# Algebraic Decomposition for Probing Security

Claude Carlet[1], Emmanuel Prouff[2], Matthieu Rivain[3], and Thomas Roche[2]

[1] LAGA, UMR 7539, CNRS, Department of Mathematics,
University of Paris XIII and University of Paris VIII
claude.carlet@univ-paris8.fr

[2] ANSSI
firstname.name@ssi.gouv.fr

[3] CryptoExperts
matthieu.rivain@cryptoexperts.com

**Abstract.** The probing security model is very popular to prove the side-channel security of cryptographic implementations protected by masking. A common approach to secure nonlinear functions in this model is to represent them as polynomials over a binary field and to secure their nonlinear multiplications thanks to a method introduced by Ishai, Sahai and Wagner at Crypto 2003. Several schemes based on this approach have been published, leading to the recent proposal of Coron, Roy and Vivek which is currently the best known method when no particular assumption is made on the algebraic structure of the function. In the present paper, we revisit this idea by trading nonlinear multiplications for low-degree functions. Specifically, we introduce an algebraic decomposition approach in which a nonlinear function is represented as a sequence of functions with low algebraic degrees. We therefore focus on the probing-secure evaluation of such low-degree functions and we introduce three novel methods to tackle this particular issue. The paper concludes with a comparative analysis of the proposals, which shows that our algebraic decomposition method outperforms the method of Coron, Roy and Vivek in several realistic contexts.

## 1 Introduction

Since their introduction in [15, 16], side-channel attacks are known to be a serious threat against implementations of cryptographic algorithms. Among the existing strategies to secure an implementation, one of the most widely used relies on *secret sharing* (aka *masking*). Using secret sharing at the implementation level enables to achieve provable security in the so-called *probing security model* [13]. In this model, it is assumed that an adversary can recover information on a limited number of intermediate variables of the computation. This model has been argued to be practically relevant to address so-called *higher-order* side-channel attacks and it has been the basis of several efficient schemes to protect block ciphers [1, 6, 11, 12, 23, 24]. More recently, it has been shown in [10] that the probing security of an implementation actually implies its security in the more realistic *noisy leakage model* introduced in [22]. This makes probing security a very appealing feature for the design of secure implementations against side-channel attacks.

The first generic probing secure scheme, here called the ISW scheme, was designed by Ishai, Sahai and Wagner in [13]. It was later used by Rivain and Prouff to design an efficient probing-secure implementation of AES.[4] Several works then followed to improve this approach and to extend it to other SPN block ciphers [6, 8, 9, 14, 25]. In a nutshell, these methods consist in representing the nonlinear functions of such ciphers (the so-called *s-boxes*) as polynomials over a binary field. Evaluating such polynomials then involves some *linear* operations (*e.g.* squaring, addition, multiplication by constants) which are secured with straightforward and efficient methods, and some so-called *nonlinear multiplications* which are secured using ISW. The proposed polynomial evaluation methods then aim at minimizing the number of nonlinear multiplications. The method recently introduced by Coron, Roy and Vivek in [9] (called CRV in the following) is currently the most efficient scheme for the probing-secure evaluation of nonlinear functions without particular algebraic structure.

In this paper we introduce a new approach based on *algebraic decomposition* for the probing-secure evaluation of nonlinear functions. More precisely, we propose a method — inspired from [9] — to efficiently compute a function from the evaluation of several other functions having a low algebraic degree. We subsequently study the problem of designing efficient probing-secure evaluation methods for low-degree functions. Specifically, we introduce three different methods to tackle this issue with different and complementary assets. The first one relies on a recursive higher-order derivation of the target function. Remarkably, it enables to get a $t$-probing secure evaluation of a function of algebraic degree $s$ (for arbitrary order $t$) from several $s$-probing secure evaluations of the function. Namely, for degree-$s$ functions, it reduces $t$-probing security to $s$-probing security. This method has a complexity exponential in $s$ and is hence only interesting for low-degree functions. In particular, for the case $s = 2$, we show how to specialize it into an efficient algorithm which happens to be a generalization of a previous method proposed in [8] to secure a peculiar type of multiplications (specifically $x \mapsto x \cdot \ell(x)$ where $\ell$ is linear). Our generalization of this algorithm can be applied to secure *any* quadratic function more efficiently than a *single* multiplication with the ISW scheme. Our second approach also applies a recursive derivation technique, but in a more direct way which allows us to design a stand-alone probing-secure evaluation method. Interestingly, this method does not rely on the polynomial representation of the target function $h$, and its processing only involves additions and evaluations of $h$. When the latter evaluation can be tabulated (which is often the case in the context of s-boxes), the proposed algorithm can be a valuable alternative to the state-of-the-art solutions based

[4] The original proposal of [24] actually involved a weak mask refreshing algorithm and the weakness was exploited in [8] to exhibit a flaw in the s-box processing. The authors of [8] also proposed a correction, which was recently verified for $d \leqslant 4$ using program verification techniques [2].

on field multiplications. However, this method also has a complexity exponential in the algebraic degree of $h$, which makes it only interesting for low degree functions. Eventually, the third method consists in tweaking the CRV method for the case of low-degree functions. The tweak is shown to substantially reduce the number of nonlinear multiplications for functions of a given (low) algebraic degree.

The theoretical analysis of our proposals is completed by an extensive comparative analysis considering several ratios for the cost of a field multiplication over the cost of other elementary operations. The results show that for functions of algebraic degree 2 our generalization of [8] is always the most efficient. On the other hand, for functions of algebraic degree 3, the tweaked CRV method achieves the best performances except for small probing orders for which our second method takes the advantage. For high-degree functions, our algebraic decomposition method reaches its best performances when it is combined with our generalization of [8]. For some multiplication cost ratios, our method is more efficient than CRV, which makes it the currently best known method for the probing-secure evaluation of nonlinear functions in these scenarios. As an example, for functions of dimension $n = 8$, our method outperforms CRV whenever a field multiplication takes more than 5 elementary operations (which we believe to be a realistic scenario).

From a general point of view, this paper shows that the issue of designing efficient probing-secure schemes for nonlinear functions may be reduced to the secure evaluation of functions of small algebraic degrees. This approach, and the first encouraging results reported in this paper, opens a promising avenue for further research on this topic. Our work also has strong connections with the field of *threshold implementations* in which secret sharing techniques are used to design hardware masking schemes resistant to glitches [19, 20]. An efficient threshold implementation is indeed usually obtained by decomposing the target nonlinear function into lower-degree functions (ideally quadratic functions). In this context, some decompositions have been proposed for specific functions, such as the PRESENT s-box [21], the AES s-box [18], and the DES s-boxes [3]. Some works have also followed an exhaustive approach to provide decompositions for small-size s-boxes, specifically all bijective 3-bit and 4-bit s-boxes [3, 17]. Certain 5-bit and 6-bit s-boxes have also been considered in [4].

## 2 Preliminaries

### 2.1 Functions in Finite Fields and Derivation

Along this paper, we shall denote by $[\![i, j]\!]$ the integer interval $[i, j] \cap \mathbb{Z}$ for any pair of integers $(i, j)$ with $i \leqslant j$, and we shall denote by $\mathbb{F}_{2^n}$ the finite field with $2^n$ elements for any integer $n \geqslant 1$. Choosing a basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ enables to represent elements of $\mathbb{F}_{2^n}$ as elements of the vector space $\mathbb{F}_2^n$ and *vice versa*. In the following, we assume that the same basis is always used to represent elements of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$. For any positive integers $m \leqslant n$, a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2^m$ is called an $(n, m)$*-function* (the dimensions could be omitted if they are clear from the context). Nonlinear functions used in block-ciphers are usually called *s-boxes*. The set of $(n,m)$-functions is denoted by $\mathcal{B}_{n,m}$. When $m$ divides $n$, any function $h \in \mathcal{B}_{n,m}$ can be represented by a polynomial function $x \in \mathbb{F}_{2^n} \mapsto \sum_{i=0}^{2^n-1} a_i x^i$ where the $a_i$ are constant coefficients in $\mathbb{F}_{2^n}$ that can be obtained by applying Lagrange's Interpolation. When $m$ does not divide $n$, the $m$-bit outputs of $h$ can be embedded into $\mathbb{F}_{2^n}$ by padding them to $n$-bit outputs (*e.g.* by setting the most significant bits to 0). This ensures that any function $h \in \mathcal{B}_{n,m}$ can be evaluated as a polynomial over $\mathbb{F}_{2^n}$. If padding has been used, then it can be removed after the polynomial evaluation by mapping the output from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2^n$. The *algebraic degree* of a function $h \in \mathcal{B}_{n,m}$ is the integer value $\max_{a_i \neq 0}(\texttt{HW}(i))$ where the $a_i$ are the coefficients of the polynomial representation of $h$ and where $\texttt{HW}(i)$ denotes the Hamming weight of $i$ (see *e.g.* [5] for more details about this notion). The algebraic degree must not be confused with the classical notion of polynomial degree which is the integer value $\max_{a_i \neq 0}(i)$. A function of algebraic degree $s$ will sometimes be called a *degree-s function*.

This paper intensively uses the notion of *higher-order derivative* of a $(n,m)$-function. It is is recalled hereafter.

**Definition 1 (Higher-Order Derivative).** *Let $n$ and $m$ be two positive integers such that $m \leqslant n$ and let $h \in \mathcal{B}_{n,m}$. For any positive integer $t$ and any $t$-tuple $(a_1, a_2, \ldots, a_t) \in (\mathbb{F}_2^n)^t$, the $(n,m)$-function $D_{a_1,a_2,\ldots,a_t}h$ which maps any $x \in \mathbb{F}_{2^n}$ to $\sum_{I \subseteq [\![1,t]\!]} h\Big(x + \sum_{i \in I} a_i\Big)$ is called the* $t^{\text{th}}$-order derivative *of $h$ with respect to $(a_1, a_2, \ldots, a_t)$.*

Any $t^{\text{th}}$-order derivative of a degree-$s$ function has an algebraic degree bounded above by $s-t$. In the following we shall denote by $\varphi_h^{(t)}$ the $(n,m)$-function defined by

$$\varphi_h^{(t)} : (a_1, a_2, \ldots, a_t) \mapsto D_{a_1,a_2,\ldots,a_t}\, h(0) \ . \tag{1}$$

This function has some well-known properties recalled hereafter.

**Proposition 1.** *Let $h \in \mathcal{B}_{n,m}$ be a degree-$s$ function. Then, the function $\varphi_h^{(s)}$ is $s$-linear symmetric and equals zero for any family of $a_i$ linearly dependent over $\mathbb{F}_2$. Conversely, if $\varphi_h^{(s)}$ is $s$-linear, then the algebraic degree of $h$ is at most $s$.*

## 2.2 Sharing and Probing Security

For two positive integers $k$ and $d$, a $(k,d)$-*sharing* of a variable $x$ defined over some finite field $\mathbb{K}$ is a random vector $(x_1, x_2, \ldots, x_k)$ over $\mathbb{K}^k$ such that $x = \sum_{i=1}^{k} x_i$ holds (*completeness equality*) and any tuple of $d-1$ shares $x_i$ is a uniform random vector over $\mathbb{K}^{d-1}$. The variable $x$ is also called the *plain value* of the sharing $(x_i)_i$. If $k = d$, the terminology simplifies to *d-sharing*.

An algorithm with domain $\mathbb{K}^d$ is said *t-probing secure* if on input a $d$-sharing $(x_1, x_2, \ldots, x_d)$ of some variable $x$, it admits no tuple of $t$ or less intermediate variables that depends on $x$. An algorithm achieving $t$-probing security is resistant to the class of $t^{\text{th}}$-order side-channel attacks (see for instance [7, 22, 24]). In this paper we will further use the following non-standard notion.

**Definition 2 (perfect $t$-probing security).** *An algorithm is said to achieve* perfect $t$-probing security *if every $\ell$-tuple of its intermediate variables can be perfectly simulated from an $\ell$-tuple of its input shares for every $\ell \leqslant t$.*

It can be shown that for the tight security case $t = d - 1$, the notion above is equivalent to the usual $t$-probing security.

**Lemma 1.** *An algorithm taking a $d$-sharing as input achieves $(d-1)$-probing security if and only if it achieves perfect $(d-1)$-probing security.*

We shall simply say that an algorithm taking a $d$-sharing as input is probing secure when it achieves (perfect) $(d-1)$-probing security. We will also say that an algorithm admits a $t$-order flaw when $t$ of its intermediate variables jointly depend on the plain input (hence contradicting $t$-probing security).

It is worth noticing that achieving (perfect) probing security for the evaluation of a linear function $g$ is pretty simple. Computing a $d$-sharing of $g(x)$ from a $d$-sharing $(x_1, x_2, \ldots, x_d)$ of $x$ can indeed be done by applying $g$ to every share. The process is clearly (perfectly) $t$-probing secure with $t = d - 1$ and the completeness holds by linearity of $g$ since we have $g(x) = g(x_1) + g(x_2) + \cdots + g(x_d)$. The same holds for an affine function but the constant term $g(0)$ must be added to the right side if and only if $d$ is odd (without impacting the probing security).

Probing security is more tricky to achieve for nonlinear functions. In [13], Ishai, Sahai, and Wagner tackled this issue by introducing the first generic $t$-probing secure scheme for the multiplication over $\mathbb{F}_2$ which can be easily extended to the multiplication over any finite field. Let $(x_i)_i$ and $(y_i)_i$ be the $d$-sharings of two variables $x$ and $y$ over some binary field $\mathbb{K}$, the ISW scheme proceeds as follows:

1. for every $1 \leqslant i < j \leqslant d$, sample a random value $r_{i,j}$ over $\mathbb{K}$ ,
2. for every $1 \leqslant i < j \leqslant d$, compute $r_{j,i} = (r_{i,j} + a_i \cdot b_j) + a_j \cdot b_i$ ,
3. for every $1 \leqslant i \leqslant d$, compute $c_i = a_i \cdot b_i + \sum_{j \neq i} r_{i,j}$ .

The completeness holds from $\sum_i c_i = \sum_{i,j} a_i \cdot b_j = (\sum_i a_i)(\sum_j b_j)$ since every random value $r_{i,j}$ appears exactly twice in the sum and hence vanishes.[5] The above multiplication procedure was proved $t$-probing secure with $t \leqslant (d-1)/2$ in [13]. This was later improved in [24] to a tight $t$-probing security with $t \leqslant d-1$.

### 2.3 Secure Evaluation of Nonlinear Functions

In the original scheme of Ishai *et al.*, a computation is represented as a Boolean circuit composed of logical operations NOT and AND. In [24], Rivain and Prouff generalized their approach to larger fields which allowed them to design an efficient probing-secure computation of the AES cipher. The main issue in securing SPN ciphers like AES lies in the s-box computations (the rest of the cipher being purely linear operations). The Rivain-Prouff scheme computes the AES s-box

[5] This is true since $\mathbb{K}$ is a binary field, but the method can easily be extended to any field by defining $r_{j,i} = (a_i \cdot b_j - r_{i,j}) + a_j \cdot b_i$ in Step 2.

using 4 multiplications over $\mathbb{F}_{2^8}$ (secured with ISW) plus some linear operations (specifically squaring over $\mathbb{F}_{2^8}$, and the initial affine transformation). This approach was then extended in [6] to secure any s-box in $\mathcal{B}_{n,m}$. The principle is to represent the s-box as a polynomial $\sum_i a_i \cdot x^i$ in $\mathbb{F}_{2^n}[x]$ whose evaluation is then expressed as a sequence of linear functions (*e.g.* squaring over $\mathbb{F}_{2^n}$, additions, multiplications by coefficients) and *nonlinear multiplications* over $\mathbb{F}_{2^n}$. The former operations can be simply turned into probing-secure operations of complexity $O(d)$ as described in the previous section, whereas the latter operations are secured using ISW with complexity $O(d^2)$. The total complexity is hence mainly impacted by the number of nonlinear multiplications involved in the underlying polynomial evaluation method. This observation led to a series of publications aiming at developing polynomial evaluation methods with least possible costs in terms of nonlinear multiplications. Some heuristics in this context were proposed by Carlet *et al.* in [6], and then improved by Roy and Vivek in [25] and by Coron, Roy and Vivek in [9]. In the latter reference, a method, that we shall call the CRV method, is introduced. It is currently the most efficient one for probing-secure evaluation of nonlinear functions.

**Tabulated multiplications.** Further works have shown that secure evaluation methods could be improved by relying on specific kind of nonlinear multiplications. Assume for instance that the dimension $n$ is such that a lookup table with $2^{2n}$ entries is too big for some given architecture whereas a lookup table with $2^n$ entries fits (*e.g.* for $n = 8$ one needs 64 kilobytes for the former and 256 bytes for the latter). This means that a multiplication over $\mathbb{F}_{2^{n/2}}$ can be computed using a single lookup, whereas a multiplication over $\mathbb{F}_{2^n}$ must rely on more costly arithmetic (*e.g.* schoolbook method, log-exp tables, Karatsuba, etc.). This observation was used by Kim, Hong, and Lim in [14] to design an efficient alternative to the Rivain-Prouff scheme based on the so-called *tower-field representation* of the AES s-box [26]. Using this representation, the AES s-box can be computed based on 5 multiplications over $\mathbb{F}_{2^4}$ instead of 4 multiplications over $\mathbb{F}_{2^8}$, which results in a significant gain of performance when the former are tabulated while the latter are computed using log-exp tables (although one more multiplication is involved). Another approach, proposed in [8] by Coron, Prouff, Rivain, and Roche, is to consider the multiplications of the form $x \cdot \ell(x)$ where $\ell$ is a linear function. Such multiplications can be secured using a variant of ISW relying on a table for the function $x \mapsto x \cdot \ell(x)$. This variant that we shall call the CPRR scheme, allowed the authors to design a faster probing-secure scheme for the AES s-box. It was further used in [12] to reduce the complexity of the probing-secure evaluation of power functions in $\mathbb{F}_{2^n}$ with $n \leqslant 8$.

## 3 The Algebraic Decomposition Method

In this section, we introduce a new algebraic decomposition method that split the evaluation of a nonlinear function with arbitrary algebraic degree into several evaluations of functions with given (low) algebraic degree $s$. Our proposed method is inspired from the CRV method but relies on low-degree polynomial evaluations instead of nonlinear multiplications.

We consider a function $h \in \mathcal{B}_{n,m}$ which is seen as a polynomial $h(x) = \sum_{j=0}^{2^n-1} a_j x^j$ over $\mathbb{F}_{2^n}[x]$. We start by deriving a family of generators $(g_i)_i$ as follows:

$$\begin{cases} g_1(x) = f_1(x) \\ g_i(x) = f_i\big(g_{i-1}(x)\big) \end{cases} , \tag{2}$$

where the $f_i$ are random polynomials of given algebraic degree $s$. Then we randomly generate $t$ polynomials $(q_i)_i$ over the subspace of polynomials $\sum_{j=1}^{r} \ell_j \circ g_j$ where the $\ell_j$ are linearized polynomials (*i.e.* polynomials of algebraic degree 1). This is done by sampling random linearized polynomials $(\ell_{i,j})_{i,j}$ and by computing

$$q_i(x) = \sum_{j=1}^{r} \ell_{i,j}\big(g_j(x)\big) + \ell_{i,0}(x) \ . \tag{3}$$

Eventually, we search for $t$ polynomials $p_i$ of algebraic degree $s$ and for $r+1$ linearized polynomials $\ell_i$ such that

$$h(x) = \sum_{i=1}^{t} p_i\big(q_i(x)\big) + \sum_{i=1}^{r} \ell_i\big(g_i(x)\big) + \ell_0(x) \ . \tag{4}$$

From such polynomials, we get a method to compute $h(x)$ by subsequently evaluating (2), (3) and (4). This method involves $r+t$ evaluations of degree-$s$ polynomials (the $f_i$ and the $p_i$), plus some linear operations. The following table gives the exact operation counts (where "#eval deg-$s$" denotes the number of evaluations of degree-$s$ functions, "#eval LP" denotes the number of evaluations of linearized polynomials, and "#add" denotes the number of additions).

| #eval deg-$s$ | #eval LP | #add |
|---|---|---|
| $r+t$ | $(t+1)(r+1)$ | $r \cdot t + t + r$ |

The complexity of the resulting $d$-probing secure evaluation can be obtained by multiplying by $d$ the number of additions and the number of evaluations of linearized polynomials (which might be tabulated) and by adding $r+t$ secure evaluations of degree-$s$ functions.

*Remark 1.* The generation step of our method can be generalized as follows:

$$\begin{cases} g_1(x) = f_1(x) \\ g_i(x) = f_i\Big(\sum_{j=1}^{i-1} \ell'_{i,j}\big(g_j(x)\big)\Big) \end{cases} \tag{5}$$

where the $f_i$ are random polynomials of given algebraic degree $s$ and the $\ell'_{i,j}$ are random linearized polynomials. This generalization has no impact when $r \leqslant 2$. In particular, it has no impact on our experimental results for $s \in \{2,3\}$ and $n \in [\![4,8]\!]$ (indeed, the best counts are always obtained with $r \leqslant 2$ since we stop at $g_2(x) = f'_2 \circ f_1(x)$ where $f'_2 = f_2 \circ \ell_{2,1}$ is of degree $s_1$ – see hereafter – ). However, (5) might give better results than (2) for higher values of $n$ and/or $s$.

As in the CRV method, the search of polynomials $(p_i)_i$ and $(\ell_i)_i$ satisfying (4) for given polynomials $(g_i)_i$ and $(q_i)_i$ amounts to solve a system of linear equations over $\mathbb{F}_{2^n}$:

$$A \cdot \boldsymbol{b} = \boldsymbol{c} \ . \tag{6}$$

The target vector $\boldsymbol{c}$ has $2^n$ coordinates which are the values taken by $h(x)$ for all $x$ over $\mathbb{F}_{2^n}$, that is

$$\boldsymbol{c} = \big(h(e_1), h(e_2), h(e_3), \ldots, h(e_{2^n})\big) \ ,$$

where $\{e_1, e_2, \ldots, e_{2^n}\} = \mathbb{F}_{2^n}$. The coordinates of the vector $\boldsymbol{b}$ are the variables of the system that represents the solutions for the coefficients of the polynomials $(p_i)_i$ and $(\ell_i)_i$. The matrix $A$ is then defined as the concatenation of several submatrices:

$$A = (\mathbf{1} \mid A_{p_1} \mid A_{p_2} \mid \cdots \mid A_{p_t} \mid A_{\ell_0} \mid A_{\ell_1} \mid \cdots \mid A_{\ell_r}) \ ,$$

where $\mathbf{1}$ is the $2^n$-dimensional column vector with all coordinates equal to $1 \in \mathbb{F}_{2^n}$, where the $A_{p_i}$ are $2^n \times N_s$ submatrices, with $N_s = \sum_{d=1}^{s} \binom{n}{d}$, and where the $A_{\ell_i}$ are $2^n \times n$ submatrices. For every $i \in [\![1, t]\!]$, the $2^n \times N_s$ matrix $A_{p_i}$ is defined as:

$$A_{p_i} = \begin{pmatrix} q_i(e_1)^{\beta_1} & q_i(e_1)^{\beta_2} & \cdots & q_i(e_1)^{\beta_{N_s}} \\ q_i(e_2)^{\beta_1} & q_i(e_2)^{\beta_2} & \cdots & q_i(e_2)^{\beta_{N_s}} \\ \vdots & \vdots & \ddots & \vdots \\ q_i(e_{2^n})^{\beta_1} & q_i(e_{2^n})^{\beta_2} & \cdots & q_i(e_{2^n})^{\beta_{N_s}} \end{pmatrix} \ ,$$

where $\{\beta_i\} = \{\beta \mid 1 \leqslant \mathtt{HW}(\beta) \leqslant s\} \subseteq [0; 2^n - 1]$ (*i.e.* the $\beta_i$ are the powers with non-zero coefficients in a degree-$s$ polynomial). On the other hand, $A_{\ell_i}$ is defined as the $2^n \times n$ matrix:

$$A_{\ell_0} = \begin{pmatrix} e_1^{\alpha_1} & e_1^{\alpha_2} & \cdots & e_1^{\alpha_n} \\ e_2^{\alpha_1} & e_2^{\alpha_2} & \cdots & e_2^{\alpha_n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{2^n}^{\alpha_1} & e_{2^n}^{\alpha_2} & \cdots & e_{2^n}^{\alpha_n} \end{pmatrix} \quad \text{and} \quad A_{\ell_i} = \begin{pmatrix} g_i(e_1)^{\alpha_1} & g_i(e_1)^{\alpha_2} & \cdots & g_i(e_1)^{\alpha_n} \\ g_i(e_2)^{\alpha_1} & g_i(e_2)^{\alpha_2} & \cdots & g_i(e_2)^{\alpha_n} \\ \vdots & \vdots & \ddots & \vdots \\ g_i(e_{2^n})^{\alpha_1} & g_i(e_{2^n})^{\alpha_2} & \cdots & g_i(e_{2^n})^{\alpha_n} \end{pmatrix}$$

for $i \geqslant 1$, where $\{\alpha_i\} = \{2^i \mid 0 \leqslant i \leqslant n-1\}$ (*i.e.* the $\alpha_i$ are the powers with non-zero coefficients in a linearized polynomial).

System (6) has $2^n$ equations and $t \cdot N_s + (r+1)\,n$ unknowns. Such a system admits a solution for every choice of $h(x)$ if and only if its rank is $2^n$, which implies the following inequality as necessary condition:

$$t \cdot N_s + (r+1)\,n \geqslant 2^n \quad \Leftrightarrow \quad t \geqslant \frac{2^n - (r+1)\,n}{N_s} \ . \tag{7}$$

In other words our method requires at least $\big(2^n - (r+1)\,n\big)/N_s$ secure evaluations of degree-$s$ polynomials. Another necessary condition is that the algebraic degree of the $p_i \circ q_i$ reaches $n$, that is $r$ verifies $s^{r+1} \geqslant n$. This constraint becomes $s^{r+1} \geqslant n-1$ if we only focus on bijective functions (since their algebraic degree is at most $n-1$).

*Remark 2.* We stress that once a full-rank system has been found for some given parameters $r$ and $t$, it can be applied to get a decomposition for *every* $n$-bit nonlinear function (the considered function being the target vector of the system). Note however that for some specific function it might be possible to find a decomposition involving less than $r + t$ degree-$s$ polynomials. For instance, the 4-bit s-box of PRESENT can be decomposed into 2 quadratic functions [21], whereas we need 3 quadratic functions to decompose any 4-bit s-box.

**Experimental results.** We provide hereafter some experimental results for our algebraic decomposition method. We experimented our method for $n \in [\![4, 8]\!]$ and $s \in \{2, 3\}$. The following table summarizes the best results that we obtained and compares them to the lower bound resulting from the above constraints.

| | $n = 4$ | $n = 5$ | $n = 6$ | $n = 7$ | $n = 8$ |
|---|---|---|---|---|---|
| #eval-2 (achieved) | **3** | **4** | **5** | **8** | **11** |
| #eval-2 (lower bound) | 2 | 4 | 5 | 6 | 9 |
| #eval-3 (achieved) | **2** | **3** | **3** | **4** | **4** |
| #eval-3 (lower bound) | 2 | 2 | 3 | 3 | 4 |

Note that our method can be generalized to involve the evaluation of functions with different (low) algebraic degrees. In particular, in our experiments, we considered the hybrid case where the $g_i$ are of degree $s_1$ and the $p_i$ are of degree $s_2$. In that case, the constraint on $r$ becomes $s_1^r \cdot s_2 \geqslant n$ (resp. $s_1^r \cdot s_2 \geqslant n - 1$ for bijective functions), and the constraint on $t$ remains as in (7) with $s_2$ instead of $s$. The following table summarizes the best results for the hybrid case $(s_1, s_2) = (2, 3)$. This case was always more favorable than the case $(s_1, s_2) = (3, 2)$.

| | $n = 4$ | $n = 5$ | $n = 6$ | $n = 7$ | $n = 8$ |
|---|---|---|---|---|---|
| #eval-2 + #eval-3 (achieved) | **1+1** | **2+1** | **1+2** | **2+2** | **2+3** |
| #eval-2 + #eval-3 (lower bound) | $1 + 1$ | $1 + 1$ | $1 + 2$ | $2 + 2$ | $2 + 3$ |

The following table gives the exact parameters $(s_1, s_2)$, $r$, and $t$, that we achieved for every $n \in [\![4, 8]\!]$. Note that the entries $4^*$, $5^*$, and $7^*$ stand for bijective functions of size $n \in \{4, 5, 7\}$, which enjoy more efficient decompositions. For the other considered values of $n$, the bijective property did not enable any improvement.

| $n$ | #eval-2 | #eval-3 | $(s_1, s_2)$ | $r$ | $t$ |
|---|---|---|---|---|---|
| 4 | 3 | - | $(2,2)$ | 1 | 2 |
|  | 3 | - | $(2,2)$ | 2 | 1 |
|  | 1 | 1 | $(2,3)$ | 1 | 1 |
|  | - | 2 | $(3,3)$ | 1 | 1 |
| 4* | - | 1 | $(1,3)$ | 0 | 1 |
| 5 | 4 | - | $(2,2)$ | 2 | 2 |
|  | 2 | 1 | $(2,3)$ | 2 | 1 |
|  | - | 3 | $(3,3)$ | 1 | 2 |
| 5* | 3 | - | $(2,2)$ | 1 | 2 |
| 6 | 5 | - | $(2,2)$ | 2 | 3 |
|  | 1 | 2 | $(2,3)$ | 1 | 2 |
|  | - | 3 | $(3,3)$ | 1 | 2 |
| 7 | 8 | - | $(2,2)$ | 2 | 6 |
|  | 2 | 2 | $(2,3)$ | 2 | 2 |
|  | - | 4 | $(3,3)$ | 1 | 3 |
|  | - | 4 | $(3,3)$ | 2 | 2 |
| 7* | 1 | 2 | $(2,3)$ | 1 | 2 |
| 8 | 11 | – | $(2,2)$ | 2 | 9 |
|  | 11 | – | $(2,2)$ | 3 | 8 |
|  | 2 | 3 | $(2,3)$ | 2 | 3 |
|  | - | 4 | $(3,3)$ | 1 | 3 |

*Remark 3.* For the case $n = 4$, it was shown in [3] that every cubic bijective function in $\mathcal{B}_{4,4}$ can be either decomposed as $h(\cdot) = f_1 \circ f_2(\cdot)$ or as $h(\cdot) = f_1 \circ f_2 \circ f_3(\cdot)$, where the $f_i$ are quadratic functions, if and only if it belongs to the alternating group of permutations in $\mathcal{B}_{4,4}$. It was then shown in [17] that the so-called *optimal s-boxes* in $\mathcal{B}_{4,4}$ can be decomposed as $h(\cdot) = f_1(\cdot) + f_2 \circ f_3(\cdot)$. The authors also suggested to use a decomposition of the form $h(\cdot) = f_1(\cdot) + f_2 \circ f_3(\cdot) + f_4 \circ f_5(\cdot) + \ldots$ for other functions in $\mathcal{B}_{4,4}$. Our results demonstrate that *every* function in $\mathcal{B}_{4,4}$ can be decomposed using 3 quadratic functions plus some linear functions. Moreover, we know from [3] that there exist functions in $\mathcal{B}_{4,4}$ that cannot be decomposed using only two quadratic functions. This show the optimality of our method for the case $n = 4$ and $s = 2$. This also suggests that the lower bound (7) might not be tight.

## 4 Reducing the Probing Order down to the Algebraic Degree

In this section we start by showing that arbitrary $t$-probing security can always be reduced to $s$-probing security where $s$ is the algebraic degree of the function to protect. Specifically, we give a method to construct a $t$-probing secure processing of a degree-$s$ function from its $s$-probing secure processing. Then, we apply our method for the case $s = 2$, where a simple 2-probing secure processing can be turned into an efficient $t$-probing secure evaluation of any (algebraically) quadratic function.

### 4.1 General Case

Let $n$ and $m$ be two positive integers such that $m \leqslant n$ and let $h \in \mathcal{B}_{n,m}$ be the vectorial function whose processing must be secured at the order $t = d - 1$ for some $d$. By definition of $\varphi_h^{(s)}$, for every tuple $(a_1, \cdots, a_s) \in (\mathbb{F}_2^n)^s$ we have:

$$h\Big(\sum_{i=1}^{s} a_i\Big) = \varphi_h^{(s)}(a_1, a_2, \ldots, a_s) + \sum_{I \subsetneq [\![1,s]\!]} h\Big(\sum_{i \in I} a_i\Big) \ . \tag{8}$$

Iterating (8) we obtain the following theorem where, by convention, $h(\sum_{i\in\emptyset} a_i)$ equals $h(0)$. The proof is given in the full version of the paper.

**Theorem 1.** *Let $h \in \mathcal{B}_{n,m}$ be a vectorial function of algebraic degree at most $s$. Then, for every $d \geqslant s$ we have:*

$$h\Big(\sum_{i=1}^{d} a_i\Big) = \sum_{1\leqslant i_1<\cdots<i_s\leqslant d} \varphi_h^{(s)}(a_{i_1}, \ldots, a_{i_s}) + \sum_{j=0}^{s-1} \eta_{d,s}(j) \sum_{\substack{I \subseteq [\![1,d]\!] \\ |I|=j}} h\Big(\sum_{i \in I} a_i\Big) \ ,$$

*where $\eta_{d,s}(j) = \binom{d-j-1}{s-j-1} \bmod 2$ for every $j \leqslant s-1$.*

From Theorem 1 we get the following corollary.

**Corollary 1.** *Let $h \in \mathcal{B}_{n,m}$ be a vectorial function of algebraic degree at most $s$. Then, for every $d > s$ we have:*

$$h\Big(\textstyle\sum_{i=1}^{d} a_i\Big) = \sum_{j=0}^{s} \mu_{d,s}(j) \sum_{\substack{I \subseteq [\![1,d]\!] \\ |I|=j}} h\Big(\sum_{i \in I} a_i\Big) \ ,$$

*where $\mu_{d,s}(j) = \binom{d-j-1}{s-j} \bmod 2$ for every $j \leqslant s$.*

Corollary 1 states that, for any $d$, the evaluation of a degree-$s$ function $h \in \mathcal{B}_{n,m}$ on the sum of $d$ shares can be expressed as several evaluations of $h$ on sums $\sum_{i\in I} a_i$ with $|I| \leqslant s$. We can then reduce a $(d-1)$-probing secure evaluation of $h$ to several $(j-1)$-probing secure evaluations of $h$ where $j = |I| \leqslant s$. Doing so, each evaluation takes $j = |I|$ shares $(a_i)_{i\in I}$ and computes a $j$-sharing of $h(\sum_{i\in I} a_i)$. Afterwards, one needs a secure scheme to combine the obtained shares of all the $h(\sum_{i\in I} a_i)$, with $I \subseteq [\![1, d]\!]$ such that $|I| \leqslant s$ and $\mu_{d,s}(|I|) = 1$, into a $d$-sharing of $h(a)$.

The overall process is summarized in the following algorithm, where SecureEval is a primitive that performs a $(j-1)$-probing secure evaluation of $h$ on a $j$-sharing input for any $j \leqslant s$, and where SharingCompress is a primitive that on input $(x_i)_{i\leqslant[\![1,k]\!]}$ produces a $d$-sharing of $\sum_{i=1}^{k} x_i$ for any $k \leqslant d$. The full version of this paper includes the description of such SharingCompress algorithm which achieves perfect $t$-probing security, and provide a security proof for the overall method.

**Algorithm 1 :** Secure evaluation of a degree-$s$ function

```
    Input  : a $d$-sharing $(x_1, x_2, \cdots, x_d)$ of $x \in \mathbb{F}_{2^n}$
    Output: a $d$-sharing $(y_1, y_2, \cdots, y_d)$ of $y = h(x)$
1 for $I \subseteq [\![1, d]\!]$ with $|I| \leqslant s$ and $\mu_{d,s}(|I|) = 1$ do
2     $(r_{I,k})_{k \leqslant |I|} \leftarrow \mathsf{SecureEval}(h, (x_i)_{i \in I})$
3 $(y_1, y_2, \ldots, y_d) \leftarrow \mathsf{SharingCompress}\big((r_{I,k})_{k \leqslant |I|, I \subseteq [\![1,d]\!], \mu_{d,s}(|I|)=1}\big)$
4 return $(y_0, y_1, \ldots, y_d)$
```

**Complexity.** For every $s$ and every $d > s$, the term $\mu_{d,s}(j)$ always equals 1 when $j = s$. On the other hand, whenever $d \equiv s \bmod 2^\ell$ with $\ell = \lfloor \log_2 s \rfloor + 1$, the term $\mu_{d,s}(j)$ equals 0 for every $j < s$. For the sake of efficiency, we shall assume that such a value of $d$ is always chosen for our method.[6] Under this assumption, the complexity of Algorithm 1 is of $\binom{d}{s}$ calls to $\mathsf{SecureEval}$ with $s$ shares and one call to $\mathsf{SharingCompress}$ from $k$ shares to $d$ shares where $k = s\binom{d}{s}$. From the complexity of the sharing compression method, we obtain the following operation count (where "#add" and "#rand" respectively denote the number of additions and the number of sampled random values in the sharing compression).

| | #SecureEval | #add | #rand |
|---|---|---|---|
| Exact count | $\binom{d}{s}$ | $\big(s\binom{d}{s} - d\big)(d+1)$ | $\frac{1}{2}\big(s\binom{d}{s} - d\big)(d-1)$ |
| Approximation | $\big(\frac{1}{s!}\big)d^s$ | $\big(\frac{1}{(s-1)!}\big)d^{s+1}$ | $\big(\frac{1}{2(s-1)!}\big)d^{s+1}$ |

## 4.2 Quadratic Case

For degree-2 (aka quadratic) functions $h$, it may be observed that $\varphi_h^{(2)}(x_i, x_j)$ equals $h(x_i + x_j + r) + h(x_i + r) + h(x_j + r) + h(r)$ whatever $(x_i, x_j, r) \in \big(\mathbb{F}_{2^n}\big)^3$ (this holds since $\varphi_h^{(3)}(x_i, x_j, r) = 0$ for quadratic functions). This observation and Theorem 1 imply the following equality for any integer $d \geqslant s$ and any $r_{i,j}$ values in $\mathbb{F}_{2^n}$:

$$h\Big(\sum_{i \in [\![1,d]\!]} x_i\Big) = \sum_{1 \leqslant i < j \leqslant d} \big(h(x_i + x_j + r_{i,j}) + h(x_j + r_{i,j}) + h(x_i + r_{i,j}) + h(r_{i,j})\big) + \sum_{i \in [\![1,d]\!]} h(x_i) + \big((d+1) \bmod 2\big) \cdot h(0) \ . \quad (9)$$

Equality (9) shows that the evaluation of a quadratic function in a sum of $d$ shares can be split into a sum of terms depending on at most two shares $x_i$ and $x_j$. In this particular case it is then possible to use an improved sharing compression inspired from ISW, which gives[7] Algorithm 2.

[6] This is a weak assumption for low algebraic degrees. For instance $s \leqslant 3$ gives $d \equiv s \bmod 4$, $s \leqslant 7$ gives $d \equiv s \bmod 8$, etc. Note that the complexity comparison in Section 7 does not take this assumption into account.

[7] Note that in Step 4 the additions must be computed from left to right in order to ensure the probing security.

```
Algorithm 2 : Secure evaluation of a quadratic function
   Input  : the $d$-sharing $(x_1, x_2, \dots, x_d)$ of $x$ in $\mathbb{F}_{2^n}$
   Output: a $d$-sharing $(y_1, y_2, \dots, y_d)$ of $y = h(x)$
1  for $i = 1$ to $d$ do
2      for $j = i + 1$ to $d$ do
3          $r_{i,j} \xleftarrow{\$} \mathbb{F}_{2^n}$; $r'_{i,j} \xleftarrow{\$} \mathbb{F}_{2^n}$
4          $r_{j,i} \leftarrow r_{i,j} + h(x_i + r'_{i,j}) + h(x_j + r'_{i,j}) + h((x_i + r'_{i,j}) + x_j) + h(r'_{i,j})$
5  for $i = 1$ to $d$ do
6      $y_i \leftarrow h(x_i)$
7      for $j = 1$ to $d$, $j \neq i$ do
8          $y_i \leftarrow y_i + r_{i,j}$
9  if $d$ is even then $y_1 = y_1 + h(0)$
10 return $(y_1, y_2, \dots, y_d)$
```

This algorithm is actually already known from [8]. However the authors only suggest to use it for the secure evaluation of a multiplication of the form $x \cdot \ell(x)$ where $\ell$ is a degree-1 function (linear or affine). We show here that this algorithm can actually be used to securely compute *any* degree-2 function. The only difference is that one must add $h(0)$ to one output share whenever $d$ is even (this term does not appear in [8] since by definition of $h : x \mapsto x \cdot \ell(x)$ one always get $h(0) = 0$). The probing security of Algorithm 2 holds from the security proof provided in [8].

**Complexity.** The following table summarizes the complexity of Algorithm 2 in terms of additions, evaluations of $h$, and sampled random values (we consider the worst case of $d$ being even). For comparison, we also give the complexity of the ISW scheme for a single multiplication.

| | # add | # eval$_h$ | # mult | # rand |
|---|---|---|---|---|
| Algorithm 2 | $\frac{9}{2}\,d(d-1)+1$ | $d(2d-1)$ | - | $d(d-1)$ |
| ISW multiplication | $2d(d-1)$ | - | $d^2$ | $\frac{1}{2}d(d-1)$ |

As explained in Section 2.3, for some values of $n$, a lookup table of $2^n$ entries might be affordable while a lookup table of $2^{2n}$ entries is not (typically for $n = 8$ giving 256 bytes *vs.* 64 kilobytes). In such a situation, the cost of one evaluation of $h$ is expected to be significantly lower than the cost of a multiplication. We hence expect Algorithm 2 to be more efficient than the ISW scheme. Moreover, as shown above, Algorithm 2 can be used to securely evaluate a degree-2 function with a polynomial representation possibly involving *many* multiplications, whereas the ISW scheme securely evaluates a *single* multiplication.

## 5 Another Method to Secure Low-Degree Functions

In this section we introduce another new scheme to secure the evaluation of any function $h \in \mathcal{B}_{n,m}$ of given algebraic degree $s$. The method only involves additions and evaluations of $h$ (that may be tabulated depending on the context). As the previous method, its complexity is exponential in $s$ which makes it suited for low-degree functions only. We start by introducing the core ideas of our method with the simple case of a 2-probing secure evaluation (*i.e.* taking a 3-shared input) of a degree-2 function. Then, we show how to extend the approach to achieve arbitrary probing security. The study is completed in the full version of this paper where we generalize our result to functions of any algebraic degree.

### 5.1 Two-Probing Security for Quadratic Functions

Let $h \in \mathcal{B}_{n,m}$ be a degree-2 function. We present hereafter a new method to securely construct a 3-sharing $(y_1, y_2, y_3)$ of $y = h(x)$ from a 3-sharing $(x_1, x_2, x_3)$ of $x$. Since $h$ is quadratic, its third-order derivatives are null (see Definition 1) which implies the following equality for every $x \in \mathbb{F}_{2^n}$ and every triplet $(r_1, r_2, r_3) \in \mathbb{F}_{2^n}^3$:

$$h(x) = \sum_{1 \leqslant i \leqslant 3} h(x + r_i) + \sum_{1 \leqslant i < j \leqslant 3} h(x + r_i + r_j) + h(x + r_1 + r_2 + r_3) \ , \quad (10)$$

or equivalently $h(x) = \sum_{i=1}^{7} h(x + e^{(i)})$, where $e^{(i)}$ denotes the scalar product $\omega_i \cdot (r_1, r_2, r_3)$ with $\omega_i$ being the binary representation of the index $i$ (*e.g.* $\omega_3 = (0, 1, 1)$). We shall say in the following that the family $(e^{(i)})_{i \in [\![1,7]\!]}$ is 3-*spanned* from $(r_1, r_2, r_3)$. Replacing $x$ by the sum of its shares then leads to:

$$h(x_1 + x_2 + x_3) = \sum_{i=1}^{7} h(x_1 + x_2 + x_3 + e^{(i)}) \ . \quad (11)$$

It may be checked that the tuple $(h_i)_{i \in [\![1,7]\!]} = \big(h(x + e^{(i)})\big)_{i \in [\![1,7]\!]}$ is a $(7, 3)$-sharing of $h(x)$ (this holds since the rank of $(e^{(i)})_{i \in [\![1,7]\!]}$ is at most 3). However, a direct evaluation of the $h_i$ would yield an obvious second-order flaw. Indeed, for every $i \in \{1, 2, 3\}$, the evaluation of at least one of the terms in (11) implies the computation of $x + r_i$ which can be combined with $r_i$ to recover $x$. A natural solution to avoid such a second-order flaw is to split the processing of each $x + e^{(i)}$ into several parts, which actually amounts to randomly split each $e^{(i)}$ into 3 shares $e_1^{(i)}$, $e_2^{(i)}$, $e_3^{(i)}$. This leads to the following expression:

$$h(x) = \sum_{i=1}^{7} h\Big((x_1 + e_1^{(i)}) + (x_2 + e_2^{(i)}) + (x_3 + e_3^{(i)})\Big) \ . \quad (12)$$

It then just remains to securely turn the $(7, 3)$-sharing $(h_i)_{i \in [\![1,7]\!]}$ into a 3-sharing $(y_i)_{i \in [\![1,3]\!]}$. This is done by using the following sharing compression procedure specific to the case 7 to 3:

1. $(m_1, m_2, m_3) \overset{\$}{\leftarrow} \mathbb{F}_{2^n}^q$
2. $y_1 \leftarrow (h_1 + m_1) + (h_4 + m_2) + h_7$
3. $y_2 \leftarrow (h_2 + m_1) + (h_5 + m_3)$
4. $y_3 \leftarrow (h_3 + m_2) + (h_6 + m_3)$

## 5.2 Arbitrary Probing Security for Quadratic Functions

The idea in previous section can be extended to any security order $d$ by starting from the following equation which generalizes (12):

$$h(x) = \sum_{i_1=1}^{7} \sum_{i_2=1}^{7} \cdots \sum_{i_t=1}^{7} h\Big( \sum_{j=1}^{d} x_j + e_j^{(i_1)} + e_j^{(i_1,i_2)} + \cdots + e_j^{(i_1,i_2,\cdots,i_t)} \Big) \ , \quad (13)$$

where for every $(j,q) \in [\![1,d]\!] \times [\![1,t]\!]$ and every tuple $(i_1,\cdots,i_{q-1}) \in [\![1,7]\!]^{q-1}$ the family $(e_j^{(i_1,\cdots,i_{q-1},i_q)})_{i_q \in [\![1,7]\!]}$ is 3-spanned from fresh random values. Let us denote by $h^{(i_1,i_2,\cdots,i_t)}$, with $(i_1,\cdots,i_t) \in [\![1,7]\!]^t$, the terms in the right-hand sum in (13). It may be checked that the family $\{h^{(i_1,i_2,\cdots,i_t)} \mid (i_1,i_2,\cdots,i_t) \in [\![1,7]\!]^t\}$ forms a $(7^t, 3^t)$-sharing of $h(x)$. To turn the latter into a $3^t$-sharing (which leaves us with a $d$-sharing of $h(x)$ taking $t = \log_3(d)$) the sharing compression procedure is recursively applied. This leads to the following recursive algorithm.

**Algorithm 3 :** TreeExplore

```
    Input: a $d$-sharing $(z_1, z_2, \ldots, z_d)$ of $z \in \mathbb{F}_{2^n}$, a depth parameter $k$
    Output: a $3^k$-sharing of $h(z)$
1   if $k = 0$ then
2       return $h(z_1 + z_2 + \cdots + z_d)$
3   for $j = 1$ to $d$ do
4       $(e_j^{(1)}, e_j^{(2)}, \ldots, e_j^{(7)}) \leftarrow$ RandSpan(3)
5   for $i = 1$ to 7 do
6       $(h_{(w)}^{(i)})_{w \in \{1,2,3\}^{k-1}} \leftarrow$ TreeExplore$(z_1 + e_1^{(i)}, z_2 + e_2^{(i)}, \ldots, z_d + e_d^{(i)}, k-1)$
7   for $w$ in $\{1,2,3\}^{k-1}$ do
8       $(h_{(w||1)}, h_{(w||2)}, h_{(w||3)}) \leftarrow$ SharingCompress$(h_{(w)}^{(1)}, h_{(w)}^{(2)}, \ldots, h_{(w)}^{(7)})$
9   return $(h_{(w)})_{w \in \{1,2,3\}^k}$
```

A detailed and didactic description of the method together with a proof of probing security is provided in the full version of this paper.

**Complexity.** The following table summarizes the complexity of Algorithm 3 in terms of additions, evaluation of $h$, and random generation over $\mathbb{F}_{2^n}$.

| | #add | #eval$_h$ | # rand |
|---|---|---|---|
| Exact count | $3d \cdot 7^t - 2d + \frac{5}{2}(7^t - 3^t)$ | $7^t$ | $\frac{d}{2}7^t - \frac{d}{2} + \frac{3(7^t - 3^t)}{4}$ |
| Approximation | $3d^{2.77}$ | $d^{1.77}$ | $\frac{1}{2}d^{2.77}$ |

## 6 Adapting the CRV Method to Low Algebraic Degrees

This section proposes an adaptation of Coron-Roy-Vivek's method (CRV) with improved complexity for functions with given (low) algebraic degree. We start by recalling the CRV method [9].

### 6.1 The CRV Method

In the following, we shall view functions over $\mathcal{B}_{n,m}$ as polynomials over $\mathbb{F}_{2^n}[x]$. Let $h(x) = \sum_{j=0}^{2^n-1} a_j x^j$ be the function that must be securely evaluated. To find a representation of $h(x)$ that minimizes the number of nonlinear multiplications, CRV starts by building the union set $\mathcal{L} = \bigcup_{i=1}^{\ell} \mathcal{C}_{\alpha_i}$ where $\mathcal{C}_{\alpha_i}$ is the cyclotomic class of $\alpha_i$ defined as $\mathcal{C}_{\alpha_i} = \{2^j \cdot \alpha_i \bmod (2^n - 1)\ ;\ j \in [\![0, n-1]\!]\}$ such that $\alpha_1 = 0$, $\alpha_2 = 1$, and $\alpha_{i+1} \in \bigcup_{j=1}^{i} \mathcal{C}_{\alpha_j} + \bigcup_{j=1}^{i} \mathcal{C}_{\alpha_j}$ for every $i$. The elements in $\{x^\alpha;\ \alpha \in \mathcal{L}\}$ can then be processed with only $\ell - 2$ nonlinear multiplications.[8] The set $\mathcal{L}$ must satisfy the constraint $\mathcal{L} + \mathcal{L} = [\![0, 2^n - 1]\!]$ and, if possible, the $\ell$ classes $\mathcal{C}_{\alpha_i}$ in $\mathcal{L}$ are chosen such that $|\mathcal{C}_{\alpha_i}| = n$.

Let $\mathcal{P} \subseteq \mathbb{F}_{2^n}[x]$ be the subspace spanned by the monomials $x^\alpha$ with $\alpha \in \mathcal{L}$. The second step of CRV consists in randomly generating $t - 1$ polynomials $q_i(x) \in \mathcal{P}$ and in searching for $t$ polynomials $p_i(x) \in \mathcal{P}$ such that

$$h(x) = \sum_{i=1}^{t-1} p_i(x) \times q_i(x) + p_t(x) \ . \tag{14}$$

This gives a linear system with $2^n$ equations (one for each $x \in \mathbb{F}_{2^n}$) and $t \times |\mathcal{L}|$ unknowns (the coefficients of the $p_i$). Such a system admits a solution for every choice of $h$ if its rank is $2^n$, which leads to the necessary condition $t \times |\mathcal{L}| \geqslant 2^n$. Finding such a solution provides a method to evaluate $h$ involving $\ell + t - 3$ multiplications: $\ell - 2$ multiplications to generate the monomial $(x^j)_{j \in \mathcal{L}}$, from which $p_i(x)$ and $q_i(x)$ are computed as linear combinations for every $i \leqslant t$, and $t - 1$ multiplications to evaluate (14). In order to optimize the number of linear operations, the polynomials $p_i$ can be represented as $p_i(x) = \sum_{\alpha_j \in \mathcal{L}} \ell_{i,j}(x^{\alpha_j})$ where the $\ell_{i,j}$ are linearized polynomials (*i.e.* polynomials of algebraic degree 1) that might be tabulated.

**Complexity.** Assuming that all the cyclotomic classes in $\mathcal{L}$ except $\mathcal{C}_0$ have maximum size $n$ (*i.e.* $|\mathcal{L}| = 1 + n \times (\ell - 1)$) and that the lower bound $t \times |\mathcal{L}| \geqslant 2^n$ is reached, it is argued in [9] that the complexity of CRV is minimized for $t = \lceil \sqrt{2^n/n} \rceil$ and $\ell = \lceil \sqrt{2^n/n} - 1/n + 1 \rceil$. Moreover, it is empirically shown in [9] that these lower bounds are often achieved for $n \leqslant 12$. Using ISW to secure nonlinear multiplications we get the following complexity (where #eval LP denotes the number of evaluations of a linearized polynomial):

[8] For $\alpha_1 = 0$ and $\alpha_2 = 1$ the building requires no nonlinear multiplication.

| #add | #eval LP | #rand | #mult |
|---|---|---|---|
| $2d^2(t+\ell-3)+d(t\ell-2t-3\ell+5)$ | $d\ell t$ | $\frac{d(d-1)(t+\ell-3)}{2}$ | $d^2(t+\ell-3)$ |

*Remark 4.* Some of the $\ell-2$ nonlinear multiplications used to generate the set of powers $\{x^\alpha; \alpha \in \mathcal{L}\}$ might take the form $x^{\alpha_i} \cdot (x^{\alpha_i})^{2^j}$ with $\alpha_i \in \mathcal{L}$. In that case, the CPRR scheme (Algorithm 2) may be preferred to ISW. Indeed, as discussed in Section 4.2, this algorithm may be more efficient than ISW when $x \mapsto x \cdot x^{2^j}$ can be tabulated whereas $(x, y) \mapsto x \cdot y$ cannot. The same observation applies for the tweaked CRV method proposed in the next section for low-degree functions. In the complexity comparisons discussed in Section 7, this optimization is used (we have actually observed that it was always possible to entirely build $\mathcal{L}$ based on $\ell-2$ multiplications of the form $x \mapsto x \cdot x^{2^j}$).

## 6.2 The CRV Method for Degree-$s$ Functions

We propose here an adaptation of the CRV method for functions with (low) algebraic degree $s$. For the generation of $\mathcal{L}$, the constraint becomes $\mathcal{L} + \mathcal{L} = \{\alpha \in [\![0, 2^n-1]\!]\ ;\ \mathtt{HW}(\alpha) \leqslant s\}$. When $s$ is even, we impose the additional condition that every cyclotomic class $\mathcal{C}_\alpha \subseteq \mathcal{L}$ verifies $\mathtt{HW}(\alpha) \leqslant \frac{s}{2}$ (the odd case is addressed hereafter). Then, as in the original method, we randomly generate $t-1$ polynomials $q_i(x)$ in the subspace $\mathcal{P}$ (*i.e.* the subspace of polynomials spanned by the monomials $x^j$ with $j \in \mathcal{L}$), and we try to solve a linear system obtained from (14). The difference is that the obtained system is of rank at most $\sum_{r=0}^{s}\binom{n}{r}$, *i.e.* the maximum number of non-null coefficients in a degree-$s$ function. Here again if this maximal rank is achieved, then the system has a solution for every target vector *i.e.* for every degree-$s$ function $h(x)$. The necessary condition to get a full-rank system then becomes $t \times |\mathcal{L}| \geqslant \sum_{r=0}^{s}\binom{n}{r}$. Assuming that the cyclotomic classes in $\mathcal{L}$ except $\mathcal{C}_0$ have maximum size $n$, this gives

$$t \geqslant \frac{\sum_{r=0}^{s}\binom{n}{r}}{n(\ell-1)+1} .$$

If this bound is reached, the number $t+\ell-3$ of nonlinear multiplications is minimized by taking $t = (\ell-1) = (\sum_{r=0}^{s}\binom{n}{r}/n)^{1/2}$, and we get $t+\ell-3 = 2(\sum_{r=0}^{s}\binom{n}{r}/n)^{1/2} - 2$. When $s$ is odd, the method is similar but $\mathcal{L}$ must contain some cyclotomic classes $\mathcal{C}_\alpha$ such that $\mathtt{HW}(\alpha) \leqslant \frac{s+1}{2}$ and the $q_i$ are constructed from powers $x^\alpha$ such that $\mathtt{HW}(\alpha) \leqslant \frac{s-1}{2}$ (this ensures that the algebraic degree of $p_i(x) \cdot q_i(x)$ is at most $s$).

The complexity for this method is the same as for CRV (see table in the previous section), but for low-degree functions, the obtained parameters $(t, \ell)$ are significantly smaller. The obtained number of nonlinear multiplications are compared in the following table.

| | $n=4$ | $n=5$ | $n=6$ | $n=7$ | $n=8$ | $n=9$ | $n=10$ |
|---|---|---|---|---|---|---|---|
| Original CRV | 2 | 4 | 5 | 7 | 10 | 14 | 19 |
| CRV for $s=2$ | 2 | 2 | 2 | 3 | 3 | 3 | 3 |
| CRV for $s=3$ | 2 | 3 | 4 | 5 | 5 | 6 | 7 |

## 7 Comparison

In this section, we first study the practical complexity of the methods introduced in Sections 4, 5, and 6 to secure functions of low algebraic degrees (specifically of degrees $s = 2$ and $s = 3$). Then, we compare our algebraic decomposition method exposed in Section 3 with the CRV method (which is the best known alternative). The comparisons are done on specific examples where the dimension $n$ fits with classical symmetric ciphers' s-boxes (*i.e.* $n \in \{4, 8\}$) and the number $d$ of shares ranges over $[\![2, 9]\!]$. An asymptotic analysis w.r.t parameters $n$, $d$, and $s$, is further provided in the full version of the paper.

**Low-Degree Functions.** For the case $s = 2$, we compare Algorithm 2 (generalization of the CPRR scheme – see Section 4), Algorithm 3 (aka `TreeExplore` – see Section 5), and the tweaked CRV method for low-degree functions (aka `CRV-LD` – see Section 6). For the case $s = 3$, our first method (Algorithm 1) must be combined with a third-order secure evaluation method (primitive SecureEval) of degree-3 functions. For such a purpose, we either use Algorithm 3 (`TreeExplore`) or the tweaked CRV method (`CRV-LD`). The exact operations counts for these methods are plotted in Figures 1 and 2.[9] In our comparisons, we assumed that an addition, a table lookup and a random generation of $n$ bits have the same cost 1.[10] For the multiplication, we considered various possible costs $C$. The case $C = 1$ corresponds to a device where the multiplication of two elements in $\mathbb{F}_{2^n}$ can be precomputed and stored in a table, hence taking $n2^{2n}$ bits of memory. When this is not possible (in a constraint environment and/or for too large values of $n$), the multiplication must be implemented and its cost essentially depends on the device architecture.[11] We believe to encompass most realistic scenarios by considering $C \in \{5, 10, 20\}$. Note that for the case $s = 3$, we used the improved CRV method based on CPRR for the generation of powers as suggested in Remark 4. We did not use it for $s = 2$ since a CPRR multiplication is similar to one call to Algorithm 2.

**Case ($\mathbf{s = 2}$).** Figure 1 clearly illustrates the superiority of Algorithm 2 for the secure evaluation of degree-2 functions at any order. The ranking between our second method (Algorithm 3 aka `TreeExplore`) and the tweaked CRV method (`CRV-LD`) depends on the sharing order $d$ and the multiplication cost ratio $C$. For high values of $C$ (*i.e.* when the multiplication cannot be tabulated), `TreeExplore` outperforms the tweaked CRV method. It is particularly interesting for a sharing order $d$ lower than 4. However, due to its tree structure of depth $\log_3(d)$, it becomes slower as soon as $d$ reaches 4. Moreover for higher

[9] Note that when $d \leqslant s$, Algorithm 1 is not specified and therefore cannot be applied.

[10] In the context of side-channel countermeasures, generating $n$ random bits usually amounts to read a $n$-bit value in a TRNG register.

[11] In [12], the authors explain that the processing of a field multiplication with the `CPU` instructions set requires between 20 and 40 cycles and they give some examples of implementations.

values of $d$, it would not be competitive since its asymptotic complexity is more than quadratic in the sharing order.

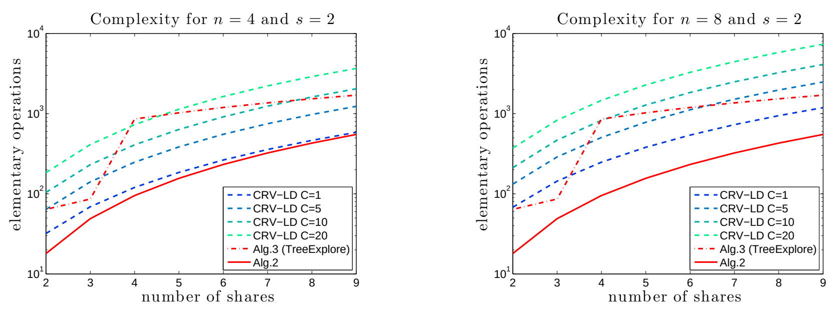

**Fig. 1.** Secure evaluation of quadratic functions

**Case** $(\mathbf{s = 3})$. Figure 2 show the superiority of the tweaked CRV method for sharing orders greater than 3 and even for costly multiplications (*i.e.* $C = 20$). For small sharing orders $d \in \{2, 3\}$, Algorithm 3 (TreeExplore) is competitive (for the same reasons as for the degree-2 case). It appears that the use of our first method (Algorithm 1) for lowering the sharing order down to the algebraic degree is never interesting in this setting. We however think that this is not a dead-end point for this method and that the ideas used to obtain Algorithm 2 from the general description of the method could be used to get an improved version for the cubic case as well. We let this question open for further research on this subject.

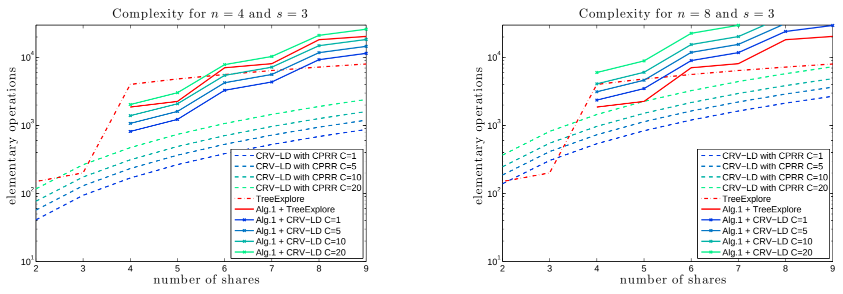

**Fig. 2.** Secure evaluation of cubic functions

**High-Degree Functions.** We now consider the algebraic decomposition method described in Section 3 and compare it to the CRV method. The following table summarizes the number $(\ell - 2) + (t - 1)$ of secure nonlinear multiplications involved in CRV, as well as the numbers $r$ and $t$ of secure evaluations of degree-$s_1$ functions and degree-$s_2$ functions in the algebraic decomposition method.

| | This Paper | | | | | CRV | | |
|---|---|---|---|---|---|---|---|---|
| | #eval-2 | #eval-3 | $(s_1, s_2)$ | $r$ | $t$ | #mult | $\ell - 2$ | $t - 1$ |
| | 3 | - | $(2, 2)$ | 1 | 2 | | | |
| $n = 4$ | 1 | 1 | $(2, 3)$ | 1 | 1 | 2 | 1 | 1 |
| | - | 2 | $(3, 3)$ | 1 | 1 | | | |
| | 5 | - | $(2, 2)$ | 2 | 3 | | | |
| $n = 6$ | 1 | 2 | $(2, 3)$ | 1 | 2 | 5 | 3 | 2 |
| | - | 3 | $(3, 3)$ | 1 | 2 | | | |
| | 11 | - | $(2, 2)$ | 2 | 9 | | | |
| $n = 8$ | 2 | 3 | $(2, 3)$ | 2 | 3 | 10 | 5 | 5 |
| | - | 4 | $(3, 3)$ | 1 | 3 | | | |

Figure 3 gives the overall cost of the CRV method and of our algebraic decomposition method for various values of $n, d$ and $C$ (these are the cross marked blue to green curves). We used the improved version of CRV suggested in Remark 4 (*i.e.* using CPRR multiplications for the first phase and ISW multiplications for the second phase). For our method, we considered quadratic decomposition (*i.e.* $s_1 = s_2 = 2$) combined with Algorithm 2 for secure quadratic evaluations, as well as cubic decomposition ($s_1 = s_2 = 3$) with `TreeExplore` for secure cubic evaluation. We further considered hybrid decomposition combined with both Algorithm 2 and `TreeExplore`.

We observe that the quadratic decomposition is always more efficient than the cubic and hybrid decompositions. This is due to the gap of efficiency between the secure quadratic evaluation (Algorithm 2) and the secure cubic evaluation (`TreeExplore`). Compared to CRV, the quadratic decomposition offers some efficiency gain depending on the multiplication cost. For $n = 4$, we observe that it is more efficient than CRV whenever a multiplication takes at least 10 elementary operations. For $n = 8$, the quadratic decomposition is better than CRV whenever the multiplication cost exceeds 5 elementary operations. This shows that our algebraic decomposition approach is the best known method for the probing secure evaluation of nonlinear functions (such as s-boxes) in a context where the field multiplication is a costly operation.

## References

1. J. Balasch, S. Faust, and B. Gierlichs. Inner Product Masking Revisited. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 486–510. Springer, 2015.
2. G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, and P. Strub. Verified proofs of higher-order masking. In E. Oswald and M. Fischlin, editors, *Advances in*

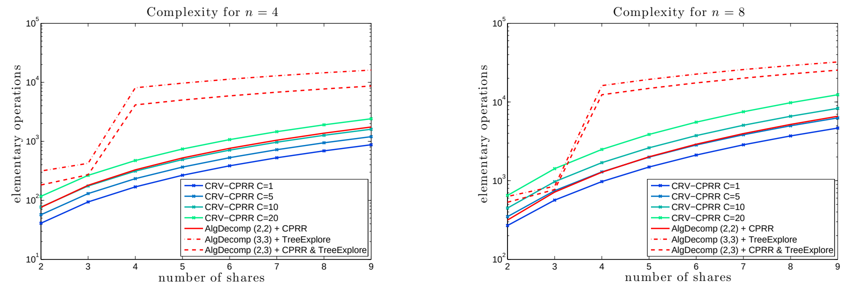

**Fig. 3.** Secure evaluation of nonlinear functions (arbitrary degree)

*Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.

3. B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. Threshold Implementations of All 3×3 and 4×4 S-Boxes. In E. Prouff and P. Schaumont, editors, *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2012.
4. B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, N. Tokareva, and V. Vitkup. Threshold implementations of small S-boxes. *Cryptography and Communications*, 7(1):3–33, 2015.
5. C. Carlet. *Vectorial Boolean Functions for Cryptography*, chapter 9, pages 398–469. Boolean Models and Methods in Mathematics, Computer Science, and Engineering. Cambridge University Press, June 2010.
6. 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.
7. J.-S. Coron. Fast Evaluation of Polynomials over Finite Fields and Application to Side-channel Countermeasures. Personnal communication., February 2014.
8. J.-S. Coron, E. Prouff, M. Rivain, and T. Roche. Higher-order side channel security and mask refreshing. In S. Moriai, editor, *FSE*, Lecture Notes in Computer Science. Springer, 2013. To appear.
9. J.-S. Coron, A. Roy, and S. Vivek. Fast Evaluation of Polynomials over Finite Fields and Application to Side-channel Countermeasures. In *CHES*, 2014. 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, *Eurocrypt*, Lecture Notes in Computer Science. Springer, 2014. To appear.
11. L. Genelle, E. Prouff, and M. Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.
12. Grosso, Vincent and Prouff, Emmanuel and Standaert, François-Xavier . Efficient Masked S-Boxes Processing, A Step Forward. In D. Poincheval and D. Vergnaud, editors, *Africacrypt*, Lecture Notes in Computer Science. Springer, 2014. To appear.

13. Y. Ishai, A. Sahai, and D. Wagner. 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. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
16. P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
17. S. Kutzner, P. H. Nguyen, and A. Poschmann. Enabling 3-Share Threshold Implementations for all 4-Bit S-Boxes. In H. Lee and D. Han, editors, *Information Security and Cryptology - ICISC 2013 - 16th International Conference, Seoul, Korea, November 27-29, 2013, Revised Selected Papers*, volume 8565 of *Lecture Notes in Computer Science*, pages 91–108. Springer, 2013.
18. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.
19. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. In P. J. Lee and J. H. Cheon, editors, *ICISC*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2009.
20. S. Nikova, V. Rijmen, and M. Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.
21. A. Poschmann, A. Moradi, K. Khoo, C. Lim, H. Wang, and S. Ling. Side-Channel Resistant Crypto for Less than 2, 300 GE. *J. Cryptology*, 24(2):322–345, 2011.
22. E. Prouff and M. Rivain. Higher-Order Side Channel Security and Mask Refreshing. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013 - 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.
23. E. Prouff and T. Roche. Higher-order glitches free implementation of the aes using secure multi-party computation protocols. In B. Preneel and T. Takagi, editors, *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2011.
24. 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.
25. 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.
26. A. Satoh, S. Morioka, K. Takano, and S. Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. In E. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.