Efficient Masked S-Boxes Processing – A Step Forward –

Vincent Grosso[1], Emmanuel Prouff[2], François-Xavier Standaert[1]

[1] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.
[2] ANSSI, 51 Bd de la Tour-Maubourg, 75700 Paris 07 SP, France.

Abstract. To defeat side-channel attacks, the implementation of block cipher algorithms in embedded devices must include dedicated countermeasures. To this end, security designers usually apply secret sharing techniques and build masking schemes to securely operate an shared data. The popularity of this approach can be explained by the fact that it enables formal security proofs. The construction of masking schemes thwarting *higher-order* side-channel attacks, which correspond to a powerful adversary able to exploit the leakage of the different shares, has been a hot topic during the last decade. Several solutions have been proposed, usually at the cost of significant performance overheads. As a result, the quest for efficient masked S-box implementations is still ongoing. In this paper, we focus on the scheme proposed by Carlet *et al* at FSE 2012, and latter improved by Roy and Vivek at CHES 2013. This scheme is today the most efficient one to secure a generic S-box at any order. By exploiting an idea introduced by Coron *et al* at FSE 2013, we show that Carlet *et al*'s scheme can still be improved for S-boxes with input dimension larger than four. We obtain this result thanks to a new definition for the addition-chain exponentiation used during the masked S-box processing. For the `AES` and `DES` S-boxes, we show that our improvement leads to significant efficiency gains.

1 Introduction

Side-channel attacks (SCA) are a class of attacks, where the attacker has access to some *leakages* about the internal state during the computation [16]. In practice, such scenario makes it possible to attack implementations that are believed secure against classical (black-box) cryptanalyses. To defeat SCA, implementations of cryptographic algorithms must embed appropriate countermeasures. This is actually mandatory for implementations dedicated to smart card payments, pay-TV applications or citizen authentication with e-Passports.

Securing block cipher implementations has been a long-standing issue for the embedded systems industry. A sound approach is to use *secret sharing* [2, 24], often called *masking* in the context of side-channel attacks [6]. The principle is to split every *sensitive* variable[1] x occurring during the computation into $d+1$

[1] A variable is said to be *sensitive* in an SCA context if it functionally depends on both a public variable and a secret whose size is small enough to enable exhaustive search.

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 (*i.e.* $\perp$ will be the bitwise addition $\oplus$). 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). The full tuple $(x_i)_i$ is further called a *dth-order encoding of* x. When d random masks are involved per sensitive variable, the masking is said to be *of order* d. It has been shown that the complexity of mounting a successful side-channel attack against a masked implementation increases exponentially with the masking order [6, 10, 19]. Starting from this observation, the design of efficient masking schemes for different ciphers has become a foreground issue.

Higher-Order Masking Schemes. A *higher-order secure (masking) 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 exploiting information on less than d intermediate results can succeed. As argued in several previous works (see *e.g.* [21] or [23]), the main difficulty in designing higher-order secure schemes for block ciphers lies in masking the *S-box(es)*, which are the only internal primitives that perform non-linear operations.

Masking and S-Boxes. Whereas many solutions have been proposed to deal with the case of first-order masking (see *e.g.* [3, 18]), only few solutions exist for the higher-order case. A scheme has been proposed by Schramm and Paar in [23] which generalizes the (first-order) table re-computation method described in [18]. 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 [20] with formal security proofs but still limited to 2nd-order security.

To the best of our knowledge, four approaches currently exist which enable the design of dth-order secure masking schemes for any arbitrary chosen d. One is due to Genelle *et al* and consists in mixing additive and multiplicative sharings (namely to use alternatively (1) for $\perp = \oplus$ and $\perp = \times$). This scheme is primarily dedicated to the `AES` algorithm and seems difficult to generalize efficiently to other block ciphers where the S-box is not affinely equivalent to a power function. The second one is due to Prouff and Roche and it relies on solutions developed in secure *multi-party computation* [1]. It is much less efficient than the other schemes (see *e.g.* [12]) but, contrary to them, remains secure even in presence of hardware glitches [17]. The third approach has been recently

proposed by Coron in [7]. The core idea is to represent the S-box by several look-up tables which are regenerated from fresh random masks and the S-box truth table, each time a new S-box processing must be done. It extends the *table re-computation technique* introduced in the original paper by Kocher *et al* [16]. The security of Coron's scheme against higher-order SCA is formally proved under the assumption that the variable shares leak independently. Its asymptotic timing complexity is quadratic in the number of shares and can be applied to any S-box. However, the `RAM` memory consumption to secure (at order d) an S-box with input (resp. output) dimension n (resp. m) is $m(d+1)2^n$ bits, which can quickly exceed the memory capacity of the hosted device.

The three methods recalled in previous paragraph have important limitations which strongly impact their practicability: the first method is hardly generalizable to any S-box and the two other ones have a large extra cost (in terms of either processing or memory complexity). Actually, when the S-box to secure is not a power function and has input/output dimensions close to 8, the fourth approach is the most practical one when d is greater than or equal to 3. This approach, proposed in [5], generalizes the study conducted in [21] for power functions. The core idea is to split the S-box processing into a sequence of field multiplications and $\mathbb{F}_2$-linear operations, and then to secure both operations independently. The complexity of the masking schemes for the multiplication and a $\mathbb{F}_2$-linear operation[2] is $O(d^2)$ and $O(d)$ respectively. Moreover, the constant terms in these complexities are (usually) significantly greater for the multiplication than for the $\mathbb{F}_2$-linear operations. Based on this observation, the authors of [5] propose to look for S-box representations that minimize the number of field multiplications which are not $\mathbb{F}_2$-linear[3] (this kind of multiplication shall be called *non-linear* in this paper). This led them to introduce the notion of *S-box masking complexity*, which corresponds to the minimal number of non-linear multiplications needed to evaluate the S-box. This complexity is evaluated for any power function defined in $\mathbb{F}_{2^n}$ with $n \leq 10$ (in particular, the complexity of $x \in \mathbb{F}_{2^8} \mapsto x^{254}$, which is the non-linear part of the `AES` S-box, is shown to be equal to 4). Tight upper bounds on the masking complexity are also given for any random S-box. The work of Carlet *et al* has been further improved by Roy and Vivek in [22], where it is in particular shown that the masking complexity of the `DES` S-boxes is lower-bounded by 3. The authors of [22] also present a method that requires 7 non-linear multiplications. Another improvement of [5] has been proposed in [9], where it is shown that it is possible to improve the processing of the non-linear multiplications with the particular form $x \times g(x)$

[2] A function f is $\mathbb{F}_2$-linear if it satisfies $f(x \oplus y) = f(x) \oplus f(y)$ for any pair (x, y) of elements in its domain. This property must not be confused with linearity of a function which is defined such that $f(ax \oplus by) = af(x) \oplus bf(y)$. A linear function is $\mathbb{F}_2$-linear but the converse is false in general (the homogeneity of degree 1 must indeed be also satisfied for the converse to be true).

[3] A multiplication over a field of characteristic 2 is $\mathbb{F}_2$-linear if it corresponds to a squaring.

with g being $\mathbb{F}_2$-linear. This type of multiplication is called *bilinear* in the rest of the paper[4].

Our Contribution. In this paper we refine the notion of S-box masking complexity introduced in [5] and further studied in [22]. We still link it to the minimum number of non-linear multiplications needed to evaluate the S-box, but we don't include bilinear multiplications in this counting. We justify this choice thanks to the analysis in [9] which shows that the complexity of the latter multiplications is between that of general non-linear multiplications (costly) and that of $\mathbb{F}_2$-linear multiplications (cheap). For all exponentiations in $\mathbb{F}_{2^n}$, with $n \in \{4, 6, 8\}$, we give the new masking complexities and we afterwards illustrate, for the `AES` and `DES` S-boxes, the effective gain obtained by using the corresponding new *addition-chain exponentiation* [11]. This works raises the need for new polynomial evaluation algorithms minimizing the number of multiplications which are neither linear nor bilinear. It could also be of interest to study whether specialized (efficient) schemes cannot be dedicated to the secure processing of other types of non-linear multiplications (which are not linear or bilinear but have some helpful properties).

2 Existing Schemes for Elementary operations

In this section, it is assumed that the S-box to protect manipulates data of bit-length n (typically $n \in \{4, 8, 16\}$). Depending on the kind of operation to process, these data can be viewed as elements of the vector space $\mathbb{F}_2^n$ defined over the field $(\mathbb{F}_2, \oplus, \&)$, where $\oplus$ is the `XOR` operation and $\&$ the `AND` operator. Or, they can be defined as elements of the field $\mathbb{F}_{2^n} \cong (\mathbb{F}_2[X]/p(X), \oplus, \times)$, where $p(X)$ is an irreducible polynomial of degree n and $\times$ denotes the polynomial multiplication modulo $p(X)$.

As recalled in the previous section, the most efficient solution which today exists to secure an S-box against higher-order SCA is to rewrite it as a polynomial function over $\mathbb{F}_{2^n}$ and to split its evaluation as a sequence of $\mathbb{F}_2$-linear operations and multiplications. Indeed, whatever d, d^{th}-order secure schemes exist for these two types of operations. We recall them hereafter. Note that, some mask refreshing must sometimes be done between different calls to these algorithms in order to guaranty the security of the whole process. Since, mask refreshing has a minor impact on the efficiency improvement proposed in this paper we do not recall here the mask refreshing algorithm and we exclude it from the description of the S-box secure evaluation procedures in Section 3 (for more details about this point we suggest the reading of [9, 21]).

[4] We chose this term because the multiplication $y \times g(x)$, viewed as a function over $\mathbb{F}_{2^n} \times \mathbb{F}_{2^n}$, is indeed $\mathbb{F}_2$-bilinear when g is $\mathbb{F}_2$-linear. For such a function g, it may be checked that the *algebraic degree* [4, Chapter 9] of $x \mapsto x \times g(x)$, viewed as a vectorial function, is quadratic.

$\mathbb{F}_2$-*linear operation.* To securely process a $\mathbb{F}_2$-linear function g on a data x encoded by the tuple $(x_i)_i$, we just need to evaluate the function on each share x_i separately. The sharing $(g(x_i))_i$ is indeed an encoding of $g(x)$.

Algorithm 1 Secure evaluation of a $\mathbb{F}_2$-linear function g

Require: Shares $(x_i)_i$ satisfying $\oplus_i x_i = x$.
Ensure: Shares $(y_i)_i$ satisfying $\oplus_i y_i = g(x)$.
1: **for** i from 0 to d **do**
2: $\quad y_i \leftarrow g(x_i)$
3: **end for**

In particular, Algorithm 1 can be applied to secure the Frobenius endomorphism over the field $\mathbb{F}_{2^n}$ (i.e. the squaring in characteristic 2) as this operation is $\mathbb{F}_2$-linear.

Multiplication. To securely process the multiplication between two sensitive variables x and y encoded by $(x_i)_i$ and $(y_i)_i$ respectively, the following algorithm has been proposed in [13] (and generalized in [21]).

Algorithm 2 Multiplication of two masked secrets x and y

Require: Shares $(x_i)_i$ and $(y_i)_i$ satisfying $\oplus_i x_i = x$ and $\oplus_i y_i = y$
Ensure: Shares $(w_i)_i$ satisfying $\oplus_i w_i = x \times y$
1: **for** i from 0 to d **do**
2: $\quad$ **for** j from $i+1$ to d **do**
3: $\quad\quad r_{i,j} \in_R \mathbb{F}_{2^n}$
4: $\quad\quad r_{i,j} \leftarrow (r_{i,j} \oplus x_i \times y_j) \oplus x_j \times y_i$
5: $\quad$ **end for**
6: **end for**
7: **for** i from 0 to d **do**
8: $\quad w_i \leftarrow x_i \times y_i$
9: $\quad$ **for** j from 0 to d, $j \neq i$ **do**
10: $\quad\quad w_i \leftarrow w_i \oplus r_{i,j}$
11: $\quad$ **end for**
12: **end for**

Remark 1. The order of the `XOR`s operations in Step 4 must be respected for the security guarantee to hold.

Starting from Lagrange's interpolation formula, [5] and [22] introduce S-box evaluation techniques which are only based on Algorithms 1 and 2 (and a third algorithm used to refresh the sharings when the input sharings correspond to dependent variables). Because the complexity of the d^{th}-order secure multiplication is quadratic, whereas that of an $\mathbb{F}_2$-linear function is linear, the polynomial

evaluation strategies try to minimize the number of calls to Algorithm 2. However, Coron *et al* have recently shown that multiplications of the form $x \times g(x)$, with g being $\mathbb{F}_2$-linear, can be securely evaluated more efficiently than standard multiplications [9]. This observation naturally raises the following new question: can we improve the complexities of the S-box evaluation strategies in [5, 22] by replacing, as much as possible, standard multiplications by multiplications in the form $x \times g(x)$. Before dealing with this question, let us first recall the particular multiplication proposed in [9].

Multiplications of the form $x \times g(x)$, with g $\mathbb{F}_2$-linear. To securely process this type of multiplication, the following algorithm is proposed in [9].

Algorithm 3 Secure evaluation of a product of $h(x) = x \times g(x)$

Require: shares $(x_i)_i$ satisfying $\oplus_i x_i = x$.
Ensure: shares $(y_i)_i$ satisfying $\oplus_i y_i = h(x)$.
1: **for** i from 0 to d **do**
2: **for** j from $i+1$ to d **do**
3: $r_{i,j} \in_R \mathbb{F}_{2^n}$
4: $r'_{i,j} \in_R \mathbb{F}_{2^n}$
5: $t \leftarrow r_{i,j}$
6: $t \leftarrow t \oplus h(x_i \oplus r'_{i,j})$
7: $t \leftarrow t \oplus h(x_j \oplus r'_{i,j})$
8: $t \leftarrow t \oplus h((x_i \oplus r'_{i,j}) \oplus x_j)$
9: $t \leftarrow t \oplus h(r'_{i,j})$
10: $r_{j,i} \leftarrow t$
11: **end for**
12: **end for**
13: **for** i from 0 to d **do**
14: $y_i \leftarrow h(x_i)$
15: **for** j from 0 to d, $j \neq i$ **do**
16: $y_i \leftarrow y_i \oplus r_{i,j}$
17: **end for**
18: **end for**

Notation. In the particular case where g is an exponentiation by a power of 2, say $g(x) = x^{2^s}$, the function h is denoted by h_{s+1}.

The complexity of Algorithm 3 is still quadratic but, for many typical application contexts, the constant terms are much smaller than in Algorithm 2. Indeed, the processing of h can be tabulated on standard embedded processors as long as $n \leqslant 10$, whereas the field multiplications $\times$ occurring in Algorithm 2 cannot if $n \geqslant 5$. In the following, functions/operations which can be evaluated thanks to Algorithm 1 or Algorithm 2 will be said to be of Type-I or Type-III respectively. Functions of the form $x \times g(x)$ with g $\mathbb{F}_2$-linear will be said to be of

Type-II. Table 1 summarizes the cost of the three algorithms in term of XORs, field multiplications and look-up table accesses (referred to as LUT access).

Table 1. Cost of different algorithms.

	XOR	Multiplication	LUT access
Algorithm 1	0	0	$d+1$
Algorithm 2	$2d^2+2d$	d^2+2d+1	0
Algorithm 3	$5d^2+5d$	0	$2d^2+3d+1$

In most of classical architectures, a memory access (or a XOR) can be done in 1 or 2 CPU clock cycles, whereas the processing of a field multiplication with the CPU instructions set only requires between 20 and 40 cycles (we recall some classical field multiplication algorithms in Appendix A). This explains why the replacing of Type-III operations by Type-II ones leads to a significant efficiency improvement when $n \in [5; 10]$. Based on this observation, we propose in the next section new sequences of operations that lead to practically more efficient processing of power functions than the state of the art solutions [5, 22].

3 New Proposal for Power Functions Evaluation

Considering the fact that the processing of power functions in the form x^{1+2^s} (which corresponds to the Type-II operation $x \times x^{2^s}$) is more efficient than that of other power functions, we followed an approach close to [5] in order to exhibit the most efficient processing for any power function defined in $\mathbb{F}_{2^n}$ for $n \leq 8$. Namely, for every power function $x \mapsto x^\alpha$, we exhibit by exhaustive search a sequence of operations of types I, II and III, which minimizes first the number of Type III operations, and then the number of Type II operations. This amounts to find, for each exponent α, the shortest *addition chain*[5] [15] with the supplementary constraint that multiplications by 2^t, for any integer t, or additions in the form $v + 2^t v$ are for free. We recall that an addition chain for $\alpha \in \mathbb{N}$ is an increasing sequence of integers v_0, ..., v_s such that $v_0 = 1$, $v_s = \alpha$ and for any $j \neq 0$ there exist two indices $i < j$ and $k < j$ (not necessary different) s.t. $v_j = v_i + v_k$. The length of such a sequence is defined as the total number of additions (including multiplications by 2) needed to get $v_s = \alpha$ from $v_0 = 1$, with only operations between elements of the sequence. The definition of length used in [5, 22] excludes multiplications by 2. For the reasons discussed previously, we extend the classical definition of the addition chain by adding the operation $v \mapsto (1 + 2^t)v$ for any integer t. We moreover assume that this operation is also excluded from the sequence length definition (it indeed corresponds to the function $h_{t+1} : x^v \mapsto x^{(1+2^t)v}$). The corresponding new length definition is referred to as *extended*

[5] In the context of exponentiation processing, these chains are sometimes also referred to as *addition-chain exponentiation* (see for instance [11]).

length in the following. Our purpose is to minimize it. This point is the main (and important) difference with the (shortest) sequences investigated in [5]. Our results are given[6] in Table 2 for $n = 8$, where the exponents are grouped into classes. Each class, say C_j, corresponds to the set of exponents which can be obtained by multiplying j by a power of 2 (modulo $2^n - 1$).

Table 2. Smallest cost to process x^α with operations of types II and III.

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

Remark 2. As the cost of Type-I operations is negligible compared to the cost of operations of types II and III, we chose to not give them in Table 2.

[6] Tables for the cases $n = 4, 6$ are given in Appendix B.

Remark 3. The costs given in Table 2 have been obtained by first minimizing the global number of Type-II and Type-III operations, and then by minimizing the number of Type-III multiplications. It can be noticed that other minimization strategies could be applied. For instance, if the goal is to minimize the number of Type-III multiplications, then it can be checked that x^{254} can be evaluated without such operation: first process x^{63}, then $(x+x^{63})^3 = x^{189}+x^{127}+x^{65}+x^3$, end eventually process x^{189}, x^{65} and x^3, and subtract them to $(x+x^{63})^3$ to get $x^{254} = (x^{127})^2$ (which gives a processing without Type-III operations and 9 Type-II operations).

For the exponentiation $x \mapsto x^{254}$ (the non-linear part of the `AES` S-box), we found the extended addition chain $(1, 2, 5, 25, 125, 127, 254)$ whose extended length is 1. This sequence indeed requires only 1 operation of Type-III(+) (to get 127), 2 operations of Type-I ($\times$) (to get and 2 and 254) and 3 operations of Type-II ($\times(1+2^2)$) (to get 5, 25 and 125)). It may moreover be observed that the sequence involves the same operation $v \mapsto (1+2^t)v$ (for $t = 2$) each time, which reduces the memory required to implement the solution.

The extended addition chain used for the `AES` S-box is represented in Figure 1.

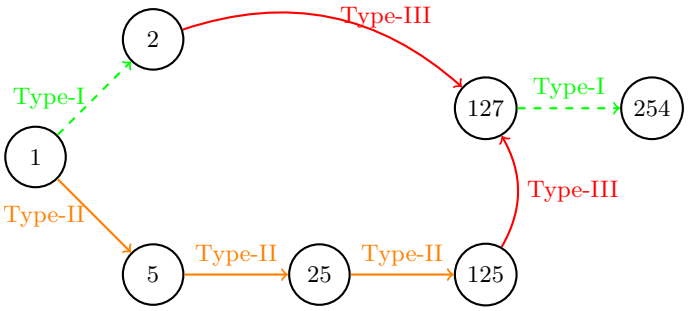

Fig. 1. `AES` S-box extended addition chain.

Algorithm 4 shows how to use the extended addition chain to calculate invert in the field $\mathbb{F}_{2^8}$.

For `DES`, we take advantage of the S-box representation proposed in [22]. In that paper it has been shown that all `DES` S-boxes can be calculated with 7 non-linear multiplications. They indeed can be represented by a polynomial of the form:

$$P_{\texttt{DES}}(x) = (x^{36} + p_1(x)) \times (((x^{18} + p_2(x)) \times p_3(x)) + (x^9 + p_4(x))) + ((x^{18} + p_5(x)) \times p_6(x) + (x^9 + p_7(x))),$$

Algorithm 4 Exponentiation to the 254

Require: x.
Ensure: $y = x^{254}$.

$y \leftarrow x^2$	Type-I
$x \leftarrow x \times x^4$	Type-II
$x \leftarrow x \times x^4$	Type-II
$x \leftarrow x \times x^4$	Type-II
$y \leftarrow y \times x$	Type-III
$y \leftarrow y^2$	Type-I

where the polynomials $p_i(x)$ are of degree at most 9, and can be obtained by successive Euclidean polynomial divisions. Hence, only monomials of degree lower than 9 plus x^{18} and x^{36} are required to calculate any DES S-box. To compute these powers, we found an extended addition chain of extended length 1. It is represented in Figure 2 allows to calculate monomials x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{18} and x^{36}, where it can be checked that only 3 Type-II operations and 1 Type-III operation are needed.

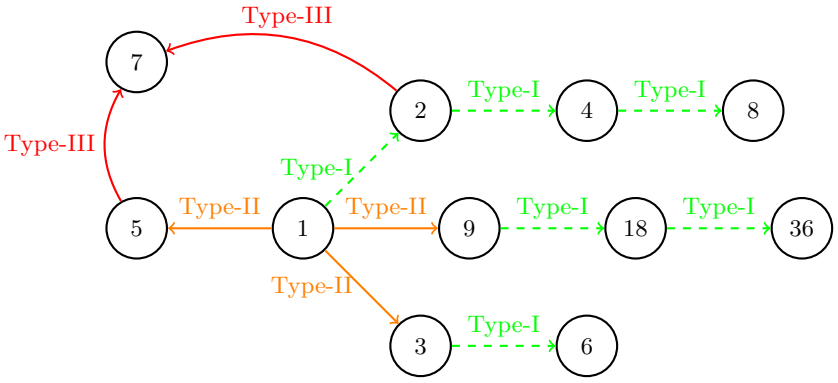

Fig. 2. DES monomials extended addition chain.

Eventually, once monomials are calculated, we can evaluate the different polynomials $p_i(x)$. Then, 3 more operations of Type-III are required to calculate $P_{\mathtt{DES}}$. These 3 multiplications are the 3 polynomial multiplications in $P_{\mathtt{DES}}$. As a result, any of the DES S-boxes can be computed using 4 Type-III operations and 3 of Type-II operations. Full description of computation of DES S-boxes is given in Algorithm 5.

Remark 4. Remark that having an optimal representation of a polynomial that has exponents in different classes (as for the DES S-boxes) is quite challenging, which explains the poorer efficiency compared to the AES S-box. Quite naturally,

Algorithm 5 DES S-boxes

Require: x_1, $p_{i,j}$ coefficients of polynomial p_i.
Ensure: $p_2 = P_{\mathtt{DES}}(x)$.

$x_2 \leftarrow x_1^2$	Type-I
$x_4 \leftarrow x_2^2$	Type-I
$x_8 \leftarrow x_4^2$	Type-I
$x_3 \leftarrow x_1 \times x_1^2$	Type-II
$x_6 \leftarrow x_3^2$	Type-I
$x_9 \leftarrow x_1 \times x_1^8$	Type-II
$x_{18} \leftarrow x_9^2$	Type-I
$x_{36} \leftarrow x_{18}^2$	Type-I
$x_5 \leftarrow x_1 \times x_1^4$	Type-II
$x_7 \leftarrow x_2 \times x_5$	Type-III
$p_1 \leftarrow \sum p_{1,j} x_j$	
$p_2 \leftarrow \sum p_{2,j} x_j$	
$p_3 \leftarrow \sum p_{3,j} x_j$	
$p_4 \leftarrow \sum p_{4,j} x_j$	
$p_5 \leftarrow \sum p_{5,j} x_j$	
$p_6 \leftarrow \sum p_{6,j} x_j$	
$p_7 \leftarrow \sum p_{7,j} x_j$	
$p_2 \leftarrow p_2 + x_{18}$	
$p_2 \leftarrow p_2 \times p_3$	Type-III
$p_2 \leftarrow p_2 + x_9$	
$p_2 \leftarrow p_2 + p_4$	
$p_1 \leftarrow p_1 + x_{36}$	
$p_2 \leftarrow p_2 \times p_1$	Type-III
$p_5 \leftarrow p_5 + x_{18}$	
$p_5 \leftarrow p_5 \times p_6$	Type-III
$p_5 \leftarrow p_5 + x_9$	
$p_5 \leftarrow p_5 + p_7$	
$p_2 \leftarrow p_2 + p_5$	

we expect that further research should allow improving this (since smaller S-boxes should generally be easier to mask).

4 Efficiency Comparisons & Simulations

In this section, we compare, for different orders $d = 1, 2, 3$, the efficiency of our new extended addition chain with that of previous techniques to securely process the `AES` S-box and the first `DES` S-box (similar results can be obtained for the other ones).

For the `AES` S-box, we implemented the schemes proposed in [21] and [9]. We also implemented the scheme in [14], which follows a Tower Fields approach to improve the timing complexities. Essentially, it consists in using the isomorphism between $\mathbb{F}_{2^8}$ and $(\mathbb{F}_{2^4})^2$ to have processing only in $\mathbb{F}_{2^4}$ where the multiplication can be tabulated. Since our approach, described in the previous section, is advantageous when the field multiplication cannot be tabulated, we did not consider to combine it with Tower Fields approach.

The implementations are done in `C` and compile for ATMEGA644p micro-controller thanks to the compiler avr_gcc with optimisation flag -o2. We also did some implementations directly in assembler for the same micro-controller.

For the `AES` the results are given in Table 3.

Table 3. Secure `AES` S-box for ATMEGA644p.

Solution	[`C`] $d = 1$	[`C`] $d = 2$	[`C`] $d = 3$	[Assembly] $d = 1$
Addition chain [21]	753	1999	3702	623
Addition chain + Tower Fields [14]	897	1805	3077	565
Addition chain + Type-II op.[9]	540	1376	2554	431
Extended addition chain	488	1227	2319	338

For the `DES` the results are given in Table 4.

Table 4. Secure `DES` S-box for ATMEGA644p.

Solution	`C` $d = 1$	`C` $d = 2$	`C` $d = 3$
Addition chain [22]	2001	4646	8182
Extended addition chain	1623	3574	7413

In Table 5, we compare, on ATMEGA644p, the practical costs of the Type-II and Type-III multiplications. For $d = 1$, it can be seen that Type-II multiplications are around 2.5 (when implemented in Assembly) or 2.4 (when implemented in `C`) faster than Type-III multiplications. This means that, on ATMEGA644p, replacing N Type-III multiplications by N' Type-II multiplications leads to a

global efficiency gain as long as $N'/N \leq 2.4$. This ratio becomes 2.8 for $d = 2$ and 2.3 for $d = 3$.

Table 5. Costs comparison (in cycles) for Type-II and Type-III operations over $\mathbb{F}_{2^8}$.

Operation	`C` $d = 1$	`C` $d = 2$	`C` $d = 3$	[Assembly] $d = 1$
Type-III	146	430	802	136
Type-II	61	152	344	54

As already pointed out, the interest of exchanging Type-III multiplications by Type-II ones is only advantageous when the field (Type-III) multiplications cannot been tabulated (i.e. when $n \geq 5$). Hence, for the 4-bit `PRESENT` S-box, our approach does not lead to practical efficiency improvement.

Conclusion

By exploiting an idea introduced by Coron *et al* at FSE 2013, we have shown in this paper that Carlet *et al*'s masking scheme can be improved when the S-box dimensions are too large to allow the tabulation of field multiplications. For this purpose, we introduced a new type of addition-chain exponentiation which combine three operations (multiplications by 2^s, multiplications by $1 + 2^s$ and additions) instead of two. For the `AES` and `DES` S-boxes, our improvement leads to an efficiency gain between 35% and 55%. Our work also opens avenues for further research of polynomial evaluation techniques minimizing the number of multiplications which are neither $\mathbb{F}_2$-linear nor $\mathbb{F}_2$-bilinear.

Acknowledgements. Work funded in parts by the European Commission through the ERC project 280141 (acronym CRASH) and the European ISEC action grant HOME/2010/ISEC/AG/INT-011 B-CCENTRE project. F.-X. Standaert is an associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). We also thank Jean-Sébastien Coron for pointing us out the minimization strategy discussed in Remark 3.

References

1. M. Bellare, S. Goldwasser, and D. Micciancio. "pseudo-random" number generation within cryptographic algorithms: the DSS case. In B. Kalisky Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 277–291. Springer, 1997.
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. C. Carlet. Boolean functions for cryptography and error correcting codes. *Boolean Methods and Models*, page 257, 2010.
5. C. Carlet, L. Goubin, E. Prouff, M. Quisquater, and M. Rivain. Higher-order masking schemes for S-Boxes. In A. Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012.
6. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Wiener [25], pages 398–412.
7. J.-S. Coron. Higher Order Masking of Look-up Tables. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, Lecture Notes in Computer Science. Springer, 2014. To appear.
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.-S. Coron, E. Prouff, M. Rivain, and T. Roche. Higher-order side channel security and mask refreshing. In S. Moriai, editor, *Fast Software Encryption – FSE 2013*, Lecture Notes in Computer Science. Springer, 2013. To appear.
10. A. Duc, S. Dziembowski, and S. Faust. Unifying Leakage Models: from Probing Attacks to Noisy Leakage. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, Lecture Notes in Computer Science. Springer, 2014. To appear.
11. D. M. Gordon. A survey of fast exponentiation methods. *J. Algorithms*, 27(1):129–146, 1998.
12. V. Grosso, F.-X. Standaert, and S. Faust. Masking vs. multiparty computation: How large is the gap for AES? In G. Bertoni and J.-S. Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 400–416. Springer, 2013.
13. 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.
14. 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, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2011.
15. D. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, third edition, 1988.
16. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In Wiener [25], pages 388–397.
17. 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.
18. 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.
19. E. Prouff and M. Rivain. Masking against side-channel attacks: A formal security proof. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.
20. 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.

21. M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
22. A. Roy and S. Vivek. Analysis and improvement of the generic higher-order masking scheme of FSE 2012. In G. Bertoni and J.-S. Coron, editors, *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 417–434. Springer, 2013.
23. K. Schramm and C. Paar. Higher order masking of the AES. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.
24. A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.
25. M. Wiener, editor. *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*. Springer, 1999.

A Field Multiplication Algorithms

In the algorithm hereafter, we recall how a multiplication over an extension of $\mathbb{F}_2$ can be done. Since we consider extensions of the form $\mathbb{F}_{2^n} \cong \mathbb{F}_2[X]/p(X)$ where the coefficients of $p(X)$ are in $\mathbb{F}_2$, we denote by $\boldsymbol{p}$ the binary vector whose coordinates are the coefficients of $p(X)$ (from MSB to LSB). The operation $\ll t$ stands for the shift of t bits and the i^{th} bit of a binary vector b is denoted by $b(i)$.

Algorithm 6 Field multiplication naive way

Require: Field elements a, b in $\mathbb{F}_{2^n} \cong \mathbb{F}_2[X]/p(X)$, the binary representation $\boldsymbol{p}$ of $p(X)$.
Ensure: The field element c such that $c = a \times b$

$tmp \leftarrow a$
$c \leftarrow 0$
for i from 0 to $degree(p(X))$ **do**
  if $b(i) = 1$ **then**
    $c \leftarrow c \oplus tmp$
  end if
  $tmp \leftarrow tmp \ll 1$
  if $tmp(degree(p(X))) = 1$ **then**
    $tmp \leftarrow tmp \oplus \boldsymbol{p}$
  end if
end for

For fields of small dimension (e.g. $n \leq 4$), the multiplication can be tabulated. Then, only one access to a double entry table is required to perform the multiplication in an efficient manner. If the field is composed of 2^n elements, the table will have 2^{2n} elements of size n. For larger fields (e.g. $n > 4$) the size of such a table becomes larger than the memory available in embedded system. Hence, other evaluation methods are applied, such that the so-called *log/alog tables* method. It is based on the fact that the non-zero elements of $\mathbb{F}_{2^n}$ can all

be represented as a power of a primitive element which is a root of $p(X)$. The *log* table is used to get this power for each $x \in \mathbb{F}_2[X]/p(X)$, whereas the *alog* table is used to get the element of $\mathbb{F}_2[X]/p(X)$ that corresponds to a given power. Under this representation, multiplying two non-zero elements x and y, simply consists in processing $alog(log(x) + log(y) \bmod 2^n - 1)$.

Algorithm 7 Field multiplication with log/alog tables

Require: Field elements a, b.
Ensure: c such that $c = a \times b$
1: $d \leftarrow log[a]$
2: $e \leftarrow log[b]$
3: $c \leftarrow d + e \bmod 2^n - 1$
4: $c \leftarrow alog[c]$

B Masking Complexity of Power Functions

For exponentiation in $\mathbb{F}_{2^4}$, we report on the cost of our extended addition chain in Table 6.

Table 6. Smallest cost to process x^α with operations of types II and III in $\mathbb{F}_{2^4}$.

# Type-II	# Type-III	Exponent α
0	0	$C_0 = \{0\}$,$C_1 = \{1, 2, 4, 8\}$
1	0	$C_3 = \{3, 6, 12, 9\}$, $C_5 = \{5, 10\}$
1	1	$C_7 = \{7, 14, 13, 11\}$

For the case of operations in $\mathbb{F}_{2^6}$, like for the `DES` S-boxes. We report on the cost of our extended addition chain in Table 7.

Table 7. Smallest cost to process x^α with operations of types II and III in $\mathbb{F}_{2^6}$.

# Type-II	# Type-III	Exponent α
0	0	$C_0 = \{0\}$,$C_1 = \{1, 2, 4, 8, 16, 32\}$
1	0	$C_3 = \{3, 6, 12, 24, 48, 33\}$, $C_5 = \{5, 10, 20, 40, 17, 34\}$, $C_9 = \{9, 18, 36\}$
2	0	$C_{11} = \{11, 22, 44, 25, 50, 37\}$,$C_{15} = \{15, 30, 60, 57, 51, 39\}$ $C_{27} = \{27, 54, 45\}$
1	1	$C_7 = \{7, 14, 28, 56, 49, 35\}$ $C_{13} = \{13, 26, 52, 41, 19, 38\}$,$C_{21} = \{21, 42\}$ $C_{31} = \{31, 62, 61, 59, 55, 47, \}$
2	1	$C_{23} = \{23, 46, 29, 58, 53, 43\}$