Higher-Order Masking Schemes for S-Boxes

Claude Carlet[1], Louis Goubin[2], Emmanuel Prouff[3], Michael Quisquater[2], and Matthieu Rivain[4]

[1] LAGA, Université de Paris 8
claude.carlet@univ-paris8.fr
[2] Université de Versailles St-Quentin-en-Yvelines
louis.goubin@prism.uvsq.fr
michael.quisquater@prism.uvsq.fr
[3] Oberthur Technologies
e.prouff@oberthur.com
[4] CryptoExperts
matthieu.rivain@cryptoexperts.com

Abstract. Masking is a widely used countermeasure against side-channel attacks. The principle is to randomly 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. The main issue while applying masking to protect a block cipher implementation is to design an efficient scheme for the s-box computations. Actually, masking schemes with arbitrary order only exist for Boolean circuits and for the AES s-box. Although any s-box can be represented as a Boolean circuit, applying such a strategy leads to inefficient implementation in software. The design of an efficient and generic higher-order masking scheme was hence until now an open problem. In this paper, we introduce the first masking schemes which can be applied in software to efficiently protect any s-box at any order. We first describe a general masking method and we introduce a new criterion for an s-box that relates to the best efficiency achievable with this method. Then we propose concrete schemes that aim to approach the criterion. Specifically, we give optimal methods for the set of *power functions*, and we give efficient heuristics for the general case. As an illustration we apply the new schemes to the DES and PRESENT s-boxes and we provide implementation results.

1 Introduction

Side-channel analysis is a class of cryptanalytic attacks that exploit the physical environment of a cryptosystem to recover some *leakage* about its secrets. It is often more efficient than a cryptanalysis mounted in the so-called *black-box model* where no leakage occurs. In particular, *continuous side-channel attacks* in which the adversary gets information at each invocation of the cryptosystem are especially threatening. Common attacks as those exploiting the running-time [19], the power consumption [20] or the electromagnetic radiations [12] of a cryptographic computation fall into this class.

Many implementations of block ciphers have been practically broken by continuous side-channel analysis — see for instance [6, 20, 22, 24] — and securing them has been a longstanding issue for the embedded systems industry. A sound approach is to use *secret sharing* [3, 32], often called *masking* in the context of side-channel attacks. This approach consists in splitting each sensitive variable of the implementation (*i.e.* variables depending on the secret key) into $d+1$ shares, where d is called the *masking order*. It has been shown that the complexity of mounting a successful side-channel attack against a masked implementation increases exponentially with the masking order [7]. Starting from this observation, the design of efficient masking schemes for different ciphers has become a foreground issue.

The DES cipher has been the focus of first designs, with the notable work of Goubin and Patarin in [14]. Further schemes have been subsequently published, in particular for the AES cipher, applying masking in hardware or software with different area-time-memory trade-offs [2, 4, 23, 25, 28, 31]. All these schemes deal with *first-order masking*, namely the intermediate

variables are split in two shares (a mask and a masked variable). As a result, they only thwart *first order* side-channel attacks in which the adversary exploits the leakage of a single intermediate computation. During the last years, several works have demonstrated that this defense strategy was not sufficient for long term security purpose and that *higher-order attacks* could be successfully performed against cryptographic implementations (see *e.g.* [24]). This has raised the need for secure and efficient higher-order masking schemes.

Higher-Order Masking. The principle of higher-order masking is to split every sensitive variable x occurring during the computation 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 addition over some field of characteristic 2. 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*. The tuple $(x_i)_i$ is further called a *dth-order encoding of x*.

When higher-order masking is involved to protect a block cipher implementation, a so-called *masking scheme* must be designed to enable the computation on masked data. Such a scheme must ensure that the final shares correspond to the expected ciphertext on the one hand, and it must ensure the dth-order security property for the chosen order d on the other hand. The latter property states that every tuple of d or less intermediate variables is independent of any sensitive variable. When satisfied, it guarantees that no attack of order lower than or equal to d is possible.

Most block cipher structures (*e.g.* AES or DES) are iterative, meaning that they apply several times a same transformation, called *round*, to an internal state initially filled with the plaintext. The round itself is composed of a key addition, one or several linear transformation(s) and one or several non-linear s-box(es). Key addition and linear transformations are easily handled as linearity enables to process each share independently. The main difficulty in designing masking schemes for block ciphers hence lies in masking the s-box(es).

Masking and S-Boxes. Whereas many solutions have been proposed to deal with the case of first-order masking (see *e.g.* [2, 4, 23, 27]), only a few solutions exist for the higher-order case. A scheme has been proposed by Schramm and Paar in [31] which generalizes the (first-order) table recomputation method described in [2,23]. Although the authors apply their method in the particular case of an AES implementation, it is generic and can be applied to protect any s-box. Unfortunately, this scheme has been shown to be vulnerable to a 3rd-order attack whatever the chosen masking order [8]. In other words, it only provides 2nd-order security. Further schemes were proposed by Rivain, Dottax and Prouff in [28] with formal security proofs but still limited to 2nd-order security.

The first scheme achieving dth-order security for an arbitrary chosen d has been designed by Ishai, Sahai and Wagner in [15]. 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 $\mathrm{NOT}(x) = \mathrm{NOT}(x_0) \oplus x_1 \cdots \oplus x_d$. The main contribution of [15] is a method to secure the AND operation for any arbitrary order d (the description of this scheme is recalled in Section 2.1). Although the ISW scheme is an important theoretical result, its practical application faces some issues. At the hardware level, the obtained circuits may have prohibitive area requirements, especially for being used in embedded systems (privileged targets of side-channel attacks). Moreover, Mangard *et al.*

have shown in [21, 22] that masking at the hardware level is sensitive to *glitches* which induce unpredicted flaws in masked circuits. Preventing glitches can be done thanks to synchronization elements (*e.g.* registers or latches) [26] or by performing additional sharing [25] but in both cases, the circuit size is still significantly increased. On the other hand, a direct application of the ISW scheme to secure an s-box computation in software would consist in taking the Boolean representation of the s-box and in processing every logical operation successively in a masked way. Since the Boolean representation of common s-boxes involves a huge number of logical operations, the resulting implementation would likely be inefficient.

In the particular case of AES, a solution has been proposed by Rivain and Prouff in [29] to efficiently mask the s-box processing at any order. Specifically, the authors use the algebraic structure of the AES s-box, which is the composition of an affine function over $\mathbb{F}_2^8$ with the power function $x \mapsto x^{254}$ over $\mathbb{F}_{256}$, and they show that it can be expressed as a sequence of operations involving a few linear functions over $\mathbb{F}_2^8$ (easy to mask) and four multiplications over $\mathbb{F}_{256}$. The latter are secured by applying the ISW scheme (generalized to $\mathbb{F}_{256}$). Subsequently, Kim, Hong and Lim have presented in [16] an extension of Rivain and Prouff's scheme, which is based on the tower-field approach from [30]. On the other hand, Genelle, Prouff and Quisquater have proposed in [13] a higher-order scheme based on the alternate use of Boolean masking and multiplicative masking. Although schemes in [16] and [13] achieve better performances than [29], they are still restricted to the AES s-box and their generalization to any s-box (or subclasses) is an open issue.

Our Contribution. The present paper introduces the first higher-order masking scheme which can be applied to efficiently protect any s-box processing in software. We first give a general method that extends the Rivain and Prouff approach to mask any s-box and we introduce a new criterion for an s-box that relates to the best efficiency achievable with our method. Then we give concrete schemes that aim to approach the so-called *masking complexity*. Specifically, we give optimal methods for the set of *power functions*, and we give efficient heuristics for the general case. As an illustration we apply our scheme to the DES and PRESENT s-boxes and we provide implementation results.

2 Higher-Order Masking of any S-Box

In this section, we describe a general method to mask any s-box and we introduce a related *masking complexity* criterion.

2.1 General Method

An s-box is a function from $\{0,1\}^n$ to $\{0,1\}^m$ with $m \leq n$ and n small (typically $n \in \{4, 6, 8\}$). We shall use the terminology of (n, m) s-box when the dimensions need to be specified. To design a higher-order masking scheme for such a function, our approach is to express it 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 (n, m) s-box can be represented by a polynomial function $x \mapsto \sum_{i=0}^{2^n-1} a_i x^i$ over $\mathbb{F}_{2^n}$ where the a_i are constant coefficients in $\mathbb{F}_{2^n}$. The a_i can be obtained from the s-box look-up table by applying Lagrange's Interpolation Theorem. When m is strictly lower than n, the m-bit outputs can be embedded into $\mathbb{F}_{2^n}$ by padding them to n-bit outputs (*e.g.* by setting most significant bits to 0). The padding is then removed after the polynomial evaluation. We recall hereafter the Lagrange Interpolation Theorem applied to our context.

Theorem 1 (Lagrange Interpolation). *Let* S *be a function* $\mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$*. Then, for every* $x \in \mathbb{F}_{2^n}$*, we have:*

$$\mathrm{S}(x) = \sum_{\alpha \in \mathbb{F}_{2^n}} \mathrm{S}(\alpha)\ell_\alpha(x) \ , \tag{2}$$

where, for every $\alpha \in \mathbb{F}_{2^n}$*,* ℓ_α *is defined as:*

$$\ell_\alpha(x) = \prod_{\substack{\beta \in \mathbb{F}_{2^n} \\ \beta \neq \alpha}} \frac{x - \beta}{\alpha - \beta} \ . \tag{3}$$

Remark 1. The ℓ_α are called the *Lagrange basis polynomials* and satisfy $\ell_\alpha(x) = 1$ if $x = \alpha$ and $\ell_\alpha(x) = 0$ otherwise. In particular, every ℓ_α is a monic polynomial of degree $2^n - 1$, and we have $\ell_\alpha(x) = (x + \alpha)^{2^n - 1} + 1$. Moreover, the coefficients of $\mathrm{S}(x)$ can be directly computed from the Mattson-Solomon polynomial by:

$$a_i = \begin{cases} \mathrm{S}(0) & \text{if } i = 0 \\ \sum_{k=0}^{2^n-2} \mathrm{S}(\alpha^k)\alpha^{-ki} & \text{if } 1 \leq i \leq 2^n - 2 \\ \mathrm{S}(1) + \sum_{i=0}^{2^n-2} a_i & \text{if } i = 2^n - 1 \end{cases}$$

for every primitive element α of $\mathbb{F}_{2^n}$.

The polynomial representation of an s-box is based on four kinds of operations over $\mathbb{F}_{2^n}$: additions, scalar multiplications (*i.e.* multiplications by constants), squares, and regular multiplications (*i.e.* of two different variables). Except for the latter, all these operations are $\mathbb{F}_2^n$*-linear* (or $\mathbb{F}_2^n$*-affine*), that is the corresponding function over $\mathbb{F}_2^n$ are linear (resp. affine). The processing of any s-box can then be performed as a sequence of $\mathbb{F}_2^n$-affine functions (themselves composed of additions, squares and scalar multiplications over $\mathbb{F}_{2^n}$) and of regular multiplications over $\mathbb{F}_{2^n}$, called *nonlinear multiplications* in the following. Masking an s-box processing can hence be done by masking every affine function and every nonlinear multiplication independently. We recall hereafter how this can be done for each category.

Masking of $\mathbb{F}_2^n$*-affine functions.* Let $x = \sum_i x_i$ be a shared variable. Every affine function g with additive part c_g satisfies:

$$g(x) = \begin{cases} \sum_{i=0}^{d} g(x_i) & \text{if } d \text{ is even,} \\ c_g + \sum_{i=0}^{d} g(x_i) & \text{if } d \text{ is odd.} \end{cases}$$

The masked processing of g then simply consists in evaluating g for every share x_i, and possibly correcting one of them by addition of c_g. Such a processing clearly achieves dth-order security as the shares are all processed independently.

Masking of nonlinear multiplications. Every nonlinear multiplication can be processed by using the ISW scheme. Let $a, b \in \mathbb{F}_{2^n}$ and let $(a_i)_{0 \leq i \leq d}$ and $(b_i)_{0 \leq i \leq d}$ be dth-order encoding of a and b. To securely compute a dth-order encoding $(c_i)_{0 \leq i \leq d}$ of $c = ab$, the ISW method over $\mathbb{F}_{2^n}$ performs as follows:[5]

1. For every $0 \leq i < j \leq d$, pick up a random value $r_{i,j}$ in $\mathbb{F}_{2^n}$.
2. For every $0 \leq i < j \leq d$, compute $r_{j,i} = (r_{i,j} + a_i b_j) + a_j b_i$.
3. For every $0 \leq i \leq d$, compute $c_i = a_i b_i + \sum_{j \neq i} r_{i,j}$.

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

It can be checked that the obtained shares are a sound encoding of c. Namely, we have:

$$\sum_{i=0}^{d} c_i = \Big(\sum_{i=0}^{d} a_i\Big)\Big(\sum_{i=0}^{d} b_i\Big) = ab = c.$$

In [15] it is shown that the above computation achieves $(d/2)$th-order security. A tighter security proof is given in [29] which shows that dth-order security is actually achieved as long as the masks of the two inputs are independent. Therefore, we shall refresh the masks before a masked multiplication when necessary. This can be done using a refreshing procedure as proposed in [29] (see Algorithm 2 in appendix).

Remark 2. Another method to process a masked multiplication at an arbitrary order is used in [10] to achieve provable security under specific leakage assumptions. However this method requires more operations and more random bits than the ISW scheme does. For this reason, the ISW scheme should be preferred in a usual dth-order security model.

2.2 Masking Complexity

The scheme described in the previous section secures the computation of any (n, m) s-box S by masking its polynomial representation over $\mathbb{F}_{2^n}$. The evaluation of such a polynomial is composed of $\mathbb{F}_2^n$-affine functions g and of nonlinear multiplications. The masked processing of each $\mathbb{F}_2^n$-affine function g merely involves $d+1$ evaluations of g itself, while it involves $(d+1)^2$ field multiplications, $2d(d+1)$ field additions and the generation of $nd(d+1)/2$ random bits for each nonlinear multiplication. The masked processing of $\mathbb{F}_2^n$-affine functions hence quickly becomes negligible compared to the masked processing of nonlinear multiplications as d grows. This observation motivates the following definition of the *masking complexity* for an s-box.

Definition 1 (Masking Complexity). *Let m and n be two integers such that $m \leq n$. The* masking complexity *of a (n, m) s-box is the minimal number of nonlinear multiplications required to evaluate its polynomial representation over $\mathbb{F}_{2^n}$.*

The following proposition directly results from this definition.

Proposition 1. *The masking complexity of an s-box is invariant when composed with $\mathbb{F}_2^n$-affine bijections in input and/or in output.*

Remark 3. Since field isomorphisms are $\mathbb{F}_2$-linear bijections, the choice of the irreducible polynomial to represent field elements does not impact the masking complexity of an s-box.

In the next sections, we address the issue of finding polynomial evaluations of an s-box that aim at minimizing the number of nonlinear multiplications. Those constructions will enable us to deduce upper bounds on the masking complexity of an s-box. We first study the case of power functions whose polynomial representation has a single monomial (*e.g.* the AES s-box). For these functions, we exhibit the exact masking complexity by deriving addition chains with minimal number of nonlinear multiplications. We then address the general case and provide efficient heuristics to evaluate any s-box with a low number of nonlinear multiplications.

3 Optimal Masking of Power Functions

In this section, we consider s-boxes for which the polynomial representation over $\mathbb{F}_{2^n}$ is a single monomial. These s-boxes are usually called *power functions* in the literature. We describe a generic method to compute the masking complexity of such s-boxes. Our method involves the notion of *cyclotomic class*.

Definition 2. *Let $\alpha \in [0; 2^n - 2]$. The* cyclotomic class *of α is the set C_α defined by:*

$$C_\alpha = \{\alpha \cdot 2^i \bmod 2^n - 1;\ i \in [0; n-1]\}.$$

We have the following proposition.

Proposition 2. *Let $\mu(m)$ denote the multiplicative order of 2 modulo m and let φ denote the Euler's totient function. For every divisor δ of $2^n - 1$, the number of distinct cyclotomic classes $C_\alpha \subseteq [0; 2^n - 2]$ with $\gcd(\alpha, 2^n - 1) = \delta$ is $\varphi\left(\frac{2^n-1}{\delta}\right)/\mu\left(\frac{2^n-1}{\delta}\right)$. It follows that the total number of distinct cyclotomic classes of $[0; 2^n - 2]$ equals:*

$$\sum_{\delta|(2^n-1)} \frac{\varphi(\delta)}{\mu(\delta)}\ .$$

Proof. Proposition 2 can be deduced from the following facts:

- An integer $\alpha \in [0; 2^n-2]$ satisfies $\gcd(\alpha, 2^n-1) = \delta$ if and only if $\alpha = \delta\beta$, with $\gcd(\beta, \frac{2^n-1}{\delta}) = 1$. There are thus $\varphi\left(\frac{2^n-1}{\delta}\right)$ integers $\alpha \in [0; 2^n - 2]$ such that $\gcd(\alpha, 2^n - 1) = \delta$.
- For any α such that $\gcd(\alpha, 2^n - 1) = \delta$ (hence of the form $\alpha = \delta\beta$ with $\gcd(\beta, \frac{2^n-1}{\delta}) = 1$), we have $\alpha \cdot 2^i \equiv \alpha \cdot 2^j \bmod 2^n - 1$ if and only if $\beta \cdot 2^i \equiv \beta \cdot 2^j \bmod \frac{2^n-1}{\delta}$, that is, if and only if $2^i \equiv 2^j \bmod \frac{2^n-1}{\delta}$. Hence C_α has cardinality $\#C_\alpha = \mu\left(\frac{2^n-1}{\delta}\right)$.

The set of integers $\alpha \in [0; 2^n - 2]$ such that $\gcd(\alpha, 2^n - 1) = \delta$ is partitioned into cyclotomic classes, each of them having cardinality $\mu\left(\frac{2^n-1}{\delta}\right)$. Hence the number of such cyclotomic classes is $\varphi\left(\frac{2^n-1}{\delta}\right)/\mu\left(\frac{2^n-1}{\delta}\right)$. It follows that the total number of distinct cyclotomic classes of $[0; 2^n - 2]$ equals $\sum_{\delta|(2^n-1)} \varphi\left(\frac{2^n-1}{\delta}\right)/\mu\left(\frac{2^n-1}{\delta}\right) = \sum_{\delta|(2^n-1)} \varphi(\delta)/\mu(\delta)$.

□

The study of cyclotomic classes is interesting in our context since a power x^α can be computed from a power x^β without any nonlinear multiplication if and only if α and β lie in the same cyclotomic class. Hence, all the power functions with exponents within a given cyclotomic class have the same masking complexity and computing the masking complexity for all the power functions over $\mathbb{F}_{2^n}$ thus amounts to compute this complexity for each cyclotomic class over $\mathbb{F}_{2^n}$. In what follows, we perform such a computation for fields $\mathbb{F}_{2^n}$ of small dimensions n.

To compute the masking complexity for an element in a cyclotomic class, we use the following observation: determining the masking complexity of a power function $x \mapsto x^\alpha$ amounts to find the addition chain for α with the least number of additions which are not doublings (see [17] for an introduction to addition chains). This kind of addition chain is usually called a 2-*addition chain*.[6] Let $(\alpha_i)_i$ denote some addition chain. At step i, it is possible to obtain any element within the cyclotomic classes $(C_{\alpha_j})_{j \le i}$ using doublings only. As we are interested in finding the addition chain with the least number of additions which are not doublings, the problem we need to solve is the following: given some $\alpha \in C_\alpha$, find the shortest chain $C_{\alpha_0} \to C_{\alpha_1} \to \cdots \to C_{\alpha_k}$ where $C_{\alpha_0} = C_1$, $C_{\alpha_k} = C_\alpha$ and for every $i \in [1; k]$, there exists $j, \ell < i$ such that $\alpha_i = \alpha'_j + \alpha'_\ell$ where $\alpha'_j \in C_{\alpha_j}$ and $\alpha'_\ell \in C_{\alpha_\ell}$.

We shall denote by $\mathcal{M}_k^n$ the class of exponents α such that $x \mapsto x^\alpha$ has a masking complexity equal to k. The family of classes $(\mathcal{M}_k^n)_k$ is a partition of $[0; 2^n - 2]$ and each $\mathcal{M}_k^n$ is the union of one or several cyclotomic classes. For a small dimension n, we can proceed by exhaustive search

[6] This problem has been studied in the general setting where the multiplication by q (and not specifically by 2) is considered *free* and the obtained addition chains are called *q-addition chains* [33]. The purpose is to find efficient exponentiation methods in $\mathbb{F}_q$ (as in such field the Frobenius map $x \mapsto x^q$ is efficient). To the best of our knowledge, apart from a specific application to the SFLASH signature algorithm in [1], the case of 2-addition chains has not been particularly investigated.

to determine the shortest 2-addition chain(s) for each cyclotomic class. We implemented such an exhaustive search from which we obtained the masking complexity classes $\mathcal{M}_k^n$ for $n \leq 11$ (note that in practice most s-boxes have dimension $n \leq 8$). Table 1 summarizes the obtained results for $n \in \{4, 6, 8\}$ (usual dimensions). Results for other dimensions are summarized in Appendix A. Additionally, Table 2 gives the optimal 2-addition chains (in exponential notation) corresponding to every cyclotomic class for $n = 8$.

Table 1. Cyclotomic classes for $n \in \{4, 6, 8\}$ w.r.t. the masking complexity k.

k	Cyclotomic classes in $\mathcal{M}_k^n$
	$n = 4$
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8\}$
1	$C_3 = \{3, 6, 12, 9\}$, $C_5 = \{5, 10\}$
2	$C_7 = \{7, 14, 13, 11\}$
	$n = 6$
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8, 16, 32\}$
1	$C_3 = \{3, 6, 12, 24, 48, 33\}$, $C_5 = \{5, 10, 20, 40, 17, 34\}$, $C_9 = \{9, 18, 36\}$
2	$C_7 = \{7, 14, 28, 56, 49, 35\}$, $C_{11} = \{11, 22, 44, 25, 50, 37\}$, $C_{13} = \{13, 26, 52, 41, 19, 38\}$, $C_{15} = \{15, 30, 29, 27, 23\}$, $C_{21} = \{21, 42\}$, $C_{27} = \{27, 54, 45\}$
3	$C_{23} = \{23, 46, 29, 58, 53, 43\}$, $C_{31} = \{31, 62, 61, 59, 55, 47\}$
	$n = 8$
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8, 16, 32, 64, 128\}$
1	$C_3 = \{3, 6, 12, 24, 48, 96, 192, 129\}$, $C_5 = \{5, 10, 20, 40, 80, 160, 65, 130\}$, $C_9 = \{9, 18, 36, 72, 144, 33, 66, 132\}$, $C_{17} = \{17, 34, 68, 136\}$
2	$C_7 = \{7, 14, 28, 56, 112, 224, 193, 131\}$, $C_{11} = \{11, 22, 44, 88, 176, 97, 194, 133\}$, $C_{13} = \{13, 26, 52, 104, 208, 161, 67, 134\}$, $C_{15} = \{15, 30, 60, 120, 240, 225, 195, 135\}$, $C_{19} = \{19, 38, 76, 152, 49, 98, 196, 137\}$, $C_{21} = \{21, 42, 84, 168, 81, 162, 69, 138\}$, $C_{25} = \{25, 50, 100, 200, 145, 35, 70, 140\}$, $C_{27} = \{27, 54, 108, 216, 177, 99, 198, 141\}$, $C_{37} = \{37, 74, 148, 41, 82, 164, 73, 146\}$, $C_{45} = \{45, 90, 180, 105, 210, 165, 75, 150\}$, $C_{51} = \{51, 102, 204, 153\}$, $C_{85} = \{85, 170\}$
3	$C_{23} = \{23, 46, 92, 184, 113, 226, 197, 139\}$, $C_{29} = \{29, 58, 116, 232, 209, 163, 71, 142\}$, $C_{31} = \{31, 62, 124, 248, 241, 227, 199, 143\}$, $C_{39} = \{39, 78, 156, 57, 114, 228, 201, 147\}$, $C_{43} = \{43, 86, 172, 89, 178, 101, 202, 149\}$, $C_{47} = \{47, 94, 188, 121, 242, 229, 203, 151\}$, $C_{53} = \{53, 106, 212, 169, 83, 166, 77, 154\}$, $C_{55} = \{55, 110, 220, 185, 115, 230, 205, 155\}$, $C_{59} = \{59, 118, 236, 217, 179, 103, 206, 157\}$, $C_{61} = \{61, 122, 244, 233, 211, 167, 79, 158\}$, $C_{63} = \{63, 126, 252, 249, 243, 231, 207, 159\}$, $C_{87} = \{87, 174, 93, 186, 117, 234, 213, 171\}$, $C_{91} = \{91, 182, 109, 218, 181, 107, 214, 173\}$, $C_{95} = \{95, 190, 125, 250, 245, 235, 215, 175\}$, $C_{111} = \{111, 222, 189, 123, 246, 237, 219, 183\}$, $C_{119} = \{119, 238, 221, 187\}$
4	$C_{127} = \{127, 254, 253, 251, 247, 239, 223, 191\}$

It is interesting to note that for every n, the *inverse function* $x \mapsto x^{2^n-2}$ related to the cyclotomic class $C_{2^{n-1}-1}$ always has the highest masking complexity. In particular, the inverse function $x \mapsto x^{254}$ (for $n = 8$) used in the AES has a masking complexity of 4 as it was conjectured in [29].

4 Efficient Heuristics for General S-Boxes

We now address the general case of an s-box having a polynomial representation $\sum_{j=0}^{2^n-1} a_j x^j$ over $\mathbb{F}_{2^n}$. A straightforward solution is to successively compute every power x^j using $x^j = (x^{j/2})^2$ if j is even and $x^j = x^{j-1}x$ if j is odd, while updating the polynomial value by adding the monomial $a_j x^j$ at every step. Such a method requires $2^{n-1} - 1$ nonlinear multiplications. As

Table 2. Optimal 2-addition chains (in exponential notation) for cyclotomic classes for $n = 8$.

k	2-addition chains with k nonlinear multiplications
1	$x^3 \leftarrow x \times x^2$ – $x^5 \leftarrow x \times x^4$ $x^9 \leftarrow x \times x^8$ – $x^{17} \leftarrow x \times x^{16}$
2	$x^7 \leftarrow x \times x^2 \times x^4$ – $x^{11} \leftarrow x \times x^2 \times x^8$ $x^{13} \leftarrow x \times x^4 \times x^8$ – $x^{15} \leftarrow x^3 \times (x^3)^4$ $x^{19} \leftarrow x \times x^2 \times x^{16}$ – $x^{21} \leftarrow x \times x^4 \times x^{16}$ $x^{27} \leftarrow x^3 \times (x^3)^8$ – $x^{37} \leftarrow x \times x^4 \times x^{32}$ $x^{45} \leftarrow x^5 \times (x^5)^8$ – $x^{51} \leftarrow x^3 \times (x^3)^{16}$ $x^{85} \leftarrow x^5 \times (x^5)^{16}$
3	$x^{23} \leftarrow x \times x^2 \times x^4 \times x^{16}$ – $x^{29} \leftarrow x \times x^4 \times x^8 \times x^{16}$ $x^{31} \leftarrow x^3 \times (x^3)^4 \times x^{16}$ – $x^{29} \leftarrow x \times x^2 \times x^4 \times x^{32}$ $x^{43} \leftarrow x \times x^2 \times x^8 \times x^{32}$ – $x^{47} \leftarrow x^3 \times (x^3)^4 \times x^{32}$ $x^{53} \leftarrow x \times x^2 \times x^{16} \times x^{32}$ – $x^{55} \leftarrow x^3 \times x^4 \times (x^3)^{16}$ $x^{59} \leftarrow x^3 \times (x^3)^8 \times x^{32}$ – $x^{59} \leftarrow x^5 \times x^{16} \times (x^5)^8$ $x^{63} \leftarrow x^7 \times (x^7)^8$ – $x^{87} \leftarrow x^2 \times x^5 \times (x^5)^{16}$ $x^{91} \leftarrow x^3 \times (x^3)^8 \times x^{64}$ – $x^{95} \leftarrow x^5 \times (x^5)^2 \times (x^5)^{16}$ $x^{111} \leftarrow x^3 \times (x^3)^4 \times (x^3)^{32}$ – $x^{63} \leftarrow x^7 \times (x^7)^{16}$
4	$x^{127} \leftarrow x^3 \times (x^3)^4 \times (x^3)^{16} \times x^{64}$

we show hereafter, less naive methods exist that substantially lower the number of nonlinear multiplications. We propose two different methods and then compare their efficiency.

4.1 Cyclotomic Method

Let q denote the number of distinct cyclotomic classes of $[0; 2^n - 2]$. The polynomial representation of S can be written as:

$$\mathrm{S}(x) = a_0 + \Big(\sum_{i=1}^{q} Q_i(x) \Big) + a_{2^n-1}\, x^{2^n-1} \ ,$$

where the Q_i are polynomials such that every Q_i has powers from a single cyclotomic class C_{α_i}, namely we can write $Q_i(x) = \sum_j a_{i,j} x^{\alpha_i 2^j}$ for some coefficients $a_{i,j}$ in $\mathbb{F}_{2^n}$. Let us then denote L_i the linearized polynomial $L_i(x) = \sum_j a_{i,j} x^{2^j}$ which is a $\mathbb{F}_2^n$-linear function of x. We have $Q_i(x) = L_i(x^{\alpha_i})$ by definition. The *cyclotomic method* simply consists in deriving the powers x^{α_i} for each cyclotomic class C_{α_i} as well as x^{2^n-1} if $a_{2^n-1} \neq 0$, and in evaluating $\mathrm{S}(x) = a_0 + \Big(\sum_{i=1}^{q} L_i(x^{\alpha_i}) \Big) + a_{2^n-1}\, x^{2^n-1}$. The powers x^{α_i} can each be derived with a single nonlinear multiplication. This is obvious for the α_i lying in $\mathcal{M}_1^n$. Then it is clear that every power x^{α_i} with $\alpha_i \in \mathcal{M}_{k+1}^n$ can be derived with a single multiplication from the powers $(x^{\alpha_i})_{\alpha_i \in \mathcal{M}_k^n}$. The power x^{2^n-1} can then be derived with a single nonlinear multiplication from the power x^{2^n-2}. The cyclotomic method hence involves a number of nonlinear multiplications equal to the number of cyclotomic classes, minus 2 (as x^0 and x^1 are obtained without nonlinear multiplication), plus 1 (to derive x^{2^n-1}). By Proposition 2, we then have the following result.

Proposition 3 (Cyclotomic Method). *Let m and n be two positive integers such that $m \leq n$. The masking complexity of every (n, m) s-box is upper-bounded by:*

$$\sum_{\delta | (2^n-1)} \frac{\varphi(\delta)}{\mu(\delta)} - 1 \ .$$

An (n, m) s-box S is said to be *balanced* if for every $y \in \{0,1\}^m$, the number of preimages of y for S is constant to 2^{n-m}. The following lemma gives a well-known folklore result.

Lemma 1. *Let m and n be two positive integers such that $m \leq n$. The polynomial representation of every balanced (n, m) s-box has degree strictly lower than $2^n - 1$.*

Proof. Since Lagrange basis polynomials are all monic of degree $2^n - 1$, the coefficient a of the power to the $2^n - 1$ in the polynomial representation of S satisfies $a = \sum_{\alpha \in \mathbb{F}_{2^n}} \mathrm{S}(\alpha)$, which equals 0 if S is balanced. □

When the polynomial representation of the s-box has degree strictly lower than $2^n - 1$, the cyclotomic method saves one nonlinear multiplication since the power x^{2^n-1} is not required. Namely, we have the following corollary of Proposition 3.

Corollary 1 (Cyclotomic Method). *Let m and n be two positive integers such that $m \leq n$ and let* S *be a (n, m) s-box. If* S *is balanced, then the masking complexity of* S *is upper-bounded by:*

$$\sum_{\delta | (2^n-1)} \frac{\varphi(\delta)}{\mu(\delta)} - 2 \ .$$

4.2 Parity-Split Method

The *parity-split method* is composed of two stages. The first stage derives a set of powers $(x^j)_{j \leq q}$ for some q using the straightforward method described in the introduction of this section. The second stage essentially consists in an application of the Knuth-Eve polynomial evaluation algorithm [9, 18] which is based on a recursive use of the following lemma.

Lemma 2. *Let n and t be two positive integers and let Q be a polynomial of degree t over $\mathbb{F}_{2^n}[x]$. There exist two polynomials Q_1 and Q_2 of degree upper-bounded by $\lfloor t/2 \rfloor$ over $\mathbb{F}_{2^n}[x]$ such that:*

$$Q(x) = Q_1(x^2) + Q_2(x^2)x \ . \tag{4}$$

By applying Lemma 2 to the polynomial representation of S, we get $\mathrm{S}(x) = Q_1(x^2) + Q_2(x^2)x$, where Q_1 and Q_2 are two polynomials of degrees upper-bounded by $2^{n-1} - 1$. We deduce that S can be computed based on the set of powers $(x^{2j})_{j \leq 2^{n-1}-1}$ plus a single multiplication by x. Then, applying Lemma 2 again to the polynomials Q_1 and Q_2 both of degrees upper bounded by $2^{n-1} - 1$, we get two new pairs of polynomials (Q_{11}, Q_{12}) and (Q_{21}, Q_{22}) such that $Q_1(x^2) = Q_{11}(x^4) + Q_{12}(x^4)x^2$ and $Q_2(x^2) = Q_{21}(x^4) + Q_{22}(x^4)x^2$. The degrees of the new polynomials are upper bounded by $2^{n-2} - 1$. We then deduce that S can be computed based on the set of powers $(x^{4j})_{j \leq 2^{n-2}-1}$ plus 1 multiplication by x and 2 multiplications by x^2. Eventually, by applying Lemma 2 recursively r times, we get an evaluation of S involving evaluations in x^{2^r} of polynomials of degrees upper-bounded by $2^{n-r} - 1$, plus $\sum_{i=0}^{r-1} 2^i = 2^r - 1$ multiplications by powers of x of the form x^{2^i} with $i \leq r - 1$. The overall evaluation of S hence requires $2^r - 1$ nonlinear multiplications (the x^{2^i} being obtained with squares only) plus the evaluation in x^{2^r} of polynomials of degrees upper-bounded by $2^{n-r} - 1$. The latter evaluation can be performed by first deriving all the powers $(x^{2^r j})_{j \leq 2^{n-r}-1}$ and then evaluating the polynomials (which only involves scalar multiplications and additions once the powers have been derived). For every $j \leq 2^{n-r} - 1$, the powers $(x^{2^r j})_{j \leq 2^{n-r}-1}$ can be computed successively from $y = x^{2^r}$ by $y^j = (y^{j/2})^2$ if j is even and $y^j = y^{j-1}x$ if j is odd. This takes some squares plus $2^{n-r-1} - 1$ nonlinear multiplications (*i.e.* one per odd integer in $[3, 2^{n-r} - 1]$).

We then deduce the following proposition.

Proposition 4. *Let m and n be two positive integers such that $m \leq n$. The masking complexity of every (n, m) s-box is upper-bounded by:*

$$\min_{0 \leq r \leq n} (2^{n-r-1} + 2^r) - 2 = \begin{cases} 3 \cdot 2^{(n/2)-1} - 2 & \text{if } n \text{ is even,} \\ 2^{(n+1)/2} - 2 & \text{if } n \text{ is odd.} \end{cases} \tag{5}$$

Note that the value of r for which the minimum is reached in (5) is $r = \lfloor \frac{n}{2} \rfloor$.

4.3 Comparison

Table 3 summarizes the number of nonlinear multiplications obtained by the cyclotomic method (for balanced s-boxes) and by the parity-split method. We see that the cyclotomic method works better for small dimensions ($n \leq 5$) and the parity-split method for higher dimensions ($n \geq 6$). Furthermore, the superiority of the parity-split method becomes significant as n grows.

Table 3. Number of nonlinear multiplications w.r.t. the evaluation method.

Method \ n	3	4	5	6	7	8	9	10	11
Cyclotomic	1	3	5	11	17	33	53	105	192
Parity-Split	2	4	6	10	14	22	30	46	62

We emphasize that these bounds may not be optimal, namely they may be higher than the maximum masking complexity of (n, m) s-boxes. We let open the issue of finding more efficient (or provably optimal) methods in the general case for further research.

5 Application to DES and PRESENT

In this section we apply the proposed methods to the s-boxes of two different block ciphers: the well-known and still widely used Data Encryption Standard (DES) [11], and the lightweight block cipher PRESENT [5]. The former uses eight different $(6, 4)$ s-boxes and the latter uses a single $(4, 4)$ s-box. According to Table 3, we shall prefer the parity-split method for the DES s-boxes (10 nonlinear multiplications), and the cyclotomic method for the PRESENT s-box (3 nonlinear multiplications).

5.1 Parity-Split Method on DES S-boxes

The parity-split method on a DES s-box uses a polynomial representation of the s-box over $\mathbb{F}_{64}$ which satisfies:

$$\begin{aligned} S \ : \ x \longmapsto \ & Q_0(x^8) + Q_1(x^8) \cdot x^4 + \left(Q_2(x^8) + Q_3(x^8) \cdot x^4\right) \cdot x^2 \\ & + \left(Q_4(x^8) + Q_5(x^8) \cdot x^4 + \left(Q_6(x^8) + Q_7(x^8) \cdot x^4\right) \cdot x^2\right) \cdot x \end{aligned} \tag{6}$$

where the Q_i are degree-7 polynomials, namely, there exist coefficients $a_{i,j}$ for $0 \leq i, j \leq 7$ such that:

$$Q_i(x^8) = a_{i,0} + a_{i,1}x^8 + a_{i,2}x^{16} + a_{i,3}x^{24} + a_{i,4}x^{32} + a_{i,5}x^{40} + a_{i,6}x^{48} + a_{i,7}x^{56} \ .$$

We first derive the powers x^{8j} for $j = 1, 2, \ldots, 7$, which is done at the cost of 3 nonlinear multiplications by:

$$\begin{gathered} x^8 \leftarrow ((x^2)^2)^2; \ x^{16} \leftarrow (x^8)^2; \ x^{24} \leftarrow x^8 \cdot x^{16}; \ x^{32} \leftarrow (x^{16})^2; \\ x^{40} \leftarrow x^8 \cdot x^{32}; \ x^{48} \leftarrow (x^{24})^2; \ x^{56} \leftarrow x^8 \cdot x^{48}; \end{gathered}$$

Then we evaluate each polynomial $Q_i(x^8)$ as a linear combination of the above powers. Finally, we evaluate (6) at the cost of 7 nonlinear multiplications and a few additions. The nonlinear multiplications are computed using the ISW scheme over $\mathbb{F}_{64}$ such as recalled in Section 2.1. A detailed algorithm for the overall masked s-box evaluation is given in Appendix B. Moreover the log/alog tables for the multiplication over $\mathbb{F}_{64}$ and for the $a_{i,j}$ coefficients for the first DES s-box are given in Appendix C.

5.2 Cyclotomic Method on PRESENT S-box

The cyclotomic method on the PRESENT s-box starts from the straightforward polynomial representation of the s-box over $\mathbb{F}_{16}$:

$$\mathrm{S} \ : \ x \ \longmapsto \ a_0 + a_1 x + a_2 x^2 + \cdots + a_{14} x^{14} \ ,$$

(where the degree is indeed strictly lower than 15 by Lemma 1). We then have:

$$\mathrm{S}(x) = a_0 + L_1(x) + L_3(x^3) + L_5(x^5) + L_7(x^7) \ . \tag{7}$$

where:

$$\begin{aligned} L_1 : x \ &\mapsto \ a_1 x + a_2 x^2 + a_4 x^4 + a_8 x^8 \\ L_3 : x \ &\mapsto \ a_3 x + a_6 x^2 + a_{12} x^4 + a_9 x^8 \\ L_5 : x \ &\mapsto \ a_5 x + a_{10} x^2 \\ L_7 : x \ &\mapsto \ a_7 x + a_{14} x^2 + a_{13} x^4 + a_{11} x^8 \end{aligned}$$

and the L_i are $\mathbb{F}_2^4$-linear.

We first derive the powers x^3, x^5, and x^7, which is done at the cost of 3 nonlinear multiplications by: $x^3 \leftarrow x \cdot x^2$; $x^5 \leftarrow x^3 \cdot x^2$; $x^7 \leftarrow x^5 \cdot x^2$. Then we evaluate (7) which costs a few linear transformations and additions. A detailed algorithm for the overall masked s-box evaluation is given in Appendix B. Moreover the look-up tables for the multiplication over $\mathbb{F}_{16}$ and for the L_i transformations are given in Appendix C.

5.3 Implementation Results

In this section, we give implementation results for our scheme applied to DES and PRESENT s-boxes. For comparison, we also give performances of some higher-order masking schemes for the AES s-box, as well as performances of existing schemes for DES and PRESENT s-boxes at orders 1 and 2. For the AES s-box processing, we implemented Rivain and Prouff's method [29] and its improvement by Kim *et al.* [16]. We did not implement Genelle *et al.* 's scheme [13] since it addresses the masking of an overall AES and is not interesting while focusing on a single s-box processing. Regarding existing schemes for DES and PRESENT s-boxes, we implemented the generic methods proposed in [27] (for $d = 1$) and in [28] (for $d = 2$). We also implemented the improvement of these schemes described in [28, §3.3] that consists in treating two 4-bit outputs at the same time.[7] Note that we did not implement the table re-computation method (for $d = 1$) since it only makes sense for an overall cipher and not for a single s-box processing.

Table 4 lists the timing/memory performances of the different implementations. We wrote the codes in assembly language for an `8051` based 8-bit architecture with bit-addressable memory. `ROM` consumptions (*i.e.* code sizes) are not listed since they are not prohibitive.

[7] This improvement is only described in [28] for $d = 2$ but it can be applied likewise to the 1st-order scheme of [27].

Table 4. Comparison of secure s-box implementations

	Method	Reference	cycles	RAM (bytes)
First Order Masking				
1.	AES s-box	[29]	533	10
2.	AES s-box	[16]	320	14
3.	DES s-box	Simple version [27]	1096	2
4.	DES s-box	Improved version [27] & [28]	439	14
5.	DES s-box	this paper	4100	50
6.	PRESENT s-box	Simple Version [27]	281	2
7.	PRESENT s-box	Improved Version [27] & [28]	231	14
4.	PRESENT s-box	this paper	220	18
Second Order Masking				
1.	AES s-box	[29]	832	18
2.	AES s-box	[16]	594	24
3.	DES s-box	Simple version [28]	1045	69
4.	DES s-box	Improved version [28]	652	39
5.	DES s-box	this paper	7000	78
6.	PRESENT s-box	Simple Version [28]	277	21
7.	PRESENT s-box	Improved Version [28]	284	15
8.	PRESENT s-box	this paper	400	31
Third Order Masking				
1.	AES s-box	[29]	1905	28
2.	AES s-box	[16]	965	38
3.	DES s-box	this paper	10500	108
4.	PRESENT s-box	this paper	630	44

As expected, the cyclotomic method is very efficient when applied to protect the PRESENT s-box. The small input dimension of the s-box indeed implies a low masking complexity (equal to 3). Moreover, it enables to tabulate the multiplication over $\mathbb{F}_{16}$. At first order, it is even slightly better than the method in [27] (or its improvement). At second order, the cost of the secure multiplications involved in the cyclotomic method is approximatively doubled, which explains that the overall cost is multiplied by 1.8. This makes it less efficient than [27] and [28], which are less impacted by the increase of the masking order from 1 to 2. At third order, our method is the only one. The number of cycles staying small (630), Table 4 shows that achieving resistance against 3rd-order side-channel analysis is realistic for an implementation of PRESENT on a 8051 architecture. For DES s-boxes, the parity-split method is less efficient than the state-of-the art methods for $d = 1, 2$. This is an expected consequence of the high number of nonlinear multiplications (here 10) achieved with the parity-split method in dimension 6 and of the fact that the field multiplications can no longer be tabulated (and must therefore be computed thanks to log/alog look-up tables). At third order, the timing efficiency of the method becomes very low. The masked s-box processing is 5 (resp. 10) times slower than the efficiency of the AES s-box protected thanks to [16] (resp. [29]), though its input dimension is smaller.

The ranking of the timing efficiencies for AES, DES and PRESENT s-boxes is correlated to the number of nonlinear multiplications in the used scheme (3, 4-5, and 10, for PRESENT, AES and DES respectively) which underline the soundness of the masking complexity criterion. Therefore, while selecting an s-box for a block cipher design, one should favor an s-box with small masking complexity if side-channel attacks are taken into account.

6 Discussion

In previous sections we have introduced the first schemes that can be used to mask any s-box at any order with fair performances in software. In particular, these schemes enable to apply higher-order masking on random s-boxes (*e.g.* the DES s-boxes) which have no specific mathematical structure. Prior to our work, the only existing methods were the circuit-oriented proposals of Ishai *et al.* [15] and of Faust *et al.* [10]. The main purpose of these works was a proof of concept for applying higher-order masking to circuits with formal security proofs, but they did not address efficient implementation. A direct application of [15] or [10] to a block cipher consists in taking its Boolean representation and in replacing every XOR and AND with $O(d)$ and $O(d^2)$ logical operations respectively (where d is the masking order). Applying such a strategy in software leads to inefficient implementation as the Boolean representation of an s-box includes a huge number of nonlinear gates (with a $O(d^2)$ factor to be protected). Compared to these techniques, our schemes achieve significant improvements. These are obtained by starting from the field representation of the s-box and applying methods to significantly reduce the number of nonlinear multiplications compared to the Boolean representation of the s-box. For instance, we have shown that a DES s-box can be computed with 10 nonlinear multiplications whereas its Boolean representation involves several dozens of logical AND operations.

We believe that our work opens up new avenues for research in block cipher implementations and side-channel security. In particular, the issue of designing s-boxes with low masking complexity and good cryptographic criteria is still to be investigated. On the other hand, our work could be extended to take into account more general definitions of the masking complexity. Indeed Definition 1 is software oriented and hence does not encompass the hardware case. As discussed above, the complexity of masking in hardware merely depends on the number of nonlinear gates [10, 15], that is on the number of nonlinear multiplications in the (n-variate) s-box representation over $\mathbb{F}_2$, the so-called *algebraic normal form*. One may also want to minimize the number of nonlinear multiplications in the (ℓ-variate) s-box representation over $\mathbb{F}_{2^k}$ for some k (and $\ell = \lceil n/k \rceil$). This approach has actually already been followed in [16], where Kim *et al.* speeds up the scheme in [29] by using the fact that the AES s-box can be processed with 5 nonlinear multiplications over $\mathbb{F}_{16}$ rather than 4 nonlinear multiplications over $\mathbb{F}_{256}$. Although requiring an additional nonlinear multiplication, the resulting implementation is faster since multiplications over $\mathbb{F}_{16}$ can be tabulated while multiplications over $\mathbb{F}_{256}$ are computed based on the slower log/alog approach. These observations motivate the following — more general — definition of the masking complexity.

Definition 3 (Masking Complexity). *Let m, n and k be three integers such that $m, k \leq n$. The* masking complexity *of a (n, m) s-box over $\mathbb{F}_{2^k}$ is the minimal number of nonlinear multiplications required to evaluate its polynomial representation over $\mathbb{F}_{2^k}$.*

Here again, the masking complexity is independent of the representation of $\mathbb{F}_{2^k}$ since one can go from one representation to another without any nonlinear multiplication. The issue of finding efficient methods with respect to the masking complexity over a smaller field $\mathbb{F}_{2^k}$ is left open for further researches.

7 Conclusion

In this paper we have introduced new generic higher-order masking schemes for s-boxes with efficient software implementation. Specifically, we have extended the Rivain and Prouff's approach for the AES s-box to any s-box. The method consists in masking the polynomial representation of the s-box over $\mathbb{F}_{2^n}$ where n is the input dimension. As argued, the complexity of this method

mainly depends on the number of nonlinear multiplications involved in the polynomial representation (*i.e.* multiplications which are not squares nor scalar multiplications). We have then introduced the masking complexity parameter for an s-box as the minimal number of nonlinear multiplications required for its evaluation. We have provided the exact values of this parameter for the set of power functions and upper bounds for all s-boxes. Namely, we have presented optimal methods to mask power functions and efficient heuristics for the general case. Eventually we have applied our schemes to the DES s-boxes and to the PRESENT s-box and we have provided implementation results. Our work stresses interesting open issues for further research. Among them the design of s-boxes taking into account the masking complexity criterion and the extension of our approach to masking over $\mathbb{F}_{2^k}$ with $k < n$ (*e.g.* for efficient hardware implementations) are of particular interest.

References

1. M.-L. Akkar, N. Courtois, R. Duteuil, and L. Goubin. A Fast and Secure Implementation of Sflash. In Y. Desmedt, editor, *Public Key Cryptography – PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 2003.
2. M.-L. Akkar and C. Giraud. An Implementation of DES and AES, Secure against Some Attacks. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.
3. G. Blakley. Safeguarding cryptographic keys. In *National Comp. Conf.*, volume 48, pages 313–317, New York, June 1979. AFIPS Press.
4. 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.
5. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
6. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.
7. 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.
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. J. Eve. The evaluation of polynomials. *Comm. ACM*, 6(1):17–21, 1964.
10. S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting Circuits from Leakage: the Computationally-Bounded and Noisy Cases. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 135–156. Springer, 2010.
11. FIPS PUB 46. *The Data Encryption Standard*. National Bureau of Standards, Jan. 1977.
12. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In Ç. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.
13. L. Genelle, E. Prouff, and M. Quisquater. Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems, 13th International Workshop – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.
14. L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.
15. 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.
16. H. Kim, S. Hong, and J. Lim. A Fast and Provably Secure Higher-Order Masking of AES S-Box. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems, 13th International Workshop – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2011.

17. D. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, third edition, 1988.
18. D. E. Knuth. Evaluation of polynomials by computers. *Comm. ACM*, 5(12):595–599, 1962.
19. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
20. 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.
21. 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.
22. 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.
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 Ç. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.
25. 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.
26. 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.
27. E. Prouff and M. Rivain. A Generic Method for Secure SBox Implementation. In S. Kim, M. Yung, and H.-W. Lee, editors, *Information Security Applications – WISA 2007*, volume 4867 of *Lecture Notes in Computer Science*, pages 227–244. Springer, 2008.
28. M. Rivain, E. Dottax, and E. Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In T. Baignères and S. Vaudenay, editors, *Fast Software Encryption – FSE 2008*, Lecture Notes in Computer Science, pages 127–143. Springer, 2008.
29. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
30. A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. In E. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.
31. 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.
32. A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.
33. J. von zur Gathen. Efficient and Optimal Exponentiation in Finite Fields. *Computational Complexity*, 1:360–394, 1991.

A Masking Complexity of Power Functions

Table 5 summarizes the masking complexity classes $(\mathcal{M}_k^n)_k$ for dimensions n in the set $\{3, 5, 7, 9, 10, 11\}$.

Table 5. Cyclotomic classes for $n \in \{3, 5, 7, 9, 10, 11\}$ w.r.t. the masking complexity k.

k	Cyclotomic classes in $\mathcal{M}_k^n$
	$n = 3$
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4\}$
1	$C_3 = \{3, 6, 5\}$
	$n = 5$
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8, 16\}$
1	$C_3 = \{3, 6, 12, 24, 17\}$, $C_5 = \{5, 10, 20, 9, 18\}$
2	$C_7 = \{7, 14, 28, 25, 19\}$, $C_{11} = \{11, 22, 13, 26, 21\}$, $C_{15} = \{15, 30, 29, 27, 23\}$
	$n = 7$
0	$C_0 = \{0\}$, $C_1 = \{1, 2, 4, 8, 16, 32, 64\}$
1	$C_3 = \{3, 6, 12, 24, 48, 96, 65\}$, $C_5 = \{5, 10, 20, 40, 80, 33, 66\}$, $C_9 = \{9, 18, 36, 72, 17, 34, 68\}$
2	$C_7 = \{7, 14, 28, 56, 112, 97, 67\}$, $C_{11} = \{11, 22, 44, 88, 49, 98, 69\}$, $C_{13} = \{13, 26, 52, 104, 81, 35, 70\}$, $C_{15} = \{15, 30, 60, 120, 113, 99, 71\}$, $C_{19} = \{19, 38, 76, 25, 50, 100, 73\}$, $C_{21} = \{21, 42, 84, 41, 82, 37, 74\}$, $C_{27} = \{27, 54, 108, 89, 51, 102, 77\}$, $C_{43} = \{43, 86, 45, 90, 53, 106, 85\}$
3	$C_{23} = \{23, 46, 92, 57, 114, 101, 75\}$, $C_{29} = \{29, 58, 116, 105, 83, 39, 78\}$, $C_{31} = \{31, 62, 124, 121, 115, 103, 79\}$, $C_{47} = \{47, 94, 61, 122, 117, 107, 87\}$, $C_{55} = \{55, 110, 93, 59, 118, 109, 91\}$, $C_{63} = \{63, 126, 125, 123, 119, 111, 95\}$
	$n = 9$
0	C_0, C_1
1	C_3, C_5, C_9, C_{17}
2	C_7, C_{11}, C_{13}, C_{15}, C_{19}, C_{21}, C_{25}, C_{27}, C_{35}, C_{37}, C_{41}, C_{45}, C_{51}, C_{73}, C_{75}, C_{83}, C_{85}
3	C_{23}, C_{29}, C_{31}, C_{39}, C_{43}, C_{47}, C_{53}, C_{55}, C_{57}, C_{59}, C_{61}, C_{63}, C_{75}, C_{77}, C_{79}, C_{87}, C_{91}, C_{93}, C_{95}, C_{103}, C_{107}, C_{109}, C_{111}, C_{117}, C_{119}, C_{123}, C_{125}, C_{127}, C_{171}, C_{175}, C_{183}, C_{187}, C_{219}
4	C_{191}, C_{223}, C_{239}
	$n = 10$
0	C_0, C_1
1	C_3, C_5, C_9, C_{17}, C_{33}
2	C_7, C_{11}, C_{13}, C_{15}, C_{19}, C_{21}, C_{25}, C_{27}, C_{35}, C_{37}, C_{41}, C_{45}, C_{49}, C_{51}, C_{69}, C_{73}, C_{85}, C_{99}, C_{147}, C_{165}
3	C_{23}, C_{29}, C_{31}, C_{39}, C_{43}, C_{47}, C_{53}, C_{55}, C_{57}, C_{59}, C_{61}, C_{63}, C_{71}, C_{75}, C_{77}, C_{79}, C_{83}, C_{87}, C_{89}, C_{91}, C_{93}, C_{95}, C_{101}, C_{103}, C_{105}, C_{107}, C_{109}, C_{111}, C_{115}, C_{117}, C_{119}, C_{121}, C_{123}, C_{125}, C_{149}, C_{151}, C_{155}, C_{157}, C_{167}, C_{171}, C_{173}, C_{175}, C_{179}, C_{181}, C_{183}, C_{187}, C_{189}, C_{205}, C_{207}, C_{213}, C_{215}, C_{219}, C_{221}, C_{231}, C_{235}, C_{237}, C_{245}, C_{255}, C_{341}, C_{347}, C_{363}, C_{447}, C_{495}
4	C_{127}, C_{159}, C_{191}, C_{223}, C_{239}, C_{247}, C_{251}, C_{253}, C_{343}, C_{351}, C_{367}, C_{375}, C_{379}, C_{383}, C_{439}, C_{479}, C_{511}
	$n = 11$
0	C_0, C_1
1	C_3, C_5, C_9, C_{17}, C_{33}
2	C_7, C_{11}, C_{13}, C_{15}, C_{19}, C_{21}, C_{25}, C_{27}, C_{35}, C_{37}, C_{41}, C_{45}, C_{49}, C_{51}, C_{67}, C_{69}, C_{73}, C_{81}, C_{85}, C_{99}, C_{137}, C_{153}, C_{163}, C_{165}, C_{293}
3	C_{23}, C_{29}, C_{31}, C_{39}, C_{43}, C_{47}, C_{53}, C_{55}, C_{57}, C_{59}, C_{61}, C_{63}, C_{71}, C_{75}, C_{77}, C_{79}, C_{83}, C_{87}, C_{89}, C_{91}, C_{93}, C_{95}, C_{101}, C_{103}, C_{105}, C_{107}, C_{109}, C_{111}, C_{113}, C_{115}, C_{117}, C_{119}, C_{121}, C_{123}, C_{125}, C_{139}, C_{141}, C_{143}, C_{147}, C_{149}, C_{151}, C_{155}, C_{157}, C_{167}, C_{169}, C_{171}, C_{173}, C_{175}, C_{179}, C_{181}, C_{185}, C_{187}, C_{189}, C_{199}, C_{201}, C_{203}, C_{205}, C_{207}, C_{211}, C_{213}, C_{217}, C_{219}, C_{221}, C_{229}, C_{231}, C_{243}, C_{245}, C_{255}, C_{295}, C_{299}, C_{301}, C_{307}, C_{309}, C_{311},C_{315}, C_{317}, C_{331}, C_{333}, C_{335}, C_{343}, C_{347}, C_{359}, C_{363}, C_{365}, C_{379}, C_{411}, C_{423}, C_{427}, C_{429}, C_{339}, C_{341}, C_{437}, C_{439}, C_{469}, C_{495}, C_{683}, C_{703}, C_{879}, C_{887}
4	C_{127}, C_{159}, C_{183}, C_{191}, C_{215}, C_{223}, C_{233}, C_{235}, C_{237}, C_{239}, C_{247}, C_{249}, C_{251}, C_{253}, C_{303}, C_{319}, C_{349}, C_{351}, C_{367}, C_{371}, C_{373}, C_{375}, C_{381}, C_{383}, C_{413}, C_{415}, C_{431}, C_{443}, C_{445}, C_{447}, C_{463}, C_{471}, C_{475}, C_{477}, C_{479}, C_{491}, C_{493}, C_{501}, C_{503}, C_{507}, C_{509}, C_{511}, C_{687}, C_{695}, C_{699}, C_{727}, C_{731}, C_{735}, C_{751}, C_{759}, C_{763}, C_{767}, C_{895}, C_{959}, C_{991}, C_{1023}

B Detailed Algorithms

We detail hereafter the algorithms to perform a masked DES/PRESENT s-box computation. As in [29], both algorithms use the ISW-based masked field multiplication (SecMult) and the mask refreshing procedure (RefreshMasks). We recall these procedures before giving the masked s-box algorithms.

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

Input: shares a_i satisfying $\sum_i a_i = a$, shares b_i satisfying $\sum_i b_i = b$
Output: shares c_i satisfying $\sum_i c_i = ab$

```
for i = 0 to d
    for j = i + 1 to d
        r_{i,j} <-$ {0,1}^n
        r_{j,i} <- (r_{i,j} + a_i b_j) + a_j b_i
    endfor
endfor
for i = 0 to d
    c_i <- a_i b_i
    for j = 0 to d, j ≠ i do c_i <- c_i + r_{i,j}
endfor
return (c_0, ..., c_d)
```

Algorithm 2 RefreshMasks

Input: shares x_i satisfying $\sum_i x_i = x$
Output: shares x_i satisfying $\sum_i x_i = x$

```
for i = 1 to d
    tmp <-$ {0,1}^n
    x_0 <- x_0 + tmp
    x_i <- x_i + tmp
endfor
```

Algorithm 3 Secure higher-order PRESENT s-box evaluation

Input: a dth-order encoding $(x_0, \ldots, x_d)$ of $x \in \{0,1\}^4$, look-up tables for the L_i
Output: a dth-order encoding $(t_0, \ldots, t_d)$ of $\mathrm{S}(x)$

1. **for** $i = 0$ **to** d **do** $y_{2,i} \leftarrow x_i^2$
2. RefreshMasks$(y_{2,0}, \ldots, y_{2,d})$
3. $(y_{3,0}, \ldots, y_{3,d}) \leftarrow \mathsf{SecMult}\big((x_0, \ldots, x_d), (y_{2,0}, \ldots, y_{2,d})\big)$
4. $(y_{5,0}, \ldots, y_{5,d}) \leftarrow \mathsf{SecMult}\big((y_{2,0}, \ldots, y_{2,d}), (y_{3,0}, \ldots, y_{3,d})\big)$
5. $(y_{7,0}, \ldots, y_{7,d}) \leftarrow \mathsf{SecMult}\big((y_{2,0}, \ldots, y_{2,d}), (y_{5,0}, \ldots, y_{5,d})\big)$
6. $t_0 \leftarrow a_0 + L_1(x_0) + L_3(y_{3,0}) + L_5(y_{5,0}) + L_7(y_{7,0})$
7. **for** $i = 1$ **to** d **do** $t_i \leftarrow L_1(x_i) + L_3(y_{3,i}) + L_5(y_{5,i}) + L_7(y_{7,i})$
8. **return** $(t_0, \ldots, t_d)$

Algorithm 4 Secure higher-order DES s-box evaluation

Input: a dth-order encoding $(x_0, \ldots, x_d)$ of $x \in \{0,1\}^6$, a table of coefficients $a_{i,j}$ for Q_i polynomials
Output: a dth-order encoding $(t_0, \ldots, t_d)$ of $\mathrm{S}(x)$

[**Computing the x^{8j} powers**]
1. **for** $i = 0$ **to** d **do** $y_{8,i} \leftarrow x_i^8$
2. $\mathsf{RefreshMasks}(y_{8,0}, \ldots, y_{8,d})$
3. **for** $i = 0$ **to** d **do** $y_{16,i} \leftarrow y_{8,i}^2$
4. $\mathsf{RefreshMasks}(y_{16,0}, \ldots, y_{16,d})$
5. $(y_{24,0}, \ldots, y_{24,d}) \leftarrow \mathsf{SecMult}\big((y_{8,0}, \ldots, y_{8,d}), (y_{16,0}, \ldots, y_{16,d})\big)$
6. **for** $i = 0$ **to** d **do** $y_{32,i} \leftarrow y_{16,i}^2$
7. $\mathsf{RefreshMasks}(y_{32,0}, \ldots, y_{32,d})$
8. $(y_{40,0}, \ldots, y_{40,d}) \leftarrow \mathsf{SecMult}\big((y_{8,0}, \ldots, y_{8,d}), (y_{32,0}, \ldots, y_{32,d})\big)$
9. **for** $i = 0$ **to** d **do** $y_{48,i} \leftarrow y_{24,i}^2$
10. $\mathsf{RefreshMasks}(y_{48,0}, \ldots, y_{48,d})$
11. $(y_{56,0}, \ldots, y_{56,d}) \leftarrow \mathsf{SecMult}\big((y_{8,0}, \ldots, y_{8,d}), (y_{48,0}, \ldots, y_{48,d})\big)$

[**Evaluating the $Q_i(x^8)$ polynomials**]
12. **for** $i = 0$ **to** 7 **do**
13. | $q_{i,0} \leftarrow a_{i,0}$
14. | **for** $k = 1$ **to** d **do** $q_{i,k} \leftarrow 0$
15. | **for** $j = 1$ **to** 7 **do**
16. | | **for** $k = 0$ **to** d **do** $q_{i,k} \leftarrow q_{i,k} + a_{i,j} \cdot y_{8j,k}$
17. | **endfor**
18. | $\mathsf{RefreshMasks}(q_{i,0}, \ldots, q_{i,d})$
19. **endfor**

[**Evaluating** $\mathrm{S}(x)$]
20. **for** $i = 0$ **to** d **do** $y_{2,i} \leftarrow x_i^2$
21. $\mathsf{RefreshMasks}(y_{2,0}, \ldots, y_{2,d})$
22. **for** $i = 0$ **to** d **do** $y_{4,i} \leftarrow y_{2,i}^2$
23. $\mathsf{RefreshMasks}(y_{4,0}, \ldots, y_{4,d})$
24. $(t_0, \ldots, t_d) \leftarrow \mathsf{SecMult}\big((y_{4,0}, \ldots, y_{4,d}), (q_{7,0}, \ldots, q_{7,d})\big)$
25. **for** $i = 0$ **to** d **do** $t_i \leftarrow t_i + q_{6,i}$
26. $(t_0, \ldots, t_d) \leftarrow \mathsf{SecMult}\big((y_{2,0}, \ldots, y_{2,d}), (t_0, \ldots, t_d)\big)$
27. $(s_0, \ldots, s_d) \leftarrow \mathsf{SecMult}\big((y_{4,0}, \ldots, y_{4,d}), (q_{5,0}, \ldots, q_{5,d})\big)$
28. **for** $i = 0$ **to** d **do** $t_i \leftarrow t_i + s_i + q_{4,i}$
29. $(t_0, \ldots, t_d) \leftarrow \mathsf{SecMult}\big((x_0, \ldots, x_d), (t_0, \ldots, t_d)\big)$
30. $(r_0, \ldots, r_d) \leftarrow \mathsf{SecMult}\big((y_{4,0}, \ldots, y_{4,d}), (q_{3,0}, \ldots, q_{3,d})\big)$
31. **for** $i = 0$ **to** d **do** $r_i \leftarrow r_i + q_{2,i}$
32. $(r_0, \ldots, r_d) \leftarrow \mathsf{SecMult}\big((y_{2,0}, \ldots, y_{2,d}), (r_0, \ldots, r_d)\big)$
33. $(p_0, \ldots, p_d) \leftarrow \mathsf{SecMult}\big((y_{4,0}, \ldots, y_{4,d}), (q_{1,0}, \ldots, q_{1,d})\big)$
34. **for** $i = 0$ **to** d **do** $t_i \leftarrow t_i + r_i + p_i + q_{0,i}$

35. **return** $(t_0, \ldots, t_d)$

C Look-Up Tables

We detail hereafter the different look-up tables used in our implementations (in C syntax). For the DES s-boxes, the 4-bit outputs are embedded into $\mathbb{F}_{64}$ by padding their most significant bits with two 0s. The used field representations are $\mathbb{F}_{64} \equiv \mathbb{F}_2[x]/(1 + x^5 + x^6)$ for the DES s-boxes and $\mathbb{F}_{16} \equiv \mathbb{F}_2[x]/(1 + x^3 + x^4)$ for the PRESENT s-box (such that $\texttt{MultGF16}[a||b] = a \times b$). The multiplication over $\mathbb{F}_{64}$ is performed using the log/alog tables given in Figure 1 while the multiplication over $\mathbb{F}_{16}$ is performed with the multiplication table given in Figure 2. The $a_{i,j}$

coefficients for the masked computation of the first DES s-box are given in Figure 3.[8] Eventually the L_i transformations for the PRESENT s-box are given in table 4.

```
unsigned char* LogGF64[64] = {
   63, 0, 1, 58, 2, 53, 59, 39,
   3, 34, 54, 18, 60, 31, 40, 48,
   4, 43, 35, 22, 55, 15, 19, 26,
   61, 51, 32, 29, 41, 13, 49, 11,
   5, 6, 44, 7, 36, 45, 23, 8,
   56, 37, 16, 46, 20, 24, 27, 9,
   62, 57, 52, 38, 33, 17, 30, 47,
   42, 21, 14, 25, 50, 28, 12, 10
};
unsigned char* AlogGF64[64] = {
   1, 2, 4, 8, 16, 32, 33, 35,
   39, 47, 63, 31, 62, 29, 58, 21,
   42, 53, 11, 22, 44, 57, 19, 38,
   45, 59, 23, 46, 61, 27, 54, 13,
   26, 52, 9, 18, 36, 41, 51, 7,
   14, 28, 56, 17, 34, 37, 43, 55,
   15, 30, 60, 25, 50, 5, 10, 20,
   40, 49, 3, 6, 12, 24, 48, 1
};
```

Fig. 1. Log/alog tables for the multiplication for the multiplication over $\mathbb{F}_{64}$.

```
unsigned char* MultGF16[16][16] = {
   {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7},
   {8, 9, 10, 11, 12, 13, 14, 15, 0, 2, 4, 6, 8, 10, 12, 14, 9, 11, 13, 15},
   {1, 3, 5, 7, 0, 3, 6, 5, 12, 15, 10, 9, 1, 2, 7, 4, 13, 14, 11, 8, 0, 4},
   {8, 12, 9, 13, 1, 5, 11, 15, 3, 7, 2, 6, 10, 14, 0, 5, 10, 15, 13, 8, 7},
   {2, 3, 6, 9, 12, 14, 11, 4, 1, 0, 6, 12, 10, 1, 7, 13, 11, 2, 4, 14, 8},
   {3, 5, 15, 9, 0, 7, 14, 9, 5, 2, 11, 12, 10, 13, 4, 3, 15, 8, 1, 6, 0, 8},
   {9, 1, 11, 3, 2, 10, 15, 7, 6, 14, 4, 12, 13, 5, 0, 9, 11, 2, 15, 6, 4},
   {13, 7, 14, 12, 5, 8, 1, 3, 10, 0, 10, 13, 7, 3, 9, 14, 4, 6, 12, 11, 1},
   {5, 15, 8, 2, 0, 11, 15, 4, 7, 12, 8, 3, 14, 5, 1, 10, 9, 2, 6, 13, 0},
   {12, 1, 13, 2, 14, 3, 15, 4, 8, 5, 9, 6, 10, 7, 11, 0, 13, 3, 14, 6},
   {5, 8, 12, 1, 15, 2, 10, 7, 9, 4, 0, 14, 5, 11, 10, 4, 15, 1, 13, 3, 8},
   {6, 7, 9, 2, 12, 0, 15, 7, 8, 14, 1, 9, 6, 5, 10, 2, 13, 11, 4, 12, 3}
};
```

Fig. 2. Look-up table for the multiplication over $\mathbb{F}_{16}$.

[8] Due to length constraints we could not include the coefficients for all the DES s-boxes. They will be given in an extended version of the paper.

```
unsigned char* A[8][8] = {
   {14, 5, 58, 31, 39, 36, 47, 54},
   {52, 3, 47, 28, 1, 53, 9, 52},
   {63, 7, 7, 6, 3, 1, 48, 49},
   {49, 12, 0, 9, 33, 50, 49, 3},
   {26, 3, 40, 0, 13, 8, 35, 59},
   {35, 31, 13, 38, 27, 29, 62, 61},
   {12, 27, 49, 46, 40, 50, 28, 41},
   {40, 14, 36, 44, 27, 27, 33, 0}
};
```

Fig. 3. Look-up table for the $a_{i,j}$ coefficients.

```
unsigned char* L1[16] = {0,14,13,3,3,13,14,0,8,6,5,11,11,5,6,8};
unsigned char* L3[16] = {0,14,8,6,10,4,2,12,15,1,7,9,5,11,13,3};
unsigned char* L5[16] = {0,3,4,7,0,3,4,7,11,8,15,12,11,8,15,12};
unsigned char* L7[16] = {0,10,0,10,14,4,14,4,4,14,4,14,10,0,10,0};
```

Fig. 4. Look-up tables for the L_i transformations.