On the Multiplicative Complexity of Boolean Functions and Bitsliced Higher-Order Masking

Dahmun Goudarzi[1,2] and Matthieu Rivain[1]

[1] CryptoExperts, Paris, France
[2] ENS, CNRS, INRIA and PSL Research University, Paris, France

dahmun.goudarzi@cryptoexperts.com
matthieu.rivain@cryptoexperts.com

Abstract. Higher-order masking is a widely used countermeasure to make software implementations of blockciphers achieve high security levels against side-channel attacks. Unfortunately, it often comes with a strong impact in terms of performances which may be prohibitive in some contexts. This situation has motivated the research for efficient schemes that apply higher-order masking with minimal performance overheads. The most widely used approach is based on a polynomial representation of the cipher s-box(es) allowing the application of standard higher-order masking building blocks such as the ISW scheme (Ishai-Sahai-Wagner, Crypto 2003). Recently, an alternative approach has been considered which is based on a bitslicing of the s-boxes. This approach has been shown to enjoy important efficiency benefits, but it has only been applied to specific blockciphers such as AES, PRESENT, or custom designs. In this paper, we present a generic method to find a Boolean representation of an s-box with efficient bitsliced higher-order masking. Specifically, we propose a method to construct a circuit with low *multiplicative complexity*. Compared to previous work on this subject, our method can be applied to any s-box of common size and not necessarily to small s-boxes. We use it to derive higher-order masked s-box implementations that achieve important performance gain compared to optimized state-of-the-art implementations.

1 Introduction

One of the most widely used strategy to protect software implementations of blockciphers against side-channel attacks consists in applying *secret sharing* at the implementation level. This strategy also known as (*higher-order*) *masking* notably achieves provable security in the *probing security model* [ISW03] and in the *noisy leakage model* [PR13,DDF14]. While designing a higher-order masking scheme for a given blockcipher, the main issue is the secure and efficient computation of the s-box. Most of the proposed solutions (see for instance [RP10,CRV14,CPRR15]) are based on a polynomial representation of the s-box over the finite field $\mathbb{F}_{2^n}$ (where n is the input bit-length), for which the field multiplications are secured using the *ISW scheme* due to Ishai, Sahai and Wagner [ISW03].

An alternative approach has recently been put forward which consists in applying higher-order masking at the Boolean level by bitslicing the s-boxes within a cipher round [GLSV15,GR16]. In the bitsliced higher-order masking paradigm, the ISW scheme is applied to secure bitwise AND instructions which are significantly more efficient than their field-multiplication counterparts involved in polynomial schemes. Moreover, such a strategy allows to compute all the s-boxes within a cipher round at the same time, which results in important efficiency gains. To the best of our knowledge, bitsliced higher-order masking has only been applied to specific blockciphers up to now. In [GLSV15], Grosso *et al.* introduce new blockciphers with LS-designs tailored to efficient masked computation in bitslice. The approach has also been used by Goudarzi and Rivain in [GR16] to get fast implementations of two prominent blockciphers, namely AES and PRESENT, masked at an order up to 10. However, no generic method to apply this approach to *any* blockcipher has been proposed so far. In contrast several generic methods have been published for the polynomial setting [CGP$^+$12,CRV14,CPRR15]. Therefore, and given the efficiency benefits of bitsliced higher-order masking approach, defining such a generic method is an appealing open issue.

Finding a Boolean representation of an s-box that yields an efficient computation in the bitsliced masking world merely consists in finding a circuit with low *multiplicative complexity.* The multiplicative complexity of Boolean functions has been studied in a few previous papers [MS92,BPP00,TP14]. In particular, optimal circuits have been obtained for some small (3-bit/4-bit/5-bit) s-boxes using SAT solvers [CMH13,Sto16]. However no general (heuristic) method has been proposed up to now to get an efficient decomposition for any s-box, and in particular for n-bit s-boxes with $n \geq 6$.

In this paper, we introduce a new heuristic method to decompose an s-box into a circuit with low multiplicative complexity. Our proposed method follows the same approach as the CRV and *algebraic decomposition* methods used to get efficient representations in the polynomial setting [CRV14,CPRR15]. We also introduce the notion of *parallel multiplicative complexity* to capture the fact that several AND gates might be bundled in a single instruction, enabling further gain in the bitslice setting [GR16]. Eventually, we describe ARM implementations of bitsliced higher-order-masked s-box layers using our decomposition method and we compare them to optimized versions of the CRV and algebraic decomposition methods. Our results show a clear superiority of the bitslice approach when the masking order exceeds a certain threshold.

The paper is organized as follows. Section 2 gives some preliminaries about Boolean functions and higher-order masking. We then introduce the notion of (parallel) multiplicative complexity and discuss previous results as well as our contribution in Section 3. Section 4 presents our heuristic method in a general setting as well as some s-box-specific improvements. Finally, Section 5 describes our implementations and the obtained performances.

2 Preliminaries

2.1 Boolean Functions

Let $\mathbb{F}_2$ denote the field with 2 elements and let n be a positive integer. A *Boolean function* f with n variables is a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The set of such functions is denoted $\mathcal{F}_n$ in this paper. Any Boolean function $f \in \mathcal{F}_n$ can be seen as a multivariate polynomial over $\mathbb{F}_2[x_1, x_2, \ldots, x_n]/(x_1^2 - x_1, x_2^2 - x_2, \ldots, x_n^2 - x_n)$:

$$f(x) = \sum_{u \in \{0,1\}^n} a_u \, x^u \ , \tag{1}$$

where $x = (x_1, x_2, \ldots, x_n)$, $x^u = x_1^{u_1} \cdot x_2^{u_2} \cdot \ldots \cdot x_n^{u_n}$, and $a_u \in \mathbb{F}_2$ for every $u \in \{0,1\}^n$. The above representation is called the *Algebraic Normal Form* (ANF).

For any family $f_1, f_2 \ldots, f_m \in \mathcal{F}_n$, the set $\langle f_1, f_2 \ldots, f_m \rangle = \big\{ \sum_{i=0}^m a_i f_i \mid a_i \in \mathbb{F}_2 \big\}$ is called the *span* of the f_i's (or the space *spanned* by the f_i's), which is a $\mathbb{F}_2$-vector space. Let $\mathcal{M}_n$ denote the set of *monomial functions* that is $\mathcal{M}_n = \big\{ x \mapsto x^u \mid u \in \{0,1\}^n \big\}$. Then, the set of Boolean functions with n variables can be defined as the span of monomial functions, that is $\mathcal{F}_n = \langle \mathcal{M}_n \rangle$.

Let n and m be two positive integers, and let S be a function mapping $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$. Such a function can be seen as a vector of Boolean functions, *i.e.* $S(x) = (f_1(x), f_2(x) \ldots, f_m(x))$ and is hence called a *vectorial (Boolean) function* also known as an $(n \times m)$ s-box in cryptography. The Boolean functions $f_1, f_2, \ldots, f_m \in \mathcal{F}_n$ are then called the coordinate functions of S.

2.2 Higher-Order Masking

Higher-order masking consists in sharing each internal variable x of a cryptographic computation into d random variables x_1, x_2, …, x_d, called *the shares* and satisfying $x_1 + x_2 + \cdots + x_d = x$, for some group operation $+$, such that any set of $d-1$ shares is randomly distributed and independent of x. In this paper, the considered masking operation will be the bitwise addition. It has been formally demonstrated that in the noisy leakage model, where the attacker gets noisy information on each share, the complexity of recovering information on x grows exponentially with the number of shares [CJRR99,PR13]. This number d, called *the masking order*, is hence a sound security parameter for the resistance of a masked implementation.

The main issue while protecting a blockcipher implementation with masking is the secure computation of the nonlinear layer applying the s-boxes to the cipher state. The prevailing approach consists in working on the polynomial representation of the s-box over the field $\mathbb{F}_{2^n}$, which is secured using the ISW scheme [ISW03] for the field multiplications [RP10,CGP$^+$12]. The most efficient polynomial evaluation method in this paradigm is due to Coron, Roy and Vivek [CRV14]. The polynomial representation can also be decomposed in functions of lower algebraic degree as recently proposed by Carlet *et al.* in [CPRR15]. In the quadratic case, these functions can then be efficiently secured using the CPRR scheme [CPRR14].

2.3 Bitsliced Higher-Order Masking

A variant of polynomial methods is to apply masking at the Boolean level using bitslicing (see for instance [DPV01,GLSV15,BGRV15]). In [GR16], the authors apply this approach to get highly efficient implementations of AES and PRESENT with masking order up to 10. In their implementations, bitslice is applied at the s-box level. Specifically, based on a Boolean circuit for an s-box S, one can perform ℓ parallel evaluations of S in software by replacing each gate of the circuit with the corresponding bitwise instruction, where ℓ is the bit-size of the underlying CPU architecture. It results that the only nonlinear operations in the parallel s-box processing are bitwise AND instructions between ℓ-bit registers which can be efficiently secured using the ISW scheme. Such an approach achieves important speedup compared to polynomial methods since (i) ISW-based ANDs are substantially faster than ISW-based field multiplications in practice, (ii) all the s-boxes within a cipher round are computed in parallel. The authors of [GR16] propose an additional optimization. In their context, the target architecture (ARM) is of size $\ell = 32$ bits, whereas the number of s-boxes per round is 16 (yielding 16-bit bitslice registers). Therefore, they suggest to group the ANDs by pair in order to perform a single ISW-based 32-bit AND where the standard method would have performed two ISW-based 16-bit AND. This roughly decreases the complexity by a factor up to two.[3]

3 Multiplicative Complexity of Boolean Functions

We shall call Boolean circuit any computation graph composed of $\mathbb{F}_2$-multiplication nodes (AND gates), $\mathbb{F}_2$-addition nodes (XOR gates), and switching nodes (NOT gates). Informally speaking, the multiplicative complexity of a Boolean function is the minimum number of $\mathbb{F}_2$-multiplication gates required by a Boolean circuit to compute it. This notion can be formalized as follows:

Definition 1. *The* multiplicative complexity $C(f_1, f_2, \ldots, f_m)$ *of a family of Boolean functions* $f_1, f_2, \ldots, f_m \in \mathcal{F}_n$*, is the minimal integer* t *for which there exist Boolean functions* $g_i, h_i \in \mathcal{F}_n$ *for* $i \in [\![1, t]\!]$ *such that:*

$$\begin{cases} g_1, h_1 \in \langle 1, x_1, x_2, \ldots, x_n \rangle \\ \forall i \in [\![2, t]\!] : g_i, h_i \in \langle 1, x_1, x_2, \ldots, x_n, g_1 \cdot h_1, \ldots, g_{i-1} \cdot h_{i-1} \rangle \end{cases} \tag{2}$$

and

$$f_1, f_2, \ldots, f_m \in \langle 1, x_1, x_2, \ldots, x_n, g_1 \cdot h_1, \ldots, g_t \cdot h_t \rangle \ . \tag{3}$$

It is easy to see that any set of Boolean functions $\{f_1, f_2, \ldots, f_m\} \subseteq \mathcal{F}_n$ has multiplicative complexity satisfying

$$C(f_1, f_2, \ldots, f_m) \le C(\mathcal{M}_n) = 2^n - (n + 1) \ . \tag{4}$$

[3] Packing the operands and depacking the result implies a linear overhead in the number of shares, whereas the number of quadratic operations (the ISW-ANDs) are divided by a factor up to 2.

Moreover, a counting argument shows that there exists $f \in \mathcal{F}_n$ such that

$$C(f) > 2^{\frac{n}{2}} - n \ . \tag{5}$$

In [BPP00], Boyar *et al.* provide a constructive upper bound for any Boolean function:

Theorem 1 ([BPP00]). *For every $f \in \mathcal{F}_n$, we have*

$$C(f) \leq \begin{cases} 2^{\frac{n}{2}+1} - \frac{n}{2} - 2 & \text{if } n \text{ is even,} \\ 3 \cdot 2^{\frac{n-1}{2}} - \frac{n-1}{2} - 2 & \text{otherwise.} \end{cases} \tag{6}$$

The particular case of Boolean functions with 4 and 5 variables has been investigated by Turan and Peralta in [TP14]. They give a complete characterization of affine-equivalence classes of these functions and they show that every $f \in \mathcal{F}_4$ has $C(f) \leq 3$ and every $f \in \mathcal{F}_5$ has $C(f) \leq 4$.

Other works have focused on the multiplicative complexity of particular kinds of Boolean functions. In [MS92], Mirwald and Schnorr deeply investigate the case of functions with quadratic ANF. In particular they show that such functions have multiplicative complexity at most $\lfloor \frac{n}{2} \rfloor$. Boyar *et al.* give further upper bounds for symmetric Boolean functions in [BPP00].

3.1 Multiplicative Complexity of S-Boxes

The multiplicative complexity of an s-box $S : x \mapsto (f_1(x), f_2(x) \dots, f_m(x))$ is naturally defined as the multiplicative complexity of the family of its coordinate functions. We shall also call multiplicative complexity of a given circuit the actual number of multiplication gates involved in the circuit, so that the multiplicative complexity of a circuit gives an upper bound of the multiplicative complexity of the underlying s-box.

The best known circuit for the AES s-box in terms of multiplicative complexity is due to Boyar *et al.* [BMP13]. This circuit achieves a multiplicative complexity of 32 which was obtained by applying logic minimization techniques to the compact representation of the AES s-box due to Canright [Can05] (and saving 2 multiplications compared to the original circuit).

In [CMH13], Courtois *et al.* use SAT-solving to find the multiplicative complexity of small s-boxes. Their approach consists in writing the Boolean system obtained for a given s-box and a given (target) multiplicative complexity t as a SAT-CNF problem, where the unknowns of the system are the coefficients of the g_i and h_i in Definition 1. For each value of t, the solver either returns a solution or a proof that no solution exists, so that the multiplicative complexity is the first value of t for which a solution is returned. They apply this approach to find Boolean circuits with the smallest multiplicative complexity for a random 3×3 s-box (meant to be used in CTC2 [Cou07]), the 4×4 s-box of PRESENT, and for several sets of 4×4 s-boxes proposed for GOST [PLW10]. These results have recently been extended by Stoffelen who applied the Courtois *et al.* approach to find optimal circuits for various 4×4 and 5×5 s-boxes [Sto16].

The main limitation of the SAT-solving approach is that it is only applicable to small s-boxes due to the combinatorial explosion of the underlying SAT problem, and getting the decomposition of an s-box of size *e.g.* $n = 8$ seems out of reach. Moreover, the method is not generic in the sense that the obtained decomposition stands for a single s-box and does not provide an upper bound for the multiplicative complexity of s-boxes of a given size.

3.2 Our Results

We give new constructive upper bounds for the multiplicative complexity of s-boxes. As a first result, we extend Theorem 1 to s-boxes (see proof in the full version):

Theorem 2. *For every $S \in \mathcal{F}_n^m$, we have:*

$$C(S) \leq \min_{k \in [\![1,n]\!]} (m2^k + 2^{n-k} + k) - (m + n + 1) \ . \tag{7}$$

When $m = n$, the min is achieved by $k = \left\lfloor \frac{n - \log_2 n}{2} \right\rceil$ for most $n \in \mathbb{N}$, which gives $C(S) \leq B_n$ with

$$B_n \approx \sqrt{n}\, 2^{\frac{n}{2}+1} - \Big(\frac{3n + \log_2 n}{2} + 1 \Big) \ . \tag{8}$$

We further introduce in this paper a heuristic decomposition method achieving lower multiplicative complexity. Our general result is summarized in the following Theorem:

Theorem 3. *For every $S \in \mathcal{F}_n^m$, we have $C(S) \leq C_{n,m}$ with*

$$C_{n,m} \approx \sqrt{m}\, 2^{\frac{n}{2}+1} - m - n - 1 \ . \tag{9}$$

And in particular

$$C_{n,n} = \begin{cases} 17 & \textit{for } n = 5 \\ 31 & \textit{for } n = 6 \\ 50 & \textit{for } n = 7 \end{cases} \quad \textit{and} \quad C_{n,n} = \begin{cases} 77 & \textit{for } n = 8 \\ 122 & \textit{for } n = 9 \\ 190 & \textit{for } n = 10 \end{cases} \tag{10}$$

In the above theorem, $C_{n,m}$ denote the multiplicative complexity of the generic method presented in Section 4. We also propose non-generic improvements of this method that might give different results depending on the s-box. Table 1 summarizes the multiplicative complexities obtained by the two above theorems and the non-generic improved method for $n \times n$ s-boxes with $n \in [\![4, 10]\!]$. For the latter, the figures represent what we hope to achieve for a random s-box (that we were able to achieve for some tested s-boxes).

Table 1. Multiplicative complexities of $n \times n$ s-boxes.

n	4	5	6	7	8	9	10
Theorem 2	8	16	29	47	87	120	190
Our generic method ($C_{n,n}$)	8	17	31	50	77	122	190
Our improved method ($C^*_{n,n}$)	7	13	23	38	61	96	145

3.3 Parallel Multiplicative Complexity

We introduce hereafter the notion of *parallel multiplicative complexity* for Boolean functions and s-boxes. We consider circuits with multiplication gates that can process up to k multiplications in parallel. The k-parallel multiplicative complexity of an s-box is the least number of k-parallel multiplication gates required by a circuit to compute it. We formalize this notion hereafter:

Definition 2. *The k-*parallel multiplicative complexity *$C^{(k)}(f_1, f_2, \ldots, f_m)$ of a family of Boolean functions $f_1, f_2, \ldots, f_m \in \mathcal{F}_n$, is the minimal integer t for which there exist Boolean functions $g_i, h_i \in \mathcal{F}_n$ for $i \in [\![1, tk]\!]$ such that:*

$$\begin{cases} g_1, h_1, g_2, h_2, \ldots, g_k, h_k \in \langle 1, x_1, x_2, \ldots, x_n \rangle \ , \\ \forall i \in [\![1, t-1]\!] : g_{ik+1}, h_{ik+1}, \ldots, g_{(i+1)k}, h_{(i+1)k} \\ \qquad\qquad\qquad\qquad \in \langle 1, x_1, x_2, \ldots, x_n, g_1 \cdot h_1, \ldots, g_{ik} \cdot h_{ik} \rangle \end{cases} \tag{11}$$

and

$$f_1, f_2, \ldots, f_m \in \langle 1, x_1, x_2, \ldots, x_n, g_1 \cdot h_1, \ldots, g_{tk} \cdot h_{tk} \rangle \ . \tag{12}$$

The main motivation for introducing this notion comes from the following scenario. Assume we want to perform m s-box computations in bitslice on an ℓ-bit architecture, where $\ell > m$. Then we have to pick a circuit computing the s-box and translate it in software by replacing Boolean gates with corresponding bitwise instructions. Since each bitsliced register contains m bits (m versions of the same input or intermediate bit), one can perform up to $k = \lceil \ell/m \rceil$ multiplication gates with a single ℓ-bit AND instruction (modulo some packing of the operands and unpacking of the results). If the used circuit has a k-parallel multiplicative complexity of t, then the resulting bitsliced implementation involve t bitwise AND instructions. This number of AND instructions is the main efficiency criterion when such an implementation is protected with higher-order masking which makes the k-parallel multiplicative complexity an important parameter for an s-box in this context.

The authors of [GR16] show that the AES circuit of Boyar *et al.* can be fully parallelized at degree 2, *i.e.* its 2-parallel multiplicative complexity is 16. In the full version of this paper, we further show that a sorted version of this circuit can achieve k-parallel multiplicative complexity of 9, 7 and 6 for $k = 4$, $k = 8$, and $k = 16$ respectively.

The decomposition method introduced in this paper has the advantage of being highly parallelizable. Table 2 summarizes the obtained k-parallel multiplicative complexity $C_{n,n}^{(k)}$ for $n \times n$ s-boxes for $k \in \{2, 4\}$. Note that we always have $C_{n,n}^{(k)} \in \{\lceil \frac{C_{n,n}}{k} \rceil, \lceil \frac{C_{n,n}}{k} \rceil + 1\}$ which is almost optimal.

Table 2. Parallel multiplicative complexities of our method for $n \times n$ s-boxes.

n	4	5	6	7	8	9	10
$C_{n,n}$	8	17	31	50	77	122	190
$C_{n,n}^{(2)}$	4	9	16	25	39	62	95
$C_{n,n}^{(4)}$	2	5	9	13	20	31	48

4 A Heuristic Decomposition for S-boxes

In this section, we introduce a heuristic decomposition method for s-boxes that aim to minimize the number of $\mathbb{F}_2$ multiplications. The proposed method follows the same approach than the CRV decomposition over $\mathbb{F}_{2^n}[x]$. We first describe the proposed heuristic for a single Boolean function before addressing the case of s-boxes.

4.1 Decomposition of a Single Boolean Function

Let f be a Boolean function. The proposed decomposition simply consists in writing f as:

$$f(x) = \sum_{i=0}^{t-1} g_i(x) \cdot h_i(x) + h_t(x) \tag{13}$$

where $g_i, h_i \in \langle \mathcal{B} \rangle$, for some basis of functions $\mathcal{B} = \{\phi_j\}_{j=1}^{|B|}$. Assume that all the $\phi_j(x)$, $\phi_j \in \mathcal{B}$, can be computed with r multiplications. Then the total multiplicative complexity of the above decomposition is of $r + t$. We now explain how to find such a decomposition by solving a linear system.

Solving a linear system. As in the CRV method, we first sample t random functions g_i from $\langle \mathcal{B} \rangle$. This is simply done by picking $t \cdot |\mathcal{B}|$ random bits $a_{i,j}$ and setting $g_i = \sum_{\phi_j \in \mathcal{B}} a_{i,j} \phi_j$. Then we search for a family of $t + 1$ Boolean functions $\{h_i\}_i$ satisfying (13). This is done by solving the following system of linear equations over $\mathbb{F}_2$:

$$A \cdot c = b \tag{14}$$

where $b = (f(e_1), f(e_2), \ldots, f(e_n))^\mathsf{T}$ with $\{e_i\} = \mathbb{F}_2^n$ and where A is a matrix defined as the concatenation of $t + 1$ submatrices:

$$A = (A_0 | A_1 | \ \cdots \ | \ A_t) \tag{15}$$

with

$$A_i = \begin{pmatrix} \phi_1(e_1)\cdot g_i(e_1) & \phi_2(e_1)\cdot g_i(e_1) & \dots & \phi_{|\mathcal{B}|}(e_1)\cdot g_i(e_1) \\ \phi_1(e_2)\cdot g_i(e_2) & \phi_2(e_2)\cdot g_i(e_2) & \dots & \phi_{|\mathcal{B}|}(e_2)\cdot g_i(e_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(e_{2^n})\cdot g_i(e_{2^n}) & \phi_2(e_{2^n})\cdot g_i(e_{2^n}) & \dots & \phi_{|\mathcal{B}|}(e_{2^n})\cdot g_i(e_{2^n}) \end{pmatrix} \tag{16}$$

for $0 \leq i \leq t-1$, and

$$A_t = \begin{pmatrix} \phi_1(e_1) & \phi_2(e_1) & \dots & \phi_{|\mathcal{B}|}(e_1) \\ \phi_1(e_2) & \phi_2(e_2) & \dots & \phi_{|\mathcal{B}|}(e_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(e_{2^n}) & \phi_2(e_{2^n}) & \dots & \phi_{|\mathcal{B}|}(e_{2^n}) \end{pmatrix} \tag{17}$$

It can be checked that the vector c, solution of the system, gives the coefficients of the h_i's over the basis $\mathcal{B}$. A necessary condition for this system to have a solution whatever the target vector b (*i.e.* whatever the Boolean function f) is to get a matrix A of full rank. In particular, the following inequality must hold:

$$(t+1)|\mathcal{B}| \geq 2^n \ . \tag{18}$$

Another necessary condition to get a full-rank matrix is that the *squared basis* $\mathcal{B} \times \mathcal{B} = \{\phi_i \cdot \phi_k \mid \phi_i, \phi_k \in \mathcal{B}\}$ spans the entire space $\mathcal{F}_n$. A classic basis of the vector space is the set of monomials $\mathcal{M}_n$. Therefore, we suggest to take a basis $\mathcal{B}$ such that $\mathcal{M}_n \subseteq \mathcal{B} \times \mathcal{B}$. Let

$$\mathcal{B}_0 = \{x \mapsto x^u, u \in \mathcal{U}\} \\ \text{with } \ \mathcal{U} = \big\{(u_1, \dots, u_\ell, 0, \dots, 0)\big\} \cup \big\{(0, \dots, 0, u_{\ell+1}, \dots, u_n)\big\} \tag{19}$$

where $\ell = \lceil \frac{n}{2} \rceil$ and where $u_i \in \{0,1\}$ for every $i \in [\![1, n]\!]$. Then, we clearly have $\mathcal{B}_0 \times \mathcal{B}_0 = \mathcal{M}_n$. We hence suggest taking $\mathcal{B} \supseteq \mathcal{B}_0$, with $\mathcal{B}$ possibly larger than $\mathcal{B}_0$ since restraining ourselves to $\mathcal{B} = \mathcal{B}_0$ could be non-optimal in terms of multiplications for the underlying decomposition method. Indeed, (18) shows that the more elements in the basis, the smaller t, *i.e.* the less multiplications $g_i \cdot h_i$. We might therefore derive a bigger basis by iterating $\mathcal{B} \leftarrow \mathcal{B} \cup \{\phi_j \cdot \phi_k\}$, where ϕ_j and ϕ_k are randomly sampled from $\mathcal{B}$ until reaching a basis $\mathcal{B}$ with the desired cardinality.

We then have $r = |\mathcal{B}| - n - 1$, where we recall that r denotes the number of multiplications to derive $\mathcal{B}$, and since $x \mapsto 1, x \mapsto x_1, \dots, x \mapsto x_n \in \mathcal{B}$ requires no multiplications. By construction, we have $|\mathcal{B}| \geq |\mathcal{B}_0| = 2^\ell + 2^{n-\ell} - 1$, implying $r \geq 2^\ell + 2^{n-\ell} - (n+2)$. Let $C_n = r + t$ denote the number of multiplications achieved by our decomposition method. Then, by injecting C_n in (18) we get:

$$(C_n - r + 1)(n + 1 + r) \geq 2^n \ , \tag{20}$$

that is:

$$C_n \geq r + \frac{2^n}{n+1+r} - 1 \ . \tag{21}$$

It can be checked that the value of r minimizing the above bound is $2^{\frac{n}{2}} - (n-1)$. However, r must satisfy $r \geq 2^{\ell} + 2^{n-\ell} - (n+2)$ where $\ell = \lceil \frac{n}{2} \rceil$, which is always greater than $2^{\frac{n}{2}} - (n-1)$ for $n \geq 2$. That is why we shall define the optimal value of the parameter r (for the single-Boolean-function case) as:

$$r_{opt} = 2^{\ell} + 2^{n-\ell} - (n+2) = \begin{cases} 2^{\frac{n}{2}+1} - (n+2) & \text{if } n \text{ even,} \\ 3 \cdot 2^{\frac{n-1}{2}} - (n+2) & \text{if } n \text{ odd,} \end{cases} \tag{22}$$

which amounts to taking $\mathcal{B} = \mathcal{B}_0$. The corresponding optimal value for t is then defined as:

$$t_{opt} = \left\lceil \frac{2^n}{r_{opt} + n + 1} \right\rceil - 1 \tag{23}$$

which gives $t_{opt} \approx 2^{\frac{n}{2}-1}$ for n even, and $t_{opt} \approx \frac{1}{3}\, 2^{\frac{n+1}{2}}$ for n odd.

Table 3. Optimal and achievable parameters for a single Boolean function.

n	4	5	6	7	8	9	10
Optimal parameters							
(r,t)	(2,2)	(5,2)	(8,4)	(15,5)	(22,8)	(37,10)	(52,16)
$\lvert\mathcal{B}\rvert$	7	11	15	23	31	47	63
C_n	4	7	12	20	30	46	68
Achievable parameters							
(r,t)	(2,3)	(5,3)	(9,5)	(16,6)	(25,9)	(41,11)	(59,17)
$\lvert\mathcal{B}\rvert$	7	11	16	24	34	51	70
C_n	5	8	14	22	34	52	78

In Table 3, we give the optimal values for (r,t) as well as the corresponding size of the basis $\mathcal{B}$ and multiplication complexity C_n for $n \in [\![4, 10]\!]$. We also give the parameter values that we could actually achieve in practice to get a full-rank system. We observe a small gap between the optimal and the achievable parameters, which results from the heuristic nature of the method (since we cannot prove that the constructed matrix A is full-rank).

4.2 S-box decomposition

Let $S \colon x \mapsto (f_1(x), f_2(x), \ldots, f_m(x))$ be an s-box. We can apply the above heuristic to each of the m coordinate functions f_i to get a decomposition as follows:

$$f_i(x) = \sum_{j=0}^{t-1} g_j(x) \cdot h_{i,j}(x) + h_{i,t}(x), \tag{24}$$

for $1 \leq i \leq m$. Here the g_j's are randomly sampled from $\langle \mathcal{B} \rangle$ until obtaining a full-rank system, which is then used to decompose every coordinate function f_i. The total number of multiplications is $C_{n,m} = r + m \cdot t$. Then, (18) gives:

$$C_{n,m} \geq r + m \left(\frac{2^n}{n+1+r} - 1 \right) . \tag{25}$$

It can be checked that the value of r minimizing the above bound is $\sqrt{m2^n} - n - 1$. We hence define

$$r_{opt} = \left\lfloor \sqrt{m2^n} \right\rceil - n - 1 \ , \tag{26}$$

which minimizes (25) for every $n \in [\![2, 10]\!]$ and every $m \in [\![1, n]\!]$. Moreover, this value satisfies the constraint (18) *i.e.* $r_{opt} \geq 2^\ell + 2^{n-\ell} - (n+2)$ for every $m \geq 4$, and in practice we shall only consider s-boxes with $m \geq 4$. The corresponding optimal value t_{opt} is then defined w.r.t. r_{opt} as in (23), which satisfies

$$t_{opt} = \left\lceil \frac{2^{\frac{n}{2}}}{\sqrt{m}} \right\rceil - 1 \tag{27}$$

for every $n \in [\![2, 10]\!]$ and every $m \in [\![1, n]\!]$. We hence get

$$C_{n,m} \geq r_{opt} + m \cdot t_{opt} \approx \sqrt{m}\, 2^{\frac{n}{2}+1} - (n + m + 1) \ . \tag{28}$$

In Table 4, we give the optimal values for the parameters (r, t) as well as the corresponding size of the basis $\mathcal{B}$ and multiplication complexity $C_{n,n}$ for $n \times n$ s-boxes with $n \in [\![4, 10]\!]$. We also give the parameter values that we could actually achieve in practice to get a full-rank system.

Table 4. Optimal and achievable parameters for an $n \times n$ s-box.

n	4	5	6	7	8	9	10
Optimal parameters							
(r, t)	(3,1)	(7,2)	(13,3)	(22,4)	(36,5)	(58,7)	(90,10)
$\lvert\mathcal{B}\rvert$	8	13	20	30	45	68	101
$C_{n,n}$	7	17	31	50	76	121	190
Achievable parameters							
(r, t)	(4,1)	(7,2)	(13,3)	(22,4)	(37,5)	(59,7)	(90,10)
$\lvert\mathcal{B}\rvert$	9	13	20	30	46	69	101
$C_{n,n}$	8	17	31	50	77	122	190

In comparison to the single-Boolean-function case, the optimal size of the basis $\mathcal{B}$ for the s-box decomposition is significantly bigger. This comes from the fact that a bigger basis implies a lower t for each of the m coordinate functions (*i.e.* decrementing t implies decreasing $C_{n,m}$ by m). We also observe a very close gap (sometimes null) between the optimal and the achievable parameters. This tightness, compared to the single-Boolean-function case, is most likely due to the fact that we use a bigger basis.

4.3 Improvements

We present hereafter some improvements of the above method which can be applied to get a decomposition with better multiplicative complexity for a given s-box. In comparison to the above results, the obtained system and the associated multiplicative complexity depend on the target s-box and are not applicable to all s-boxes.

Basis update. Our first improvement of the above method is based on a dynamic update of the basis, each time a coordinate function $f_i(x)$ is computed.[4] Indeed, the term $g_j(x) \cdot h_{i,j}(x)$ involved in the computation of $f_i(x)$ can be reused in the computation of the following $f_{i+1}(x)$, ..., $f_n(x)$. In our decomposition process, this means that the $g_j \cdot h_{i,j}$ functions can be added to the basis for the decomposition of the next coordinate functions f_{i+1}, ..., f_n. Basically, we start with some basis $\mathcal{B}_1 \supseteq \mathcal{B}_0$, where $\mathcal{B}_0$ is the minimal basis as defined in (19). Then, for every $i \geq 1$, we look for a decomposition

$$f_i(x) = \sum_{j=0}^{t_i - 1} g_{i,j}(x) \cdot h_{i,j}(x) + h_{i,t_i}(x), \tag{29}$$

where $t_i \in \mathbb{N}$ and $g_{i,j}, h_{i,j} \in \langle \mathcal{B}_i \rangle$. Once such a decomposition has been found, we carry on with the new basis $\mathcal{B}_{i+1}$ defined as:

$$\mathcal{B}_{i+1} = \mathcal{B}_i \cup \{g_{i,j} \cdot h_{i,j}\}_{j=0}^{t_i-1}. \tag{30}$$

Compared to the former approach, we use different functions $g_{i,j}$ and we get a different matrix A for every coordinate function f_i. On the other hand, for each decomposition, the basis grows and hence the number t_i of multiplicative terms in the decomposition of f_i might decrease. In this context, we obtain a new condition for every i that is:

$$t_i \geq \frac{2^n}{|\mathcal{B}_i|} - 1 \ . \tag{31}$$

The lower bound on t_i hence decreases as $\mathcal{B}_i$ grows. The total multiplicative complexity of the method is then of:

$$C^*_{n,m} = r + \sum_{i=1}^{m} t_i \ , \tag{32}$$

where $r = |\mathcal{B}_1| - (n+1)$ is the number of multiplications required to derive the initial basis $\mathcal{B}_1$. From the above inequality, we can define the optimal sequence of t_i and $s_i = |\mathcal{B}_i|$ as:

$$t_1 = \psi_n(s_1) \quad \text{and} \quad \begin{cases} s_{i+1} = s_i + t_i \\ t_{i+1} = \psi_n(s_{i+1}) \end{cases} \quad \text{for every } i > 1 \tag{33}$$

[4] A similar idea is used in [BMP13] to construct an efficient circuit for the inversion in $\mathbb{F}_{16}$.

where $\psi_n \colon x \mapsto \left\lceil \frac{2^n}{x} \right\rceil - 1$. The sequence (s_i, t_i) is fully determined by the cardinality of the original basis $s_1 = |\mathcal{B}_1|$, and we have:

$$\begin{cases} s_i = (\psi_n + \mathrm{Id})^{(i-1)}(s_1) \\ t_i = \psi_n \circ (\psi_n + \mathrm{Id})^{(i-1)}(s_1) \end{cases} \tag{34}$$

for every $i \geq 1$, where Id denote the identity function. The obtained optimal complexity is then:

$$\begin{aligned} C^*_{n,m} &= s_1 - (n+1) + \sum_{i=1}^{m} \psi_n \circ (\psi_n + \mathrm{Id})^{(i-1)}(s_1) \\ &= (\psi_n + \mathrm{Id})^{(m)}(s_1) - (n+1) \ . \end{aligned}$$

By definition of ψ_n, the obtained functions $(\psi_n + \mathrm{Id})^{(i)}$ are sums of continued fractions with ceiling, for which we do not have an analytic expression.

Table 5. Optimal parameters with basis-update improvement.

n	$\lvert\mathcal{B}_1\rvert$	r	$t_1, t_2, \ldots, t_n$	$C^*_{n,n}$
4	7	2	2,1,1,1	7
5	11	5	2,2,2,1,1	13
	12	6	2,2,1,1,1	13
6	15	8	4,3,2,2,2,2	23
	16	9	3,3,2,2,2,2	23
7	23	15	5,4,3,3,3,3,2	38
8	31	22	8,6,5,5,4,4,4,3	61
	32	23	7,6,5,5,4,4,4,3	61
	33	24	7,6,5,5,4,4,3,3	61
	34	25	7,6,5,4,4,4,3,3	61
9	47	37	10,8,7,7,6,6,5,5,5	96
	48	38	10,8,7,7,6,5,5,5,5	96
	49	39	10,8,7,6,6,5,5,5,5	96
10	63	52	16,12,11,10,9,8,7,7,7,6	145
	64	53	15,12,11,10,9,8,7,7,7,6	145
	65	54	15,12,11,9,9,8,7,7,7,6	145

In Table 5, we give the optimal parameters $s_1 = |\mathcal{B}_1|$ and corresponding $r = s_1 - (n+1)$, t_1, t_2, $\ldots$, t_n, and $C^*_{n,n}$ for $n \times n$ s-boxes with $n \in [\![4, 10]\!]$. When the optimal multiplicative complexity is obtained for several values of s_1, we give all the obtained set of parameters. We observe that the optimal multiplicative complexity is always achieved by starting with the minimal basis *i.e.* by taking $\mathcal{B}_1 = \mathcal{B}_0$. It can also be obtained by taking s_1 up to $|\mathcal{B}_0| + 3$ depending on the values of n.

The achievable counterpart of Table 5 only exists with respect to a given s-box since the functions $g_{i,j} \cdot h_{i,j}$ added to the basis at each step depend on the actual s-box. But while focusing on a given s-box, we can still improve the method as we show hereafter.

Rank drop. Our second improvement is based on the observation that even if the matrix A is not full-rank, the obtained system can still have a solution for some given s-box. Specifically, if A is of rank $2^n - \delta$ then we should get a solution for one s-box out of 2^δ in average. Hence, instead of having t_i satisfying the condition $(t_i + 1)|\mathcal{B}| \geq 2^n$, we allow a rank drop in the system of equations, by taking $t_i \geq \frac{2^n - \delta}{|\mathcal{B}_i|} - 1$ for some integer δ for which solving 2^δ systems is affordable. We hence hope to get smaller values of t_i by trying 2^δ systems. Note that heuristically, we can only hope to achieve the above bound if δ is (a few times) lower than the maximal rank 2^n (*e.g.* $\delta \leq \frac{2^n}{4}$). We can then define the (theoretical) optimal sequence (s_i, t_i) and the corresponding multiplicative complexity $C^*_{n,m}$ from $s_1 = |\mathcal{B}_1|$ as in (33) and (35) by replacing the function ψ_n for $\psi_{n,\delta} \colon x \mapsto \lceil \frac{2^n - \delta}{x} \rceil - 1$. As an illustration, Table 6 provides the obtained parameters for a δ up to 32. We see that the rank-drop improvement (theoretically) saves a few multiplications.

Table 6. Optimal parameters with basis-update and rank-drop improvements.

n	δ	$\lvert\mathcal{B}_1\rvert$	r	$t_1, t_2, \ldots, t_n$	$C^*_{n,n}$
4	4	7	2	1,1,1,1	6
5	8	11	5	2,1,1,1,1	11
	8	12	6	1,1,1,1,1	11
6	16	15	8	3,2,2,2,1,1	19
	16	16	9	2,2,2,2,1,1	19
7	32	23	15	4,3,3,2,2,2,2	33
	32	24	16	3,3,3,2,2,2,2	33
8	32	31	22	7,5,5,4,4,3,3,3	56
8	32	32	23	6,5,5,4,4,3,3,3	56
9	32	47	37	10,8,7,6,6,5,5,5,4	93
	32	48	38	9,8,7,6,6,5,5,5,4	93
10	32	63	52	15,12,11,9,9,8,7,7,7,6	143
	32	64	53	15,12,10,9,9,8,7,7,7,6	143
	32	65	54	15,12,10,9,8,8,7,7,7,6	143
	32	66	55	15,12,10,9,8,8,7,7,6,6	143
	32	67	56	14,12,10,9,8,8,7,7,6,6	143

In practice, we observe that the condition $(t_i + 1)|\mathcal{B}_i| \geq 2^n - \delta$ is not always sufficient to get a matrix A of rank $2^n - \delta$. We shall then start with $t_i = \psi_{n,d}(|\mathcal{B}_i|)$ and try to solve $\alpha \cdot 2^\delta$ systems, for some constant α. In case of failure, we increment t_i and start again until a solvable system is found. The overall process is summarized in Algorithm 1.

The execution time of Algorithm 1 is dominated by the calls to a linear-solving procedure (Step 6). The number of trials is in $o(n\,\alpha\,2^\delta)$, where the con-

Algorithm 1 Improved method with exhaustive search

```
Input: An s-box $S \equiv (f_1, f_2, \ldots, f_m)$, parameters $s_1 = |\mathcal{B}_1|$, $\alpha$, and $\delta$
Output: A basis $\mathcal{B}_1$ and the functions $\{h_{i,j}\}_{i,j}$ and $\{g_{i,j}\}_{i,j}$
 1. $i = 1$; $t_1 = \psi_{n,\delta}(s_1)$
 2. do $\alpha \cdot 2^\delta$ times:
 3.     if $i = 1$ then randomly generate $\mathcal{B}_1 \supseteq \mathcal{B}_0$ with $|\mathcal{B}_1| = s_1$
 4.     randomly sample $t_i$ functions $g_{i,j} \in \langle \mathcal{B}_i \rangle$
 5.     compute the corresponding matrix $A$
 6.     if $A \cdot c = b_{f_i}$ has a solution then
 7.         store the corresponding functions $\{h_{i,j}\}_j$ and $\{g_{i,j}\}_j$
 8.         if $i = n$ then return $\mathcal{B}_1$, $\{h_{i,j}\}_{i,j}$, $\{g_{i,j}\}_{i,j}$
 9.         $\mathcal{B}_{i+1} = \mathcal{B}_i \cup \{h_{i,j} \cdot g_{i,j}\}_j$; $t_{i+1} = \psi_{n,\delta}(|\mathcal{B}_{i+1}|)$; $i$++
10.         goto Step 2
11.     endif
12. enddo
13. $t_i$++; goto Step 2
```

stant in the $o(\cdot)$ is the average incrementation of t_i (*i.e.* the average number of times Step 13 is executed per i). In our experiments, we observed that the optimal value of $t_1 = \psi_{n,\delta}(s_1)$ is rarely enough to get a solvable system for f_1. This is because we start with the minimal basis as in the single-Boolean-function case. We hence have a few incrementations for $i = 1$. On the other hand, the next optimal t_i's are often enough or incremented a single time.

We used Algorithm 1 to get the decomposition of various $n \times n$ s-boxes for $n \in [\![4, 8]\!]$, namely the eight 4×4 s-boxes of Serpent [ABK98], the s-boxes S_5 (5×5) and S_6 (6×6) of SC2000 [SYY$^+$02], the 8×8 s-boxes S_0 and S_1 of CLEFIA [SSA$^+$07], and the 8×8 s-box of Khazad [BR00]. The obtained results are summarized in Table 7. Note that we chose these s-boxes to serve as examples for our decomposition method. Some of them may have a mathematical structure allowing more efficient decomposition (*e.g.* the CLEFIA S_0 s-box is based on the inversion over $\mathbb{F}_{256}$ and can therefore be computed with a 32-multiplication circuit as the AES).

We observe that Algorithm 1 achieves improved parameters compared to the optimal ones with basis update and without the rank-drop improvement (see Table 5) for $n \in \{4, 5, 6\}$. For $n = 8$, we only get parameters close to the optimal ones for the basis update ($C^*_{n,n} = 62$ instead of 61). This can be explained by the fact that when n increases the value of δ becomes small compared to 2^n and the impact of exhaustive search is lowered. Thus Algorithm 1 can close the gap and (almost) achieve optimal parameters even in presence of a minimal starting basis, however it does not go beyond.

4.4 Parallelization

The proposed decomposition method is highly parallelizable. In practice, most SPN blockciphers have a nonlinear layer applying 16 or 32 s-boxes and most processors are based on a 32-bit or a 64-bit architecture. Therefore we shall

Table 7. Achieved parameters for several s-boxes.

	$\lvert\mathcal{B}_1\rvert$	r	$t_1, t_2, \ldots, t_n$	$C^*_{n,n}$
$n = 4$				
Serpent S_1–S_5	7	2	1, 1, 1, 1	6
Serpent S_6, S_7	7	2	1, 2, 1, 1	7
$n = 5$				
SC2000 S_5	11	5	2, 1, 1, 1, 1	11
	12	6	1, 1, 1, 1, 1	11
$n = 6$				
SC2000 S_6	15	8	4, 2, 2, 2, 2, 1	21
	16	9	3, 2, 2, 2, 2, 1	21
$n = 8$				
Khazad & CLEFIA (S_0, S_1)	31	22	11, 6, 5, 4, 4, 4, 3, 3	62
	33	24	9, 6, 5, 4, 4, 4, 3, 3	62
	32	23	10, 6, 5, 4, 4, 4, 3, 3	62

focus our study on the k-parallel multiplicative complexity of our method for $k \in \{2, 4\}$.

General method. In the general method (without improvement) described in Section 4.2, the multiplications between the g_j's and the $h_{i,j}$'s can clearly be processed in parallel. Specifically, they can be done with exactly $\lceil \frac{m \cdot t}{k} \rceil$ k-multiplications. The multiplications involved in the minimal basis $\mathcal{B}_0 = \{x \mapsto x^u, u \in \mathcal{U}\}$ can also be fully parallelized at degree $k = 2$ and $k = 4$ for every $n \geq 4$. In other words, the k-multiplicative complexity for deriving $\mathcal{B}_0$ equals $\lceil \frac{r_0}{k} \rceil$ for $k \in \{2, 4\}$ where $r_0 = C(\mathcal{B}_0) = |\mathcal{B}_0| - (n + 1)$ (see Section 4.1). One just has to compute x^u by increasing order of the Hamming weight of $u \in \mathcal{U}$ (where $\mathcal{U}$ is the set defined in (19)), then taking the lexicographical order inside a Hamming weight class. As an illustration, the 4-parallel evaluation of $\mathcal{B}_0$ is given for $n \in \{4, 6, 8\}$ in Table 8.

Once all the elements of $\mathcal{B}_0$ have been computed, and before getting to the multiplicative terms $g_j \cdot h_{i,j}$, we have to update it to a basis $\mathcal{B} \supseteq \mathcal{B}_0$ with target cardinality (see Table 4). This is done by feeding the basis with $|\mathcal{B}| - |\mathcal{B}_0|$ products of random linear combinations of the current basis. In order to parallelize this step, these new products are generated 4-by-4 from previous elements of the basis. We could validate that, by following such an approach, we still obtain full-rank systems with the achievable parameters given in Table 8. This means that for every $n \in [\![4, 10]\!]$, the k-multiplicative complexity of the general method is $\lceil \frac{r}{k} \rceil + \lceil \frac{m \cdot t}{k} \rceil$. The obtained results (achievable parameters) are summarized in Table 9.

Improved method. The parallelization of the improved method is slightly more tricky since all the multiplicative terms $g_{i,j} \cdot h_{i,j}$ cannot be computed in parallel. Indeed, the resulting products are fed to the basis so that they are potentially involved in the linear combinations producing the next functions

Table 8. Parallel evaluation of $\mathcal{B}_0$ for $n \in \{4, 6, 8\}$.

$n = 4$	$n = 6$	
$x_1x_2 \leftarrow x_2 \cdot x_1$	$x_1x_2 \leftarrow x_2 \cdot x_1$	$x_4x_6 \leftarrow x_6 \cdot x_4$
$x_3x_4 \leftarrow x_4 \cdot x_3$	$x_1x_3 \leftarrow x_3 \cdot x_1$	$x_5x_6 \leftarrow x_6 \cdot x_5$
	$x_2x_3 \leftarrow x_3 \cdot x_2$	$x_1x_2x_3 \leftarrow x_3 \cdot x_1x_2$
	$x_4x_5 \leftarrow x_5 \cdot x_4$	$x_4x_5x_6 \leftarrow x_6 \cdot x_4x_5$

$n = 8$		
$x_1x_2 \leftarrow x_2 \cdot x_1$	$x_2x_4 \leftarrow x_4 \cdot x_2$	$x_5x_8 \leftarrow x_8 \cdot x_5$
$x_1x_3 \leftarrow x_3 \cdot x_1$	$x_3x_4 \leftarrow x_4 \cdot x_3$	$x_6x_7 \leftarrow x_7 \cdot x_6$
$x_1x_4 \leftarrow x_4 \cdot x_1$	$x_5x_6 \leftarrow x_6 \cdot x_5$	$x_6x_8 \leftarrow x_8 \cdot x_6$
$x_2x_3 \leftarrow x_3 \cdot x_2$	$x_5x_7 \leftarrow x_7 \cdot x_5$	$x_7x_8 \leftarrow x_8 \cdot x_7$
$x_1x_2x_3 \leftarrow x_3 \cdot x_1x_2$	$x_5x_6x_7 \leftarrow x_7 \cdot x_5x_6$	$x_1x_2x_3x_4 \leftarrow x_4 \cdot x_1x_2x_2$
$x_1x_2x_4 \leftarrow x_4 \cdot x_1x_2$	$x_5x_6x_8 \leftarrow x_8 \cdot x_5x_6$	$x_5x_6x_7x_8 \leftarrow x_8 \cdot x_5x_6x_7$
$x_1x_3x_4 \leftarrow x_4 \cdot x_1x_3$	$x_5x_7x_8 \leftarrow x_8 \cdot x_5x_7$	
$x_2x_3x_4 \leftarrow x_4 \cdot x_2x_3$	$x_6x_7x_8 \leftarrow x_8 \cdot x_6x_7$	

Table 9. Parallel multiplicative complexity of our general method an $n \times n$ s-box.

n	4	5	6	7	8	9	10
(r, t)	(4,1)	(7,2)	(13,3)	(22,4)	(37,5)	(59,7)	(90,10)
$\lvert\mathcal{B}\rvert$	9	13	20	30	46	69	101
$C_{n,n}$	8	17	31	50	77	122	190
$C_{n,n}^{(2)}$	4	9	16	25	39	62	95
$C_{n,n}^{(4)}$	2	5	9	13	20	31	48

$g_{i+1,j}$, $h_{i+1,j}$, ..., $g_{m,j}$, $h_{m,j}$. In order to fully parallelize our improved method we customize Algorithm 1 as follows. We keep a counter q of the number of products added to the basis. Each time a new f_{i+1} is to be decomposed, if the current counter q is not a multiple of k, then the first q_0 products $g_{i+1,j} \cdot h_{i+1,j}$ will be bundled with the last q_1 products $g_{i,j} \cdot h_{i,j}$ in the parallel version of our improved decomposition, where

$$\begin{cases} q_0 = (k - q) \bmod k \\ q_1 = q \bmod k \end{cases} \tag{35}$$

We must then ensure that the functions $\{g_{i+1,j}, h_{i+1,j}\}_{j=0}^{q_0-1}$ are independent of the few last products $\{g_{i,j} \cdot h_{i,j}\}_{j=t_i-q_1}^{t_i-1}$. This can be done at no cost for the $g_{i+1,j}$'s which can be generated without the last q_1 products, and this add a constraint on the linear system for the first q_0 searched $h_{i+1,j}$ functions. However, we observed in our experiments that for small values of k such as $k \in \{2, 4\}$, this constraint has a negligible impact on Algorithm 1. We could actually obtain the exact same parameters than in Table 7 for all the tested s-boxes (Serpent, SC2000, CLEFIA, and Khazad) for a parallelization degree of $k = 2$, except for the s-box S_3 of Serpent that requires 1 more multiplication.

5 Implementations

This section describes our implementations of a bitsliced s-box layer protected with higher-order masking based on our decomposition method. Our implementations evaluate 16 $n \times n$ s-boxes in parallel where $n \in \{4, 8\}$, and they are developed in generic 32-bit ARM assembly. They take n input sharings $[\boldsymbol{x}_1]$, $[\boldsymbol{x}_2]$, …, $[\boldsymbol{x}_n]$ defined as

$$[\boldsymbol{x}_i] = (\boldsymbol{x}_{i,1}, \boldsymbol{x}_{i,2}, \ldots, \boldsymbol{x}_{i,d}) \text{ such that } \sum_{j=1}^{d} \boldsymbol{x}_{i,j} = \boldsymbol{x}_i \tag{36}$$

where $\boldsymbol{x}_i$ is a 16-bit register containing the i-th bit of the 16 s-box inputs. Our implementations then output n sharings $[\boldsymbol{y}_0]$, $[\boldsymbol{y}_1]$, …, $[\boldsymbol{y}_n]$ corresponding to the bitsliced output bits of the s-box. Since we are on a 32-bit architecture with 16-bit bitsliced registers, we use a degree-2 parallelization for the multiplications. Namely, the 16-bit ANDs are packed by pairs and replaced by 32-bit ANDs which are applied on shares using the ISW scheme as explained in [GR16].

The computation is then done in three stages. First, we need to construct the shares of the elements of the minimal basis $\mathcal{B}_0$, specifically $[\boldsymbol{x}^u]$ for every $u \in \mathcal{U}$, where $\boldsymbol{x}^u$ denote the bitsliced register for the bit x^u, and where $\mathcal{U}$ is the set defined in (19). This first stage requires $r_0/2$ 32-bit ISW-ANDs, where $r_0 = 2$ for $n = 4$ and $r_0 = 22$ for $n = 8$ (see Table 8).

Once the first stage is completed, all the remaining multiplications are done between linear combinations of the elements of the basis. Let us denote by $[\boldsymbol{t}_i]$ the sharings corresponding to the elements of the basis which are stored in memory. After the first stage we have $\{[\boldsymbol{t}_i]\} = \{[\boldsymbol{x}^u] \mid u \in \mathcal{U}\}$. Each new $\boldsymbol{t}_i$ is defined as

$$\Big(\sum_{j<i} a_{i,j} \boldsymbol{t}_j\Big) \odot \Big(\sum_{i<j} b_{i,j} \boldsymbol{t}_j\Big) \tag{37}$$

where $\odot$ denote the bitwise multiplication, and where $\{a_{i,j}\}_j$ and $\{b_{i,j}\}_j$ are the binary coefficients obtained from the s-box decomposition (namely the coefficients of the functions $g_{i,j}$ and $h_{i,j}$ in the span of the basis). The second stage hence consists in a loop on the remaining multiplications that

1. computes the linear-combination sharings $[\boldsymbol{r}_i] = \sum_{j<i} a_{i,j} [\boldsymbol{t}_j]$ and $[\boldsymbol{s}_i] = \sum_{j<i} b_{i,j} [\boldsymbol{t}_j]$
2. refreshes the sharing $[\boldsymbol{r}_i]$
3. computes the sharing $[\boldsymbol{t}_i]$ such that $\boldsymbol{t}_i = \boldsymbol{r}_i \odot \boldsymbol{s}_i$

where the last step is performed for two successive values of i at the same time by a call to a 32-bit ISW-AND. The sums in Step 1 are performed on each share independently. The necessity of the refreshing procedure in Step 2 is explained in [GR16] since an ISW multiplication of two linear combinations of the same sharings can introduce a security flaw (see for instance [CPRR14]). As in [GR16], this refreshing is implemented from an ISW multiplication with $(1, 0, 0, \ldots, 0)$.

Once all the basis sharings $[\boldsymbol{t}_i]$ have been computed, the third stage simply consists in deriving each output sharing $[\boldsymbol{y}_i]$ as a linear combination of the $[\boldsymbol{t}_i]$, which is refreshed before being returned.

We compare our results with the optimized implementations from [GR16] of the CRV method [CRV14] and the algebraic decomposition (AD) method [CPRR15]. These implementations compute four s-boxes in parallel for $n = 8$ and eight s-boxes in parallel for $n = 4$ on a 32-bit ARM architecture. Table 10 summarizes the obtained performances in clock cycles with respect to the masking order d. It is worth noticing that packing and unpacking the bitslice registers for the parallelization of the ISW-ANDs implies a linear overhead in d. For $d \in [\![2, 20]\!]$, this overhead is between 4% and 6% of the overall s-box computations for $n = 8$, and between 7% and 11% for $n = 4$ (and this ratio is asymptotically negligible). For $d = 2$, the overhead slightly exceeds the gain, but for every $d \geq 3$, parallelizing the ISW-ANDs always results in an overall gain of performances.

Table 10. Performances in clock cycles.

	CRV [GR16]	**AD [GR16]**	**Our implementations**
	$4 \times 4 /\!/$ s-boxes	$4 \times 4 /\!/$ s-boxes	$16 /\!/$ s-boxes
$n = 8$	$2576\,d^2 + 5476\,d + 2528$	$2376\,d^2 + 3380\,d + 5780$	$656\,d^2 + 19786\,d + 5764$
	$2 \times 8 /\!/$ s-boxes	$2 \times 8 /\!/$ s-boxes	$16 /\!/$ s-boxes
$n = 4$	$337\,d^2 + 563\,d + 434$	$564\,d^2 + 270\,d + 660$	$59\,d^2 + 1068\,d + 994$

We observe that our implementations are asymptotically faster than the optimized implementations of CRV and AD methods (3.6 times faster for $n = 8$ and 5.7 times faster for $n = 4$). However, we also see that the linear coefficient is significantly greater for our implementations, which comes from the computation of the linear combinations in input of the ISW-ANDs (*i.e.* the sharings $[\boldsymbol{r}_i]$ and $[\boldsymbol{s}_i]$). As an illustration, Figures 1 and 2 plots the obtained timings with respect to d. We see that for $n = 4$, our implementation is always faster than the optimized AD and CRV. On the other hand, for $n = 8$, our implementation is slightly slower for $d \leq 8$. We stress that our implementations could probably be improved by optimizing the computation of the linear combinations.

The RAM consumption and code size of our implementations are given in Table 11 and compared to those of the CRV and AD implementations from [GR16]. We believe these memory requirements to be affordable for not-too-constrained embedded devices. In terms of code size, our implementations are always the best. This is especially significant for $n = 8$ where CRV and AD needs a high amount of storage for the lookup tables of the linearized polynomials (see [GR16]). On the other hand, we observe a big gap between our implementations and those from [GR16] regarding the RAM consumption. Our method is indeed more consuming in RAM because of all the $[\boldsymbol{t}_i]$ sharings that must be stored while such a large basis is not required for the CRV and AD methods, and

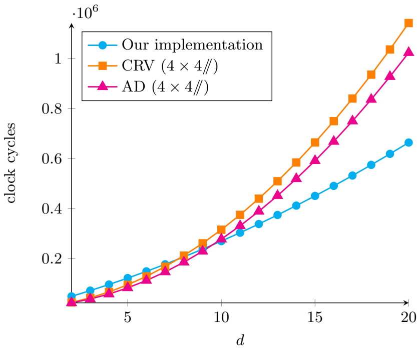

Fig. 1. Timings for $n = 8$.

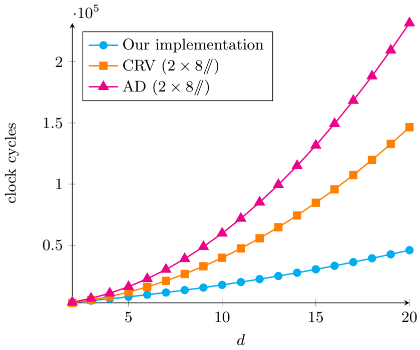

Fig. 2. Timings for $n = 4$.

because of some optimizations in the computation of the linear combinations (see the full version).

Table 11. Code sizes and RAM consumptions.

	CRV [GR16]	**AD [GR16]**	**Our implementations**
$n = 8$	$4 \times 4/\!/$ s-boxes	$4 \times 4/\!/$ s-boxes	$16/\!/$ s-boxes
Code size	27.5 KB	11.2 KB	4.6KB
RAM	80d bytes	188d bytes	644d bytes
$n = 4$	$2 \times 8/\!/$ s-boxes	$2 \times 8/\!/$ s-boxes	$16/\!/$ s-boxes
Code size	3.2 KB	2.6 KB	2.2 KB
RAM	24d bytes	64d bytes	132d bytes

References

[ABK98] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A Proposal for the Advanced Encryption Standard. NIST AES Proposal, 1998.

[BGRV15] Josep Balasch, Benedikt Gierlichs, Oscar Reparaz, and Ingrid Verbauwhede. DPA, bitslicing and masking at 1 GHz. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 599–619, Saint-Malo, France, September 13–16, 2015. Springer, Heidelberg, Germany.

[BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 26(2):280–312, April 2013.

[BPP00] Joan Boyar, René Peralta, and Denis Pochuev. On the multiplicative complexity of Boolean functions over the basis $(\wedge, \oplus, \mathbf{1})$. *Theor. Comput. Sci.*, 235(1):43–57, 2000.

[BR00] Paulo Barreto and Vincent Rijmen. The Khazad Legacy-Level Block Cipher. First Open NESSIE Workshop, 200.

[Can05] D. Canright. A very compact S-box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455, Edinburgh, UK, August 29 – September 1, 2005. Springer, Heidelberg, Germany.

[CGP$^+$12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for S-boxes. In Anne Canteaut, editor, *Fast Software Encryption – FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384, Washington, DC, USA, March 19–21, 2012. Springer, Heidelberg, Germany.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.

[CMH13] Nicolas Courtois, Theodosis Mourouzis, and Daniel Hulme. Exact Logic Minimization and Multiplicative Complexity of Concrete Algebraic and Cryptographic Circuits. *Advances in Intelligent Systems*, 6(3–4):43–57, 2013.

[Cou07] Nicolas T. Courtois. CTC2 and fast algebraic attacks on block ciphers revisited. Cryptology ePrint Archive, Report 2007/152, 2007. `http://eprint.iacr.org/2007/152`.

[CPRR14] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany.

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 742–763, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[CRV14] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 170–187, Busan, South Korea, September 23–26, 2014. Springer, Heidelberg, Germany.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[DPV01] Joan Daemen, Michael Peeters, and Gilles Van Assche. Bitslice ciphers and power analysis attacks. In Bruce Schneier, editor, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 134–149, New York, NY, USA, April 10–12, 2001. Springer, Heidelberg, Germany.

[GLSV15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption – FSE 2014*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37, London, UK, March 3–5, 2015. Springer, Heidelberg, Germany.

[GR16] Dahmun Goudarzi and Matthieu Rivain. How Fast Can Higher-Order Masking Be in Software? Cryptology ePrint Archive, 2016. `http://eprint.iacr.org/`.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

[MS92] Roland Mirwald and Claus-Peter Schnorr. The Multiplicative Complexity of Quadratic Boolean Forms. *Theor. Comput. Sci.*, 102(2):307–328, 1992.

[PLW10] Axel Poschmann, San Ling, and Huaxiong Wang. 256 bit standardized crypto for 650 GE - GOST revisited. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 219–233, Santa Barbara, California, USA, August 17–20, 2010. Springer, Heidelberg, Germany.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427, Santa Barbara, California, USA, August 17–20, 2010. Springer, Heidelberg, Germany.

[SSA$^+$07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *Fast Software Encryption – FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195, Luxembourg, Luxembourg, March 26–28, 2007. Springer, Heidelberg, Germany.

[Sto16] Ko Stoffelen. Optimizing s-box implementations for several criteria using sat solvers. Fast Software Encryption, 2016.

[SYY$^+$02] Takeshi Shimoyama, Hitoshi Yanami, Kazuhiro Yokoyama, Masahiko Takenaka, Kouichi Itoh, Jun Yajima, Naoya Torii, and Hidema Tanaka. The block cipher SC2000. In Mitsuru Matsui, editor, *Fast Software Encryption – FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 312–327, Yokohama, Japan, April 2–4, 2002. Springer, Heidelberg, Germany.

[TP14] Meltem Sönmez Turan and René Peralta. The Multiplicative Complexity of Boolean Functions on Four and Five Variables. In Thomas Eisenbarth and Erdinç Öztürk, editors, *Lightweight Cryptography for Security and Privacy - Third International Workshop, LightSec 2014, Istanbul, Turkey, September 1-2, 2014, Revised Selected Papers*, volume 8898 of *Lecture Notes in Computer Science*, pages 21–33. Springer, 2014.