuple of k_i's is uniformly distributed and independent of k.

d-independent one of each other if every $(2d)$-tuple composed of d elements from $(x_i)_i$ and of d elements from $(y_i)_i$ is uniformly distributed and independent of any sensitive variable. Two $(d+1)$-families of shares are said to be *d-dependent* one on each other if they are not d-independent. A randomized encryption algorithm aiming at dth-order SCA security typically operates on $(d+1)$-families of shares. Such an algorithm can hence be split into several *randomized elementary transformations* defined as algorithms taking one or two d-independent $(d+1)$-families of shares as input and returning a $(d+1)$-family of shares.

To prove the dth-order SCA security of our scheme, we will first show that it can be split into several randomized elementary transformations each achieving dth-order SCA security. Afterward, the security of the whole algorithm will be demonstrated.

As in [18], our proofs shall apply similar techniques as zero-knowledge proofs [16]. We shall show that the distribution of every d-tuple of intermediate variables $(v_1, v_2, \ldots, v_d)$ of our randomized AES algorithm can be *perfectly simulated* without knowing p and k. Namely, we show that it is possible to construct a d-tuple of random variables which is identically distributed as $(v_1, v_2, \ldots, v_d)$, independently of any statement about p and k. In some cases, the simulated distribution shall involve some intermediate variables $(w_i)_i$ (different from the v_i's). We shall then say that $(v_1, v_2, \ldots, v_d)$ can be *perfectly simulated from the w_i's*. It follows that if $(v_1, v_2, \ldots, v_d)$ can be perfectly simulated from some intermediate variables w_i's which are jointly independent of p and k, then $(v_1, v_2, \ldots, v_d)$ is also independent of p and k. We are now able to introduce the first lemma of our security proof.

Lemma 1. *A randomized elementary transformation $\mathcal{T}$ achieves dth-order SCA security if and only if the distribution of every d-tuple of its intermediate variables can be perfectly simulated from at most d shares of each of its input $(d+1)$-families.*

Proof. Let us assume that every d-tuple $\mathbf{v} = (v_1, v_2, \ldots, v_d)$ of intermediate variables of $\mathcal{T}$ can be perfectly simulated from at most d shares of each of its input $(d+1)$-families of shares. By definition of a $(d+1)$-family of shares, this amounts to assume that every such $\mathbf{v}$ can be simulated from (at most) $2d$ uniform random variables that are independent of any sensitive variable. It follows that every d-tuple of intermediate variables $\mathbf{v}$ is independent of any sensitive variable, which implies that $\mathcal{T}$ is dth-order SCA secure. Let us now assume that there exits a d-tuple $\mathbf{v}$ of intermediate variables which requires all the $d+1$ shares of one of its input $(d+1)$-families – let say $(x_i)_i$ – to be perfectly simulated. Then, denoting $\bigoplus_i x_i = x$ where x is a sensitive variable, we get that $\mathbf{v}$ depends on x (otherwise d shares x_i would suffice to the simulation) which contradicts the dth-order SCA security of $\mathcal{T}$. □

Lemma 1 shows that proving the security of a randomized elementary transformation $\mathcal{T}$ can be done by exhibiting a method for perfectly simulating the distribution of any d-tuple of intermediate variables of $\mathcal{T}$ from the values of at most d shares of each input $(d+1)$-family of $\mathcal{T}$. We follow this approach in the next section to prove the security of the secure field multiplication algorithm SecMult (Algorithm 1).

4.2 Improved Security Proof for the ISW Scheme

The theorem hereafter states that the generalized ISW scheme (Algorithm 1) achieves dth-order SCA security.

Theorem 1. *Let $(a_i)_{0\leqslant i\leqslant d}$ and $(b_i)_{0\leqslant i\leqslant d}$ be two d-independent $(d+1)$-families of shares in input of Algorithm 1. Then, the distribution of every tuple of d or less intermediate variables in Algorithm 1 is independent of the distribution of values taken by $a = \bigoplus_{0\leqslant i\leqslant d} a_i$ and $b = \bigoplus_{0\leqslant i\leqslant d} b_i$.*

The proof given hereafter follows the outlines of that given by Ishai *et al.* in their paper but it is tighter: we prove that the scheme achieves dth-order SCA security rather than $(d/2)$th-order SCA security as proved in [18]. The core idea of our improvement is to simulate the distribution of any d-tuple of intermediate variables of Algorithm 1 from d shares in $(a_i)_i$ and d shares in $(b_i)_i$ instead of simulating any $(d/2)$-tuple of intermediate variables from d pairs of shares in $(a_i, b_i)_i$.

Proof. Our proof consists in constructing two sets I and J of indices in $[0; d]$ with cardinalities 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 1 can be perfectly simulated from $a_{|I} := (a_i)_{i\in I}$ and $b_{|J} = (b_j)_{j\in J}$. This will prove the theorem statement since, by definition, $a_{|I}$ and $b_{|J}$ are jointly independent of (a, b) as long as the cardinalities of I and J are strictly smaller than d. We describe the constructions of I and J hereafter.

1. Initially, I and J are empty and all the v_h's are unassigned.
2. For every intermediate variable v_h of the form a_i, b_i, a_ib_i, $r_{i,j}$ (for any $i \neq j$) or a sum of values of the above form (including c_i as a special case) add i to I and J. This covers all the intermediate variables of Algorithm 1 except those appearing in the computation of $r_{j,i}$ (Step 4) which are of the form a_ib_j or $r_{i,j} \oplus a_ib_j$. For those intermediate variables add i to I and j to J.
3. Now that the sets I and J have been determined – and note that since there are at most d intermediate variables v_h, the cardinalities of I and J can be at most d – we show how to complete a perfect simulation of the d-tuple $(v_0, v_1, \ldots, v_d)$ using only the values of $a_{|I}$ and $b_{|J}$. First, we assign values to every $r_{i,j}$ entering in the computation of any v_h as follows:
   - If $i \notin I$ (regardless of j), then $r_{i,j}$ does not enter into the computation for any v_h. Thus, its value can be left unassigned.
   - If $i \in I$, but $j \notin I$, then $r_{i,j}$ is assigned a random independent value. Indeed, if $i < j$ this is what would have happened in Algorithm 1. If $i > j$, however, we are making use of the fact that $r_{j,i}$ will never be used in the computation of any v_h (otherwise we would have $j \in I$ by construction). Hence we can treat $r_{i,j}$ as a uniformly random and independent value.
   - If $\{i, j\} \subseteq I$ and $\{i, j\} \subseteq J$, then we have access to a_i, a_j, b_i and b_j and we thus compute $r_{i,j}$ and $r_{j,i}$ exactly as they would have been computed in Algorithm 1; i.e., one of them (say $r_{i,j}$) is assigned a random value and the other $r_{j,i}$ is assigned $r_{i,j} \oplus a_ib_j \oplus a_jb_i$.
   - If $\{i, j\} \subseteq I$ and $\{i, j\} \nsubseteq J$, then at least $r_{i,j}$ or $r_{j,i}$ (or both) does not enter into the computation for any v_h (otherwise we would have $\{i, j\} \subseteq J$ by construction). Following the same reasoning as previously (case $i \in I$, $j \notin I$), we can then assign a random independent value to the one (if any) that enters in the computation of the v_h's.
4. For every intermediate variable v_h of the form a_i, b_i, a_ib_i, $r_{i,j}$ (for any $i \neq j$), or a sum of values of the above form (including c_i as a special case), we know that $i \in I$ and $i \in J$,

and all the needed values of $r_{i,j}$ have already been assigned in a perfect simulation. Thus, v_h can be computed in a perfect simulation.

5. The only types of intermediate variables remaining are $v_h = a_i b_j$ or $v_h = r_{i,j} \oplus a_i b_j$. By construction, we have $i \in I$ and $j \in J$ which allows us to compute $a_i b_j$, and since all the $r_{i,j}$ (entering into the computation of the v_h's) has been assigned, the value of v_h can be simulated perfectly.

□

4.3 Security Proof of Our Scheme

The following theorem states the security of our whole randomized AES (Algorithm 7).

Theorem 2. *The randomized AES computation depicted in Algorithm 7 achieves dth-order SCA security.*

In order to demonstrate the theorem statement, we will use the following lemma.

Lemma 2. *Let $\mathcal{T}$ be a randomized elementary transformation. If $\mathcal{T}$ achieves dth-order SCA security then the distribution of every intermediate variable of $\mathcal{T}$ can be perfectly simulated from at most one share of every input $(d+1)$-families of $\mathcal{T}$.*

Proof. Suppose that the simulation of the distribution of an intermediate variable v from $\mathcal{T}$ requires at least two shares x_{i_1} and x_{i_2} from the same family $(x_i)_i$. The d-tuple composed of v and of the $d-2$ shares $(x_i)_{i \neq i_1, i_2}$ requires the whole $(d+1)$-family of shares $(x_i)_i$ to be perfectly simulated which by Lemma 1 is in contradiction with the dth-order security of $\mathcal{T}$. □

Proof (Theorem 2). An execution of our randomized AES algorithm can be expressed as a sequence of executions of the following randomized elementary transformations[7]:

- the secure key addition (Steps 6, 10 and 14 of Algorithm 7),
- the secure affine transformation (Steps 1 and 2 of Algorithm 5),
- the secure square (Step 1 of Algorithm 3), the secure raising to the 4 (Step 4 of Algorithm 3) and the secure raising to the 16 (Step 7 of Algorithm 3),
- the RefreshMasks procedure (Algorithm 4),
- the SecMult algorithm (Algorithm 1),
- the secure ShiftRows and the secure MixColumns transformations (Steps 9 and 13 of Algorithm 7).

[7] For simplicity we omit the randomized elementary transformations used in the secure key expansion (Algorithm 6). Note that they could be listed without affecting the rest of the proof.

All these transformations take as input either a single $(d+1)$-family of shares (all transformations but the secure key addition and SecMult) or two d-independent $(d+1)$-families of shares (secure key addition and SecMult). Moreover they all achieve dth-order SCA security (it has been proven for SecMult in the previous section and it is straightforward for the remaining randomized elementary transformations since they operate on each input share independently). Let us consider a d-tuple $(v_1, v_2, \ldots, v_d)$ of intermediate variables each from a randomized elementary transformation $\mathcal{T}_i$. By Lemma 2, the distribution of every v_i can be perfectly simulated given the value of at most one share of every $(d+1)$-families in input of $\mathcal{T}_i$. Since by definition the $(d+1)$-families in input of the same $\mathcal{T}_i$ are independent, the set of shares which are necessary to simulate $(v_1, v_2, \ldots, v_d)$ does not contain more than d shares from the same $(d+1)$-family or from d-dependent $(d+1)$-families. It follows that the distribution of $(v_1, v_2, \ldots, v_d)$ can be perfectly simulated from uniform random values that are jointly independent of any sensitive variable. In other words, $(v_1, v_2, \ldots, v_d)$ is independent of any sensitive variable. □

5 Implementation Results

To compare the efficiency of our proposal with that of other methods proposed in the literature, we applied them to protect an implementation of the AES-128 algorithm in encryption mode. We have implemented our new countermeasure for $d \in \{1, 2, 3\}$, namely to counteract either first-order SCA ($d = 1$) or second-order SCA ($d = 2$) or third-order SCA ($d = 3$). Among the numerous methods proposed in the literature to thwart first-order SCA we chose to implement only that having the best timing performance (the *table re-computation method* [23]) and that offering the best memory performance (the *tower field method* [28]). In the second-order case, we implemented the only two existing methods: the one proposed in [38][8] and the one proposed [34]. Eventually, since no countermeasure against 3rd-order SCA was existing before that introduced in this paper, it is the single one in its category.

We wrote the codes in assembly language for an 8051-based 8-bit architecture. The implementations only differ in their approaches to protect the S-box computations. The linear steps of the AES have been implemented in the same way, by following the outlines of the method presented in Sect. 3.2 (and also used in [38] and [34]). In Table 2, we list the timing/memory performances of the different implementations.

As expected, in the first-order case the countermeasures introduced in [23] and [28,29] are much more efficient than ours. This is a consequence of the generic character of our method which is not optimized for one choice of d but aims to work for any d. For instance, the representation of the AES S-box used in [28, 29] involves less field multiplications than our representation. Moreover, those field multiplications can be defined in the subfield $\mathbb{F}_{16}$ of $\mathbb{F}_{256}$, where the field operations can be entirely looked-up thanks to a table of 256 bytes in code memory.

In the second-order case, our proposal becomes much more efficient than the existing solutions. It is 2.2 times faster than the countermeasure proposed in [38] with a RAM memory requirement divided by around 10. It is also 2.5 times faster than the countermeasure in [34] and requires 5.3 times less RAM. Memory allocation differences are merely due to the fact

[8] Initially, the method of [38] was devoted to thwart dth-order SCA for any chosen order d but it has been shown insecure for $d \geqslant 3$ [8].

Table 2. Comparison of secure AES implementations

Method	Reference	cycles	RAM (bytes)	ROM (bytes)
Unprotected Implementation				
No Masking	Na.	3×10^3	32	1150
First Order Masking				
Re-computation	[23]	10×10^3	$256 + 35$	1553
Tower Field in $\mathbb{F}_4$	[28, 29]	77×10^3	42	3195
Our scheme for $d = 1$	This paper	129×10^3	73	3153
Second Order Masking				
Double Re-computations	[38]	594×10^3	$512 + 90$	2336
Single Re-computation	[34]	672×10^3	$256 + 86$	2215
Our scheme for $d = 2$	This paper	271×10^3	79	3845
Third Order Masking				
Our scheme for $d = 3$	This paper	470×10^3	103	4648

that the methods [38] and [34] generalize the table re-computation method and thus require the storage of one (for [34]) or two (for [38]) randomized representation(s) of the AES S-box. The differences in timing performances come from the fact that the methods in [38] and [35] process one loop over all the 256 elements of the S-box look-up table (each loop iteration processing itself a few elementary operations), which is more costly than the 36 field multiplications and 56 bitwise additions involved in our method (see Table 1).

Remark 4. In [34], an improvement of the method implemented for Table 2 is proposed that enables to decrease the number of iterations required for the secure S-box computation when implemented on a 16-bit or 32-bit architecture. In such a context ($d = 2$, 16-bit or 32-bit architecture), the method would still requires much more RAM allocation than ours but it could be slightly more efficient in timing.

Eventually, in the third-order case our method has acceptable timing/memory performances. For comparison, it stays faster than the second-order countermeasures proposed in [38] and [34] and it still requires much less RAM memory. For chips running at 5MHz and 31MHz, an AES encryption of one block requiring 470×10^3 cycles, takes 94ms and 15ms respectively. For some use cases where the size of the message to encrypt/decrypt is not too long such a timing performance is acceptable (*e.g.* challenge-response protocols, Message Authentication Codes for one-block messages as in banking transactions).

6 Conclusion

In this paper, we have presented the first masking scheme dedicated to AES which is provably secure at any chosen order and which can be implemented in software at the cost of a reasonable overhead. We gave a formal security proof of our scheme including an improved security proof for the scheme published by Ishai *et al.* at Crypto 2003. We also provided implementation results showing the practical interest of our scheme as well as its efficiency compared to the existing second-order masking schemes.

References

1. M.-L. Akkar and C. Giraud. An Implementation of DES and AES, Secure against Some Attacks. In **c**C. Ko**c**c, 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. G. Blakely. Safeguarding cryptographic keys. In *National Comp. Conf.*, volume 48, pages 313–317, New York, June 1979. AFIPS Press.
3. J. Blömer, J. G. Merchan, and V. Krummel. Provably Secure Masking of AES. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004.
4. M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM J. Comput.*, 13(4):850–864, 1984.
5. D. Canright. A Very Compact S-Box for AES. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.
6. 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.
7. S. Chari, J. Rao, and P. Rohatgi. Template Attacks. In B. Kaliski Jr., **c**C. Ko**c**c, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–29. Springer, 2002.
8. 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.
9. R. Cramer, I. Damgård, and Y. Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In J. Kilian, editor, *Theory of Cryptography Conference – TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 2005.
10. G. D. Crescenzo, R. J. Lipton, and S. Walfish. Perfectly Secure Password Protocols in the Bounded Retrieval Model. In S. Halevi and T. Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 225–244. Springer, 2006.
11. J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer, 2002.
12. I. Damgård and M. Keller. Secure Multiparty AES (full paper). Cryptology ePrint Archive, Report 20079/614, 2009. `http://eprint.iacr.org/`.
13. S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.
14. FIPS PUB 197. *Advanced Encryption Standard*. National Institute of Standards and Technology, Nov. 2001.
15. FIPS PUB 46-3. *Data Encryption Standard (DES)*. National Institute of Standards and Technology, Oct. 1999.
16. S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1):186–208, 1989.
17. L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In **c**C. Ko**c**c and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
18. Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
19. 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.
20. S. Mangard, T. Popp, and B. M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
21. S. Mangard, N. Pramstaller, and E. Oswald. Successfully Attacking Masked AES Hardware Implementations. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.
22. U. Maurer. A provably-secure strongly-randomized cipher. In I. Damgård, editor, *Advances in Cryptology – EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 361–388. Springer, 1990.
23. 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.
24. T. Messerges. Using Second-order Power Analysis to Attack DPA Resistant Software. In **c**C. Ko**c**c and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.

25. S. Micali and L. Reyzin. Physically Observable Cryptography (Extended Abstract). In M. Naor, editor, *Theory of Cryptography Conference – TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004.
26. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In P. J. Lee and J. H. Cheon, editors, *Information Security and Cryptology – ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008.
27. 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.
28. E. Oswald, S. Mangard, and N. Pramstaller. Secure and Efficient Masking of AES – A Mission Impossible ? Cryptology ePrint Archive, Report 2004/134, 2004.
29. E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen. A Side-Channel Analysis Resistant Description of the AES S-box. In H. Handschuh and H. Gilbert, editors, *Fast Software Encryption – FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2005.
30. C. Petit, F.-X. Standaert, O. Pereira, T. Malkin, and M. Yung. A block cipher based pseudo random number generator secure against side-channel key recovery. In M. Abe and V. D. Gligor, editors, *Symposium on Information, Computer and Communications Security – ASIACCS 2008*, pages 56–65. ACM, 2008.
31. K. Pietrzak. A Leakage-Resilient Mode of Operation. In A. Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 462–482. Springer, 2009.
32. T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard. Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2007.
33. M. Rivain. *On the Physical Security of Cryptographic Implementations*. PhD thesis, University of Luxembourg, September 2009.
34. M. Rivain, E. Dottax, and E. Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In K. Nyberg, editor, *Fast Software Encryption – FSE 2008*, Lecture Notes in Computer Science, pages 127–143. Springer, 2008.
35. M. Rivain, E. Dottax, and E. Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. Cryptology ePrint Archive, Report 2008/021, 2008. `http://eprint.iacr.org/`.
36. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. Cryptology ePrint Archive, 2010. `http://eprint.iacr.org/`.
37. M. Rivain, E. Prouff, and J. Doget. Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.
38. 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.
39. A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.
40. F.-X. Standaert, O. Pereira, Y. Yu, J.-J. Quisquater, M. Yung, and E. Oswald. Leakage resilient cryptography in practice. Cryptology ePrint Archive, Report 2009/341, 2009. `http://eprint.iacr.org/`.
41. F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The World is Not Enough: Another Look on Second-Order DPA. Cryptology ePrint Archive, Report 2010/180, 2010. `http://eprint.iacr.org/`.
42. 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.