Computing the restricted algebraic immunity, and application to WPB functions.

Luca Bonamino, Pierrick Méaux

Luxembourg university, Luxembourg
pierrick.meaux@uni.lu

Abstract. Algebraic immunity is a fundamental property in the security analysis of stream ciphers and Boolean functions, measuring the resistance of a function against algebraic attacks. In this work, we focus on the computation of algebraic immunity and, more specifically, on its generalization to restricted algebraic immunity, which considers annihilators constrained to specific subsets of $\mathbb{F}_2^n$. While the computation of algebraic immunity has been studied using methods based on Reed-Muller codes and iterative rank-based algorithms, these approaches have not been formally adapted to the restricted setting. We address this gap by establishing the theoretical foundations required for the adaptation of these techniques and providing explicit algorithms for computing restricted algebraic immunity.
To assess the efficiency of our algorithms, we conduct practical experiments comparing the computational cost of the Reed-Muller and iterative approaches in terms of time and memory. As a case study, we analyze the algebraic immunity restricted to the slices of $\mathbb{F}_2^n$, *i.e.* the sets of binary vectors of fixed Hamming weight, denoted AI_k. The slices are of particular interest in areas of cryptography, such as side-channel analysis and stream cipher design, where the input weight may be partially predictable. We further investigate restricted algebraic immunity for Weightwise (Almost) Perfectly Balanced (W(A)PB) functions, which have been extensively studied since 2017. Our results include an empirical analysis of AI_k distributions for WPB functions with 4, 8, and 16 variables, as well as an evaluation of AI_k for various known function families.

Keywords: Boolean functions, Algebraic immunity, Restricted algebraic immunity, Weightwise (almost) perfectly balanced functions.

1 Introduction

Since the introduction of algebraic attacks on filtered LFSRs [CM03] in 2003, Algebraic Immunity (AI) has become a crucial property to consider in the security of stream ciphers. As defined in [MPC04], algebraic immunity refers to the minimum degree of a nontrivial annihilator of a Boolean function f or its complement $1 + f$. In the context of filtered LFSRs, algebraic immunity directly corresponds to the degree of the algebraic system an adversary can derive from the keystream, since only affine mappings occur before the application of the nonlinear filter f. Consequently, algebraic immunity plays a key role in settings where affine updates are followed by a filtering function, as demonstrated in [AL18] for Goldreich's pseudorandom generator [Gol00] and for (improved) filtered permutators [MJSC16, MCJS19]. Studying algebraic immunity (*e.g.* [Did06, CM13, Méa22]) or constructing Boolean functions that optimize this parameter (*e.g.* [DMS06, Car07, CF08, QFLW09, ZCSH11, TD11]) has been an important line of research in the area of Boolean functions used in cryptography.

In this work, we focus on the computation of this parameter, specifically on a generalization of it. The complexity of computing the algebraic immunity of a Boolean function has been investigated in various works (*e.g.* [CM03, ACG$^+$06, HR04, DT06]). One approach, which to our knowledge has rarely been formally articulated and is often considered folklore, relies on submatrices of generator matrices of the Reed-Muller code. We briefly recall its theoretical foundation here. The truth table of an n-variable Boolean function of degree at most r can be identified with a codeword of the Reed-Muller code $RM(r, n)$, where each codeword's element corresponds to the evaluation of a function at an element of $\mathbb{F}_2^n$. Thus, by

puncturing the Reed-Muller code of order r, keeping only the elements x such that $f(x) = 1$, one obtains the code of the products of f with functions of degree at most r. The dimension of this code equals that of the Reed-Muller code of order r if and only if f does not admit a nonzero annihilator of degree at most d. The most efficient algorithm for computing the AI is the iterative method introduced in [ACG+06]. Rather than constructing submatrices of the generator matrix of a Reed-Muller code for successive values of d, this algorithm builds them incrementally by adding one row and one column at a time. Using the same theoretical foundation as the initial approach, a rank drop in the constructed matrices indicates the existence of an annihilator. This iterative process improves computational efficiency, and specific operations designed to reduce the cost of updating successive matrices improve the initial time complexity from $\mathcal{O}(\binom{n}{\mathsf{AI}(f)}^{\omega})$ (where $\mathsf{AI}(f)$ denotes the AI of f) with $\omega = 3$ to a value closer to 2 [ACG+06, HR04].

In this work, we focus on a generalization of algebraic immunity, namely the restricted algebraic immunity, introduced in [CMR17]. This property considers a subset $S \subseteq \mathbb{F}_2^n$ and corresponds to the minimal degree of an annihilator g that annihilates f or $f + 1$ over the entire set S but is not identically zero on S. More formally, it is defined as:

$$\mathsf{AI}_S(f) = \min_g \left(\deg(g) \mid fg = 0 \text{ or } g(f+1) = 0, \text{ and } \exists x \in S \text{ such that } g(x) = 1\right).$$

In this setting, classical algebraic immunity corresponds to the special case where $S = \mathbb{F}_2^n$. Since neither the Reed-Muller approach nor the iterative method has been fully formalized and described for this generalization, we address this gap by establishing the key properties that ensure the correctness of the adapted algorithms for restricted algebraic immunity. Additionally, we provide corresponding algorithms and code implementations.

In Section 3, we formalize the Reed-Muller approach, and in Section 4, we analyze the iterative method, emphasizing the theoretical and practical challenges that arise when applying this technique to restricted sets. Variants of the algebraic attack, such as fast algebraic attacks [Cou03, Arm04] and attacks leveraging monomials of extreme degrees [MW24], require computing parameters in a manner similar to restricted algebraic immunity. Accordingly, we hope that this work will be useful for further research in these directions.

Next, we perform experiments with our algorithms. First, we compare the practical efficiency of the two main approaches in terms of both runtime and memory usage. For this comparison, we consider algebraic immunity restricted to a slice of $\mathbb{F}_2^n$, specifically a set $\mathsf{E}_{k,n}$ consisting of all binary vectors of length n with Hamming weight k. These sets, as well as functions that are balanced on all such sets, known as Weightwise (Almost) Perfectly Balanced (W(A)PB) functions, have attracted considerable attention since the introduction of restricted criteria in [CMR17]. Slices of $\mathbb{F}_2^n$ frequently appear in cryptographic research, notably in contexts such as the Hamming weight model in side-channel analysis and the fixed input weight setting in FLIP [MJSC16]. In particular, WAPB functions have been extensively studied, with research focusing primarily on two aspects. The first is the construction of such functions, which has been explored in various works [CMR17, TL19, LM19, LS20, MS21, ZS22, MPJ+22, MKCL22, GM22b, MSLZ22, GS22, ZLC+23, ZS23, ZJZQ23, YCL+23, GM24, DM24, Méa24, DMM24]. The second is the study of their cryptographic properties, including nonlinearity restricted to slices [GM22a], overall nonlinearity [GM23b], and algebraic immunity [GM23a].

In this work, we further investigate algebraic immunity restricted to slices, denoted as AI_k, particularly in the context of WAPB functions, which are relevant when the Hamming weight of a function's input can be inferred. In Section 5.2, we analyze the distribution of AI_k for WPB functions with 4, 8, and even 16 variables. Then, in Section 5.3, we determine the values of AI_k for several families of functions introduced in previous works.

2 Preliminaries

For readability, we use the notation $+$ instead of $\oplus$ to denote the addition in $\mathbb{F}_2$, and $\sum$ instead of $\bigoplus$. In addition to classic notations, we denote by $[a, b]$ the subset of all integers between a and b: $\{a, a+1, \ldots, b\}$. For a vector $v \in \mathbb{F}_2^n$ we use $\mathsf{w_H}(v)$ to denote its Hamming weight $\mathsf{w_H}(v) = |\{i \in [1, n] \,|\, v_i = 1\}|$. For two vectors v and w in $\mathbb{F}_2^n$ we denote by $\mathsf{d_H}(v, w)$ the Hamming distance between v and w, that is, $\mathsf{d_H}(v, w) = \mathsf{w_H}(v + w)$. For two functions f and g we denote by $\mathsf{d_H}(f, g)$ the Hamming distance between their vectors of values.

2.1 Boolean functions, cryptographic criteria, and weightwise properties

In this section, we recall fundamental concepts concerning Boolean functions and their weightwise properties, which are utilized throughout this article. For a more comprehensive introduction to Boolean functions and their cryptographic parameters, we recommend consulting the book by Carlet [Car21], and for insights into weightwise properties—also known as properties on the slices—the article by [CMR17]. We denote by $\mathsf{E}_{k,n}$ the set $\{x \in \mathbb{F}_2^n \,|\, \mathsf{w_H}(x) = k\}$ for $k \in [0, n]$, referring to it as a slice of the Boolean hypercube (of dimension n). Consequently, the Boolean hypercube is divided into $n + 1$ slices, where the elements share the same Hamming weight.

Definition 1 (Boolean Function). *A Boolean function f in n variables is a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The set of all Boolean functions in n variables is denoted by $\mathcal{B}_n$.*

When a property or a definition is restricted to a slice, we denote it by using the subscript k. For example, for an n-variable Boolean function f we denote its support $\mathsf{supp}(f) = \{x \in \mathbb{F}_2^n \,|\, f(x) = 1\}$. Furthermore, we denote by $\mathsf{supp}_k(f)$ the support of f restricted to a slice, which is defined as $\mathsf{supp}(f) \cap \mathsf{E}_{k,n}$.

Definition 2 (Balancedness). *A Boolean function $f \in \mathcal{B}_n$ is called balanced if $|\mathsf{supp}(f)| = 2^{n-1} = |\mathsf{supp}(f + 1)|$.*

For $k \in [0, n]$ the function is said balanced on the slice k if $||\mathsf{supp}_k(f)| - |\mathsf{supp}_k(f + 1)|| \leq 1$. In particular when $|\mathsf{E}_{k,n}|$ is even $|\mathsf{supp}_k(f)| = |\mathsf{supp}_k(f + 1)| = |\mathsf{E}_{k,n}|/2$.

Using the notion of restricted balancedness we can define the weightwise (almost) perfectly balanced functions, the focus of our work.

Definition 3 (Weightwise (Almost) Perfectly Balanced Function (WPB and WAPB)). *Let $m \in \mathbb{N}^*$ and f be a Boolean function in $n = 2^m$ variables. It will be called weightwise perfectly balanced (WPB) if, for every $k \in [1, n-1]$, f is balanced on the slice k, that is $\forall k \in [1, n-1], |\mathsf{supp}_k(f)| = \binom{n}{k}/2$, and:*

$$f(0, \cdots, 0) = 0, \quad \text{and } f(1, \cdots, 1) = 1.$$

The set of WPB functions in 2^m variables is denoted $\mathcal{WPB}_m$.

When n is not a power of 2, other weights than $k = 0$ and n can lead to slices of odd cardinality, we call $f \in \mathcal{B}_n$ weightwise almost perfectly balanced (WAPB) if:

$$|\mathsf{supp}_k(f)| = \begin{cases} |\mathsf{E}_{k,n}|/2 & \text{if } |\mathsf{E}_{k,n}| \text{ is even,} \\ (|\mathsf{E}_{k,n}| \pm 1)/2 & \text{if } |\mathsf{E}_{k,n}| \text{ is odd.} \end{cases}$$

The set of WAPB functions in n variables is denoted $\mathcal{WAPB}_n$.

We define additional crucial concepts for studying Boolean functions, namely the algebraic normal form and the Walsh transform. Subsequently, we introduce key cryptographic criteria for these functions, including algebraic immunity (both general and weightwise) and nonlinearity (both general and weightwise).

Definition 4 (Algebraic Normal Form (ANF) and degree). *We call Algebraic Normal Form of a Boolean function f its n-variable polynomial representation over $\mathbb{F}_2$ (i.e. belonging to $\mathbb{F}_2[x_1, \ldots, x_n]/(x_1^2 + x_1, \ldots, x_n^2 + x_n)$):*

$$f(x_1, \ldots, x_n) = \sum_{I \subseteq [1,n]} a_I \left(\prod_{i \in I} x_i \right),$$

where $a_I \in \mathbb{F}_2$. The (algebraic) degree of f, denoted $\mathsf{deg}(f)$ is:

$$\mathsf{deg}(f) = \max_{I \subseteq [1,n]} \{|I| \,|\, a_I = 1\} \text{ if } f \text{ is not null}, 0 \text{ otherwise}.$$

We also recall the concept of order on $\mathbb{F}_2^n$, as it will be utilized in the algorithms involving Boolean functions.

Definition 5 (Order). *A binary relation $\preceq$ on a set X is called partial order if $\preceq$ is reflexive, transitive and antisymmetric. Moreover, $\preceq$ is a total order if for all $a, b \in X$ it holds $a \preceq b$ or $b \preceq a$.*

We give two examples of orders on n-length binary strings, more examples can be found in *e.g.* [SW12]:

- Lexicographic, given $a, b \in \mathbb{F}_2^n$ as $a = a_1, \ldots, a_n$ and $b = b_1, \ldots, b_n$, $a \preceq_{lex} b$ if and only if $a_i < b_i$ on the first index $i \in [1, n]$ such that $a_i \neq b_i$, or $a = b$.
- Graded lexicographic, given $a, b \in \mathbb{F}_2^n$ as $a = a_1, \ldots, a_n$ and $b = b_1, \ldots, b_n$, $a \preceq_{glex} b$ if $w_H(a) < w_H(b)$ or $w_H(a) = w_H(b)$ and $a \preceq_{lex} b$

Definition 6 (Truth table). *Let $f \in \mathcal{B}_n$ be a Boolean function, its truth table is the Boolean vector defined as:*

$$TT(f) \in \mathbb{F}_2^{2^n},\ TT(f) = (f(0_n), \ldots, f(1_n)).$$

Denoting by f_S the restriction of f to a subset $S \subseteq \mathbb{F}_2^n$ written as $S = \{a_1, a_2, \cdots, a_{|S|}\}$, the truth table of f_S refers to the vector $\big(f(a_1), f(a_2), \ldots, f(a_{|S|})\big)$.

If not specified otherwise, the (restricted) truth table follows the lexicographic order.

Definition 7 (Algebraic immunity and restricted algebraic immunity). *The algebraic immunity of a Boolean function $f \in \mathcal{B}_n$, denoted as $\mathsf{AI}(f)$, is defined as:*

$$\mathsf{AI}(f) = \min_{g \neq 0} \{\mathsf{deg}(g) \mid fg = 0 \text{ or } (f+1)g = 0\},$$

where $\mathsf{deg}(g)$ is the algebraic degree of g. The function g is called an annihilator of f (or $f + 1$).

The restricted algebraic immunity of a Boolean function $f \in \mathcal{B}_n$ on the set $S \subset \mathbb{F}_2^n$, denoted as $\mathsf{AI}_S(f)$, is defined as:

$$\mathsf{AI}_S(f) = \min_{g \neq 0 \text{ over } S} \{\mathsf{deg}(g) \mid fg = 0 \text{ or } (f+1)g = 0\}.$$

For $S = \mathsf{E}_{k,n}$ we denote $\mathsf{AI}_{\mathsf{E}_{k,n}}(f)$ by $\mathsf{AI}_k(f)$ and call it weightwise algebraic immunity.

Remark 1. Let $f \in \mathcal{B}_n$. We use the notation $f^{(o)}$ referring to either f or $f + 1$ when the same process or result applies to any of them. In particular, $f^{(o)} \in \{f, f + 1\}$.

2.2 Reed-Muller codes

Definition 8 (Monomials and set of monomials). *Let $u, x \in \mathbb{F}_2^n$ denoted as $x = (x_1, \cdots, x_n)$ and $u = (u_1, \cdots, u_n)$, the monomial x^u is calculated as:*

$$x^u = \prod_{i=1}^{n} x_i^{u_i}.$$

Let $E = \{\alpha_1, \alpha_2, \cdots, \alpha_k\} \subseteq \mathbb{F}_2^n$, we denote by $x^E = \{x^{\alpha_1}, x^{\alpha_2}, \cdots, x^{\alpha_k}\}$ the set of all monomials associated to E.

Definition 9 (Reed-Muller code - RM). *Let $n \in \mathbb{N}^*$ and $r \in \mathbb{N}$ with $r \leq n$, the Reed-Muller code $RM(r, n)$ is the set of all 2^n-bit length binary vectors where each code word is the evaluation of one distinct Boolean function of degree at most r in n variables.*

Definition 10 (RM generator matrix). *Let X be the set of the 2^n elements in $\mathbb{F}_2^n$ in lexicographical order and $P_{r,n} = \{ u \in \mathbb{F}_2^n \mid w_H(u) \leq r\}$ the set of all the elements of $\mathbb{F}_2^n$ with Hamming weight smaller than or equal to r. The generator matrix $G_{r,n} \in Mat_{D_r^n, 2^n}$ is defined as*

$$(G_{r,n})_{i,j} = x_j^{u_i},\ x_j \in X,\ u_i \in P_{r,n}.$$

We denote by $G_{r,n}^Y \in Mat_{D_r^n, |Y|}$, the Reed-Muller generator matrix restricted on the set $Y \subset X$

$$(G_{r,n})_{i,j}^Y = x_j^{u_i},\ x_j \in Y,\ u_i \in P_{r,n}.$$

Remark 2. Let $\hat{a}$ be the vector corresponding to the coefficients in the ANF of a Boolean function f of n variables and degree r, then

$$\hat{a} \cdot G_{r,n} = TT(f).$$

3 Computing the restricted AI from (punctured) generator matrices of Reed-Muller codes

In this section we provide an algorithm to compute the algebraic immunity of a Boolean function restricted on a general subset $S \subseteq \mathbb{F}_2^n$.

The algorithm aims to determine whether the systems of equations $S_1^r \hat{=} g_1^r * f_S(x) = \hat{0}$ or $S_2^r \hat{=} g_2^r (f + 1)_S(x) = \hat{0}$ admit a non-zero solution for $x \in \mathbb{F}_2^n$, where g_1^r and g_2^r are Boolean functions of algebraic degree at most r. This is achieved by examining whether the rank of one of the punctured Reed-Muller (RM) matrices is smaller than one with the columns corresponding to the entire S. If f is non-constant on S, the algorithm searches for the existence of either g_1^r solving S_1^r or g_2^r solving S_2^r, where neither function vanishes over the entire set S. If such solutions g_1^r or g_2^r do not exist, new systems S_1^{r+1} and S_2^{r+1} are considered (through bigger matrices), and the search for solutions g_1^{r+1} and g_2^{r+1} continues. This iterative process is repeated until either an annihilator of f or $f + 1$ is found or an upper bound $r + \delta$ is reached. The procedure for f_S and $(f + 1)_S$ is carried out in parallel. Conversely, if f is constant on S, the algorithm terminates and returns 0, as justified by Proposition 1.

Proposition 1. *A Boolean function f has an AI_S of 0 if and only if the restriction f_S on S is constant.*

Proof. First, we show the reverse implication. We suppose that $f \in \mathcal{B}_n$ is constant on S, either $f(x) = 0$ for all $x \in S$, or $f(x) = 1$ for all $x \in S$. If f is not null everywhere but $f(x) = 0 \; \forall x \in S$, then the constant function $g(x) = 1$ is not zero on all S, and is it such that $g(x)f(x) = 0$ for all $x \in S$. Similarly, if for all $x \in S$ $f(x) = 1$, the same argument can be used for the function $f + 1$ on S.

Then, we show the direct implication. We suppose that a function $g \in \mathcal{B}_n$ is a non-zero-annihilator of f restricted to S of degree 0. Then, $g \neq 0$ and $\deg(g) = 0 \Rightarrow g = 1$, therefore since g annihilates f in S, $g(x)f(x) = 0$ for all $x \in S$, resulting in $f(x)g(x) = f(x) = 0$. Similarly, if g annihilates $f + 1$ on S instead of f, we have that $g(x)(f(x) + 1) = 0$ implying that $f + 1 = 0 \Rightarrow f(x) = 1$ for all $x \in S$. □

The matrices corresponding to systems of equations S_1^r and S_2^r are constructed by extracting the S-f-generator and the S-$(f + 1)$-generator given by Definition 11, by extracting them from the Reed-Muller generator matrix where we only consider the columns corresponding to elements in S. This is justified by Remark 3.

Definition 11. *Let $r, n \in \mathbb{N}^*$ with $r \leq n$. Let $G_{r,n}$ be the generator matrix of the Reed-Muller code $RM(r, n)$, we define the object $G_{r,n}^{f_S^{(o)},S}$ by setting to zero all the columns of $G_{r,n}^S$ which do not correspond to elements of $supp(f^{(o)}) \cap S$. For the remaining columns, we calculate $G_{r,n}^{f_S^{(o)},S}$ as*

$$\left(G_{r,n}^{f_S^{(o)},S}\right)_{i,j} = f_S^{(o)}(u_j) \cdot \left(G_{r,n}^S\right)_{i,j}, \; u_j \in S$$

Moreover, we define the object $G_{r,n}^{f^{(o)}_{supp\left(f^{(o)}\right)\cap S},S}$, by taking all the columns of $G_{r,n}$ which corresponds to element of $\mathbb{F}_2^n$ which are in $supp\left(f^{(o)}\right) \cap S$.

$$\left(G_{r,n}^{f^{(o)}_{supp\left(f^{(o)}\right)\cap S},S}\right)_{:,j} = (G_{r,n}^S)_{:,j} \text{ if } f_S^{(o)}(u_j) = 1 \tag{1}$$

Remark 3. Let $g \in \mathcal{B}_n$ a Boolean function of degree $r \leq n$ and $\hat{a} = (a_I)_{\substack{I\subseteq[1,n] \\ deg\left(x^I\right)\leq r}}$ be the vector of the coefficients of the ANF of g, as a consequence of the construction of $G_{r,n}^{f,S}$, we have that

$$\begin{aligned}
\hat{a} \cdot G_{r,n}^{f_S,S} &= \left(\sum_{i=1}^{D_r^n} a_i f_S(u_1)(G_{r,n}^S)_{i,1}, \sum_{i=1}^{D_r^n} a_i f_S(u_2)(G_{r,n}^S)_{i,2}, \cdots, \sum_{i=1}^{D_r^n} a_i f_S(u_n)(G_{r,n}^S)_{i,n}\right) \\
&= \left(f_S(u_1)\sum_{i=1}^{D_r^n} a_i (G_{r,n}^S)_{i,1}, f_S(u_2)\sum_{i=1}^{D_r^n} a_i (G_{r,n}^S)_{i,2}, \cdots, f_S(u_n)\sum_{i=1}^{D_r^n} a_i (G_{r,n}^S)_{i,n}\right) \\
&= (f_S(u_1)g(u_1), f_S(u_2)g(u_2), \cdots, f_S(u_n)g(u_n)) \\
&= f_S \cdot g.
\end{aligned} \tag{2}$$

Therefore, solving $\hat{a} \cdot G_{r,n}^{f_S,S} = 0$ is equivalent to solving $f_S \cdot g = 0$, or in other words, finding the kernel of $G_{r,n}^{f_S,S}$ is equivalent to finding all annihilators of f restricted to S of degree up to r.

Proposition 2. *Let $Z = \mathsf{supp}\left(f^{(o)}\right) \cap S$. If rank $\left(G_{r.n}^{f_Z^{(o)},S}\right) <$ rank$(G_{r.n}^S)$, then $f^{(o)}$ admits a non-zero-annihilator restricted to S of degree at most r.*

Proof. Since $D_r^n = \text{rank}(G_{r.n}^{f_S^{(o)},S}) + |\text{Ker}(G_{r.n}^{f_S^{(o)},S})| = \text{rank}(G_{r.n}^S) + |\text{Ker}(G_{r.n}^S)|$, we have that $\text{rank}(G_{r.n}^{f_S^{(o)},S}) < \text{rank}(G_{r.n}^S) \iff |\text{Ker}(G_{r.n}^{f_S^{(o)},S})| > |\text{Ker}(G_{r.n}^S)|$. Therefore $\exists v \in \text{Ker}(G_{r.n}^{f_S^{(o)},S})$ such that $v \notin \text{Ker}(G_{r.n}^S)$. Hence, it exists a Boolean function g of degree r, given by $v \cdot G_{r,n}$, which is such that $v \cdot G_{r,n} \neq 0$ and $v \cdot G_{r,n}^S \neq 0$, but $(g \cdot f_S^{(o)})_S = v \cdot G_{r,n}^{f_S^{(o)},S} = 0$.
Since $\text{rank}\left(G_{r,n}^{f_S^{(o)},S}\right) = \text{rank}\left(G_{r,n}^{f^{(o)},S}\right)$, $\text{rank}(G_{r.n}^{f^{(o)},S}) < \text{rank}(G_{r.n}^S) \Rightarrow \text{rank}(G_{r.n}^{f_Z^{(o)},S}) < \text{rank}(G_{r.n}^S)$ and hence the condition $\text{rank}(G_{r.n}^{f_Z^{(o)},S}) < \text{rank}(G_{r.n}^S)$ is enough to justify the existence of such v giving the non-zero-annihilator g of f^o restricted to S. □

Algorithm 1, provides the full procedure to determine the restricted algebraic immunity of a function f of n variables on the subset S. We give additional details:

Remark 4.

1. [CMR17] established that the upper bound of $AI_s(f)$ for any Boolean function f and any set $S \subseteq \mathbb{F}_2^n$, is the smallest integer r such that $\left(G_{r,n}^n\right) > |S|/2$. As a consequence, the search of the annihilator for f and $f+1$ in our algorithm terminates once $rank\left(G_{r,n}^S\right) > \lfloor |S|/2 \rfloor$.
2. The generator matrix of a Reed-Muller code does not depend on the Boolean function. It is therefore possible to pre-compute and store the generator matrices of all Reed-Muller codes with r and n smaller than some maximum values r_{max} and n_{max}. This suggests that we can pre-compute $G_{r_{max},n_{max}}$ for some fixed n_{max} and $r_{max=n_{max}}$, and extract all others $G_{r,n}$ for $r \leq r_{max}$ and $n \leq n_{max}$ from it. We therefore pre-compute the hash table

$$G^{\leq n_{max}} = \{(i, G_{n,i})\}_{i \leq 1 \leq n_{max}} \tag{3}$$

   and store it in a file.
   Similarly, we can also pre-compute and store a second hash table, containing all the D^ns such that $n \leq n_{max}$. Such hash table is

$$D^{\leq n_{max}} = \{(i, D^n)\}_{0 \leq i \leq n_{max}} = \{\left(i, \{\left(j, D_i^j\right)\}_{0 \leq j \leq \lceil i/2 \rceil}\right)\}_{0 \leq i \leq n_{max}} \tag{4}$$

   In Algorithm 1, we therefore use the function READ_VALUES_FROM_FILE, to read to two pre-computed hash tables from the files where they have been pre-stored before the execution of the algorithm.
3. The procedure FIND_DEG_SMALLEST_S_ANNIHILATOR in 1 is done in parallel for f and $f+1$ with the additional condition that if one of the two branches returns the value of 1, the other branch is stopped, without having to wait for it to finish.

4 Computing the restricted AI iteratively

In this section, we present an algorithm (Algorithm 2) to determine the algebraic immunity restricted to a set $S \subset \mathbb{F}_2^n$ with improved time and space complexity compared to the method presented in Section 3.

[1] The symbol || is used as row-concatenation of matrices, meaning that $C = A||B$ is the matrix constructed by appending all the row of the matrix B to the matrix A.

[2] The function MIN takes care of $null$ values $\text{MIN}(immunity_f, null) = immunity_f$ and $\text{MIN}(null, immunity_{f+1}) = immunity_{f+1}$.

Algorithm 1 Algebraic immunity of f restricted to the set S, Reed-Muller method.

Input: $x \in \mathbb{F}_2^{2^n}$ evaluation of the truth table of $f \in \mathcal{B}_n$ and the restricted set $S \subseteq \mathbb{F}_2^n$.
Output: $\mathsf{AI}_S(f)$

function FIND_DEG_SMALLEST_S_ANNIHILATOR($G_{n,n}^{f_Z^{(o)},S}, G_{d,n}^S, D^n$)
    $r \leftarrow 0$;
    $i \leftarrow 0$;
    $ef_f, ef_S \leftarrow null, null$;
    $previousIndex \leftarrow 0$;
    while $r \leq |S|/2$ **do**
        $D_i^n \leftarrow D^n[i]$;
        $ef_f \leftarrow$ ECHELON_FORM_LAST_D_i^n_ROWS $\left(ef_f \mid\mid G_{n,n}^{f_Z^{(o)},S}[previousIndex : D_i^n]\right)$[1];
        $ef_S \leftarrow$ ECHELON_FORM_LAST_D_i^n_ROWS $\left(ef_S \mid\mid G_{n,n}^S[previousIndex : D_i^n]\right)$;
        $r \leftarrow rank\,(ef_S)$;
        if $rank\,(ef_f) < r$ **then**
            return i;
        end if
        $previousIndex \leftarrow D_i^n$;
        $i \leftarrow i + 1$;
    end while
end function

if IS_CONSTANT(x) **then return** 0
end if
$m_{max}, D^{\leq n_{max}} \leftarrow$ READ_VALUES_FROM_FILE();
$G_{n,n} \leftarrow m_{max}[n]$;
$D^n \leftarrow D^{\leq n_{max}}[n]$;
$G_{rows,n}^{f_Z,S}, G_{rows,n}^{(f+1)_Z,S}, G_{n,n}^S \leftarrow [\,], [\,], [\,]$;
for $i \leftarrow 0$ to $2^n - 1$ **do**
    if $i \in S$ **then**
        $G_{rows,n}^S \leftarrow G_{rows,n}^S \mid\mid G_{n,n}[:][i]$;
        if $x[i] = 1$ **then**
            $G_{rows,n}^{f_S,S} \leftarrow G_{rows,n}^{f_S,S} \mid\mid G_{n,n}[:][i]$;
        else
            $G_{rows,n}^{(f+1)_S,S}.append\,(G_{n,n}[:][i])$;
        end if
    end if
end for
$immunity_f, immunity_{f+1} \leftarrow ($
FIND_DEG_SMALLEST_S_ANNIHILATOR($G_{rows,n}^{f_Z,S}, G_{n,n}^S, D^n$)
| FIND_DEG_SMALLEST_S_ANNIHILATOR($G_{rows,n}^{(f+1)_Z,S}, G_{n,n}^S, D^n$)
);
return MIN($immunity_f, immunity_{f+1}$)[2];

The core idea is to iteratively construct the objects $G_{r,n}^{f_{supp(f)\cap S},S}$ and $G_{r,n}^{(f+1)_{supp(f+1)\cap S},S}$ for varying r, increasing the size of the matrix by one row per iteration while minimizing computations. This approach checks for the existence of annihilators dynamically, rather than extracting them from a large, pre-computed Reed-Muller generator matrix restricted to S. The method extends the approach of [ACG+06] for determining the standard algebraic immunity. As with Algorithm 1, this procedure returns 0 if the function f is constant on S. Hence, we focus on explaining the case of determining $\mathsf{AI}_S(f)$ for functions such that $\mathsf{AI}_S(f) \geq 1$. (The full algorithm, however, also addresses functions with null AI_S.)

We begin with definitions and remarks necessary to introduce the general idea of the algorithm.

Definition 12 (T-Vandermonde matrix). *Let $Z = \{z_1, \cdots, z_{k_Z}\} \subseteq \mathbb{F}_2^n$ and $E = \{\alpha_1, \cdots, \alpha_{k_E}\} \subseteq \mathbb{F}_2^n$, we define the matrix $V_{E,Z}$ whose entries are*

$$(V_{E,Z})_{i,j} = z_j^{\alpha_i} \quad \forall i \in [1, k_E] \text{ and } j \in [1, k_Z].$$

We refer to this matrix as the T-Vandermonde matrix, as it is the transpose of a Vandermonde matrix. Denoting by $Z_i = \{z_1, z_2, \cdots, z_i\}$ and $E_j = \{\alpha_1, \ldots, \alpha_j\}$ for $i \in [1, z_{k_Z}]$ and $j \in [1, \alpha_{k_E}]$, the sub-matrix V_{E_i,Z_j} of $V_{E,Z}$, is given by the elements:

$$\left(V_{E_i,Z_j}\right)_{n,l} = z_l^{\alpha_n} \quad \forall n \in [1, i] \text{ and } l \in [1, j].$$

Similarly, we define the matrices $S_{E,S}$ and S_{E_i,S_j} as the T-Vandermonde sub-matrices corresponding to the elements of the set $S \subseteq \mathbb{F}_2^n$.

Remark 5. Let $E = (\{\alpha_2, \alpha_2, \cdots, \alpha_{D_d^n}\})$ be such that x^E is the set of all monomials with degree at most d ordered by the graded lex order. We remark the following:

- $V_{E,Z} = G_{d,n}^{f_Z,S}$.
- Let $E^* \subset E$, ordered by graded lex order. V_{Z,E^*} is a sub-matrix of $G_{d,n}^{f_Z,S}$.

Remark 6. Let $i \geq 1$. If $1 < i + 1 \leq \min\left(|Z|, |E_{D_n^n}|\right)$, then $V_{Z_{i+1},E_{i+1}}$ can be computed as

$$V_{E_{i+1},Z_{i+1}} = \begin{pmatrix} & & z_{i+1}^{\alpha_1} \\ & V_{E_i,Z_i} & z_{i+1}^{\alpha_2} \\ & & \vdots \\ & & z_{i+1}^{\alpha_i} \\ z_1^{\alpha_{i+1}} & z_2^{\alpha_{i+1}} \cdots z_i^{\alpha_{i+1}} & z_{i+1}^{\alpha_{i+1}} \end{pmatrix}.$$

This can be seen by combining Definition 12 and the fact that for all i such that $1 < i + 1 \leq \min(|Z|, |E| - 1)$, the matrix $V_{Z_{i+1},E_{i+1}}$ is square.

Definition 13 (Sequence of V_k matrices). *As a consequence of Remark 6, we define the sequence of sub-matrices $(V_k)_{k\in[1,D^n_{\lceil n/2\rceil-1}]}$ by*

$$V_k = \begin{cases} \begin{pmatrix} & & z_k^{\alpha_1} \\ & V_{k-1} & z_k^{\alpha_2} \\ & & \vdots \\ & & z_k^{\alpha_{k-1}} \\ z_1^{\alpha_k} & z_2^{\alpha_k} \cdots z_{k-1}^{\alpha_k} & z_k^{\alpha_k} \end{pmatrix} & \text{if } 2 \le k \le \min\left(|Z|, |E_{D^n_{\lceil n/2\rceil-1}}|\right), \\ \begin{pmatrix} V_{k-1} \\ z_1^{\alpha_k} \; z_2^{\alpha_k} \cdots z_k^{\alpha_k} \end{pmatrix} & \text{if } |Z| < k \le |E_{D^n_n}|, \end{cases}$$

with $V_1 = \left(z_1^{\alpha_1}\right) = z_1^{0_n} = (1)$.

With Definition 12 and Remark 5 in mind, we describe the algorithm's procedure as follows. Let $Z = \text{supp}\left(f^{(o)}\right) \cap S$, and let $1 \le i \le |Z|$. If the function f is not constant on S, the algorithm begins by constructing the matrix V_{E_i,Z_i} and searches for functions vanishing on the set Z_i by identifying vectors corresponding to the elements of the kernel of V_{E_i,Z_i}. If such functions exist, the procedure checks whether any of them are also annihilators restricted to S for the full function $f^{(o)}$. This is done by verifying that they vanish on all elements of Z but not on all elements of S. If such an annihilator is found, the algorithm returns the degree of the last monomial added, x^{α_i}. Otherwise, it increments the index i, constructs the next matrix following Definition 13, and continues until a solution is found or all square matrices up to $V_{E_i,|Z|}$ have been constructed and tested. If no annihilator is found after all the elements of Z have been added, the algorithm iteratively adds new monomials and generates the matrices $V_{E_i,Z}$ and S_{E_i,S_i}. It then returns the degree of the monomial x^{α_i} if $\text{rank}\,(V_{E_i,Z}) < \text{rank}\,(S_{E_i,S_i})$. Following point 1 in Remark 4, if this condition is not satisfied at step i, the iteration continues until $\text{rank}\,(S_{E_i,S_i}) > |S|/2$. Indeed, if for some i, $\text{rank}\,(V_{E_i,Z}) < \text{rank}\,(S_{E_i,S_i})$, then there exists a non-null annihilator over S of degree at most $\deg(\alpha_i)$. Conversely, if the condition is not met before $\text{rank}\,(S_{E_i,S_i}) > |S|/2$, Remark 4 guarantees that either f or $f+1$ admits a non-trivial annihilator on S of this degree.

4.1 Keeping the cardinality of the kernel of V_k upper-bounded by 1

In the procedure described, we aim to identify at least one element from a vector space that satisfies certain required conditions (corresponding to an annihilator, not trivial on S), while maintaining the algorithm's desired efficiency. Since this vector space can potentially be too large, we address this issue as follows: if we find an element $\hat{g}_k$ in the right kernel of V_k that does not vanish on Z, we re-order the vector Z by swapping the element z_{k+1} with the element that does not vanish in $\hat{g}_k$. This ensures that such elements of the kernel of V_k do not propagate into the kernel of subsequent matrices.

Furthermore, if an element of the kernel of V_k is found that vanishes on all of Z and S, the monomial α_k is dropped. The matrix V_{k+1} is then constructed by replacing the row corresponding to the monomial α_k with one corresponding to the monomial α_{k+1}. This procedure guarantees that the cardinality of the kernel of V_k is upper-bounded by 1 for all $k \le |Z|$. Propositions 3, 4, and 5 provide formal justifications for these claims.

Proposition 3. *Let $k < \min\{(|Z|, |E_{D^n_n}|\}$ and let V_k constructed following Definition 13. Suppose $\ker(V_k) = \langle \hat{g}_k \rangle = \langle (\epsilon_1, \epsilon_2, \ldots, \epsilon_k) \rangle$ and let $g_k \in \mathcal{B}_n$ be the Boolean function having $\hat{g}_k$ as ANF.*

Furthermore, assume there exists a z_{k+j}, for $j \geq 1$ such that $g_k(z_{k+j}) = 1$. Then, the matrix $V_{k+1}^{'}$ constructed as:

$$V_{k+1}^{'} = \begin{pmatrix} & & z_{k+j}^{\alpha_1} \\ & V_k & z_{k+j}^{\alpha_2} \\ & & \vdots \\ & & z_{k+j}^{\alpha_k} \\ z_1^{\alpha_{k+1}} & z_2^{\alpha_{k+1}} \cdots z_k^{\alpha_{k+1}} & z_{k+j}^{\alpha_{k+1}} \end{pmatrix}$$

is such that $(\epsilon_1, \epsilon_2, \ldots, \epsilon_k, 0) \notin ker(V_{k+1}^{'})$.

Proof. Suppose $\ker(V_k) = \langle \hat{g}_k \rangle = \langle (\epsilon_1, \epsilon_2, \ldots, \epsilon_k) \rangle$, and that there exists $j \geq 1$ such that $g_k(z_{k+j}) = 1$, but it is also such that $(\epsilon_1, \epsilon_2, \ldots, \epsilon_k, 0) \in \ker(V'_{k+1})$.

If $(\epsilon_1, \epsilon_2, \ldots, \epsilon_k, 0) \in \ker(V'_{k+1})$, then $(\epsilon_1, \epsilon_2, \ldots, \epsilon_k, 0) \cdot V'_{k+1} = \hat{0}$. Specifically, the last equation in the system of equations described by $(\epsilon_1, \epsilon_2, \ldots, \epsilon_k, 0) \cdot V'_{k+1} = \hat{0}$ is:

$$\epsilon_1 z_{k+j}^{\alpha_1} + \cdots + \epsilon_k z_{k+j}^{\alpha_k} + 0 \cdot z_{k+j}^{\alpha_{k+1}} = \epsilon_1 z_{k+j}^{\alpha_1} + \cdots + \epsilon_k z_{k+j}^{\alpha_k} = 0.$$

However, from our hypothesis, we have:

$$g_k(z_{k+j}) = 1 \iff g_k(z_{k+j}) = (\epsilon_1, \epsilon_2, \ldots, \epsilon_k) \begin{pmatrix} z_{k+j}^{\alpha_1} \\ z_{k+j}^{\alpha_2} \\ \vdots \\ z_{k+j}^{\alpha_k} \end{pmatrix} = \epsilon_1 z_{k+j}^{\alpha_1} + \cdots + \epsilon_k z_{k+j}^{\alpha_k} = 1.$$

This contradiction shows that it cannot be the case that $(\epsilon_1, \ldots, \epsilon_k, 0)$ is in the kernel of V'_{k+1}.

□

Proposition 4. *The sequence of matrices $(V_k)_k$, constructed as described in Proposition 3, ensures that:*

$$dim\,(ker\,(V_k)) \leq 1, \quad \forall k \in [1, |Z|].$$

Proof. We prove the result by induction:

- Base case. The initial matrix $V_1 = (1)$ has full rank, so $\dim(\ker(V_1)) = 0 \leq 1$.
- Induction step. We suppose that $\dim(\ker(V_k)) \leq 1$. We show that $\dim(\ker(V_{k+1})) \leq 1$.
  - If $\dim(\ker(V_k)) = 0$, adding a new row and column to form V_{k+1} can increase the kernel by at most 1. Hence, $\dim(\ker(V_{k+1})) \leq 1$.
  - If $\dim(\ker(V_k)) = 1$, let $\hat{z}$ be an element of $\ker(V_{k+1})$. Then either:
    - $\hat{z} = (\hat{\epsilon}, 0)$:
    $$\hat{z} \cdot V_{k+1} = 0 \implies \hat{\epsilon} \cdot V_k = 0 \implies \hat{\epsilon} \in \ker(V_k).$$
    By Proposition 3, $\hat{\epsilon}$ can only be a trivial solution.
    - $\hat{z} = (\hat{y}, 1)$, where $\hat{y} \in \mathbb{F}_2^k$: In this case $\hat{z} \cdot V_{k+1} = 0$ implies:
    $$\hat{y} \cdot V_k + (z_1^{\alpha_{k+1}}, \cdots, z_k^{\alpha_{k+1}}) = 0 \iff \hat{y} \cdot V_k = (z_1^{\alpha_{k+1}}, \cdots, z_k^{\alpha_{k+1}}), \tag{5}$$
    and
    $$\hat{y} \cdot (z_{k+1}^{\alpha_1}, \cdots, z_{k+1}^{\alpha_k})^T + z_{k+1}^{\alpha_{k+1}} = 0 \iff \hat{y} \cdot (z_{k+1}^{\alpha_1}, \cdots, z_{k+1}^{\alpha_k})^T = z_{k+1}^{\alpha_{k+1}}. \tag{6}$$

Since $\dim(\ker(V_k)) = 1$, $\text{rank}(V_k) = k - 1$. Hence, there exists at most one particular solution $\hat{y}_p$ for Equation 5. If $\hat{y}_p$ also solves Equation 6, then $\hat{y}_p$ is in $\ker(V_{k+1})$.

Since $(\hat{\epsilon}, 0)$ cannot be in $\ker(V_{k+1})$, the only possible element in $\ker(V_{k+1})$ is $\hat{y}_p$, if it exists. Therefore, $\dim(\ker(V_{k+1})) \leq 1$.

□

Remark 7. Notice that the definition of V'_{k+1} in Proposition 3, does not conflict with Definition 13, s constructing the sequence $(V_k)_k$ does not require a specific order of the elements in the support Z.

Proposition 5. *Let ker$(V_{E_k,Z_k}) = \langle \hat{g}_k \rangle = \langle (\epsilon_1, \ldots, \epsilon_k) \rangle$, where the corresponding Boolean function $g_k \in \mathcal{B}_n$ satisfies $g_k(x) = 0$, $\forall x \in Z$, but satisfies a $y \in S$ such that $g_k(y) = 1$.*

Define $E^* = (\alpha_1, \alpha_2, \ldots, \alpha_{k-1}, \alpha_{k+1})$, a vector constructed from E_k by replacing the last element α_k with α_{k+1}. Let $V_{k+1} = V_{E^*_{k+1},Z_k}$ be the matrix defined by replacing the monomial α_k in V_{E_k,Z_k}, with the monomial α_{k+1}:

$$V_{E^*_{k+1},Z_k} = \begin{pmatrix} & & z_k^{\alpha_1} \\ & E_{k-1}, V_{Z_{k-1}} & z_k^{\alpha_2} \\ & & \vdots \\ & & z_k^{\alpha_{k-1}} \\ z_1^{\alpha_{k+1}} & z_2^{\alpha_{k+1}} \cdots z_{k-1}^{\alpha_{k+1}} & z_k^{\alpha_{k+1}} \end{pmatrix}.$$

Then, either $\hat{g}_k \notin$ ker$(V_{E^*_{k+1},Z_k})$, or $\hat{g}_k$ is the only element (aside from $\hat{0}$) in the right kernel of $V_{E^*_{k+1},Z_k}$. Hence:

$$\dim\left(\ker(V_{E^*_{k+1},Z_k})\right) \leq 1.$$

Proof. Proposition 3 ensures that the kernel of $V_{E^*_{k+1},Z_k}$ does not contain elements derived from the kernel of $V_{E_{k-1},Z_{k-1}}$. Therefore, $V_{E_{k-1},Z_{k-1}}$ can be treated as a full-rank matrix. Modifying the last row of a matrix that contains a full-rank sub-matrix does not increase the size of the kernel. This is because the first k entries of the elements in the kernel of V_{E_k,Z_k} and $V_{E^*_{k+1},Z_k}$ are the same, with the only difference arising from the last row. Since this row is the only one that can introduce linear dependence, the dimension of the kernel of $V_{E^*_{k+1},Z_k}$ is bounded by 1. □

4.2 Using column echelon form matrices of the previous steps

At the iteration k, the matrix V_k already contains all the columns of the matrix from the previous step, V_{k-1}. This means the column echelon form of V_{k-1} is already embedded within V_k. Therefore, instead of recalculating the column echelon form for the entire matrix V_k, we only need to update it for the newly added column. This approach saves computation because we are focusing only on the last column rather than reprocessing the whole matrix.

Definition 14 (Column Echelon form). *Let $k_r, k_c \in \mathbb{N}^*$ and $M \in Mat_{k_r,k_c}$. We denote by $EF : Mat_{k_r,k_c} \to Mat_{k_r,k_c}$, the function transforming a matrix to its column echelon form.*

For each iteration k of the algorithm, the matrix V_k already includes the sub-matrix V_{k-1}. Thus, it is possible to compute the column echelon form of V_{k-1} and use it to derive the column echelon form of

V_k. This approach allows V_k to inherently contain $EF(V_{k-1})$, which is already in column echelon form, thereby reducing the computational steps required to determine the rank of V_k. However, to apply this method effectively, careful attention must be paid during the construction of the matrix. Since rows are appended to the matrix at each step, computing $EF(V_k)$ requires keeping track of all column operations performed while computing $EF(V_j)$ for all $j < k$. These operations must be applied before computing $EF(V_k)$. Remark 8 and Proposition 6 provide further justification for this claim.

Definition 15 (Equality up to row permutation). *Two matrices M_1 and M_2 are equal up-to-permutation if there exists a column permutation ρ such that*

$$M_1 = \rho M_2.$$

We denote it as $M_1 =_p M_2$

Remark 8. Let $k_r, k_c \in \mathbb{N}$ and $M \in Mat_{k_r,k_c}(\mathbb{F}_2^n)$. Let $M = L \cdot U$ where L a lower triangular matrix and U is an upper triangular matrix, then: $EF(M) =_p L$.

Proposition 6. *Let k be such that $1 < k < \min\left(|Z|, |E_{D_n^n}|\right)$ it holds that*

$$EF\left(V_k\right) =_p EF\left(\begin{pmatrix} & & z_k^{\alpha_1} \\ & EF\left(V_{k-1}\right) & z_k^{\alpha_2} \\ & & \vdots \\ & & z_k^{\alpha_{k-1}} \\ (z_1^{\alpha_k}\ z_2^{\alpha_k} \cdots z_{k-1}^{\alpha_k})P_{k-1} & & z_k^{\alpha_k} \end{pmatrix}\right).$$

where P_{k-1} is such that $U_{k-1}^{-1} = P_{k-1}$, where $V_{k-1} = L_{k-1}U_{k-1}$ with $EF(V_{k-1}) =_p L_{k-1}$

Proof. Let k be such that $1 < k < \min\left(|Z|, |E_{D_n^n}|\right)$. The LU-decomposition of V_{k-1} is

$$V_{k-1} = L_{k-1}U_{k-1},$$

where U_{k-1} is an upper triangular matrix and $L_{k-1} = EF\left(V_{k-1}\right)$ is the lower triangular matrix.

$$V_k = \begin{pmatrix} & & z_k^{\alpha_1} \\ & L_{k-1}U_{k-1} & z_k^{\alpha_2} \\ & & \vdots \\ & & z_k^{\alpha_{k-1}} \\ z_1^{\alpha_k}\ z_2^{\alpha_k} \cdots z_{k-1}^{\alpha_k} & & z_k^{\alpha_k} \end{pmatrix} = \begin{pmatrix} & & z_k^{\alpha_1} \\ & L_{k-1} & z_k^{\alpha_2} \\ & & \vdots \\ & & z_k^{\alpha_{k-1}} \\ (z_1^{\alpha_k}\ z_2^{\alpha_k} \cdots z_{k-1}^{\alpha_k})U_{k-1}^{-1} & & z_k^{\alpha_k} \end{pmatrix} \begin{pmatrix} & & 0 \\ & U_{k-1} & 0 \\ & & \vdots \\ & & 0 \\ 0\ 0 \cdots 0 & & 1 \end{pmatrix}.$$

Since U_{k-1} is an upper triangular matrix, $\begin{pmatrix} & & 0 \\ & U_{k-1} & 0 \\ & & \vdots \\ & & 0 \\ 0\,0 & \cdots & 0\,1 \end{pmatrix}$ is also an upper triangular matrix, and therefore

$$EF\left(V_k\right) =_p EF\left(\begin{pmatrix} & & z_k^{\alpha_1} \\ & L_{k-1} & z_k^{\alpha_2} \\ & & \vdots \\ & & z_k^{\alpha_{k-1}} \\ (z_1^{\alpha_k}\ z_2^{\alpha_k}\ \cdots\ z_{k-1}^{\alpha_k})U_{k-1}^{-1} & & z_k^{\alpha_k} \end{pmatrix}\right).$$

Taking $L_{k-1} =_p EF\left(V_{k-1}\right)$ and $P_{k-1} = U_{k-1}^{-1}$:

$$EF\left(V_k\right) =_p EF\left(\begin{pmatrix} & & z_k^{\alpha_1} \\ & EF\left(V_{k-1}\right) & z_k^{\alpha_2} \\ & & \vdots \\ & & z_k^{\alpha_{k-1}} \\ (z_1^{\alpha_k}\ z_2^{\alpha_k}\ \cdots\ z_{k-1}^{\alpha_k})P_{k-1} & & z_k^{\alpha_k} \end{pmatrix}\right).$$

□

Remark 9. P_{k-1} is the matrix corresponding to the history of operations applied to the columns of V_{k-1} to be able to obtain $EF\left(V_{k-1}\right)$.

As a consequence of Remark 9, it can be seen that when calculating the column echelon form of the matrix V_k using P_{k-1}, we construct the application P_k which can be used to compute the column echelon form of V_{k+1} in the next iteration. Remark 10 shows this more formally.

Remark 10.

$$EF\left(V_k\right) = EF\left(\begin{pmatrix} & & z_k^{\alpha_1} \\ & EF\left(V_{k-1}\right) & z_k^{\alpha_2} \\ & & \vdots \\ & & z_k^{\alpha_{k-1}} \\ (z_1^{\alpha_k}\ z_2^{\alpha_k}\ \cdots\ z_{k-1}^{\alpha_k})P_{k-1} & & z_k^{\alpha_k} \end{pmatrix}\right) =_p \begin{pmatrix} & & 0 \\ & EF\left(V_{k-1}\right) & 0 \\ & & \vdots \\ & & 0 \\ (z_1^{\alpha_k}\ z_2^{\alpha_k}\ \cdots\ z_{k-1}^{\alpha_k}\ z_k^{\alpha_k})P_k & & \end{pmatrix}.$$

Algorithm 2 shows the procedure to calculate the degree of the minimal degree non-zero-annihilator restricted to $S \subseteq \mathbb{F}_2^n$ of a function $f^{(o)} \in \{f, f+1\}$, while Algorithm 3 shows the full procedure to determine the restricted algebraic immunity of f in S, calling Algorithm 2 for f and $f+1$ in parallel.

Algorithm 3 includes the additional function FIND_DEG_SMALLEST_S_ANNIHILATOR_SEQ which triggers the same process as Algorithm 2, but applies it sequentially to f and $f+1$, to avoid waiting for both processes to complete simultaneously. This function is triggered only when $\mathsf{supp}(f) \cap S$ and $\mathsf{supp}(f+1) \cap S$ differ significantly in cardinality, as in such cases, the probability of one of the two processes terminating quickly is high.

Algorithm 2 Algorithm to find the degree of the minimum-degree non-zero-annihilator restricted on S of a function $f^{(o)}$.

Input: restricted set $S \subset \mathbb{F}_2^n$, $Z = supp\left(f^{(o)}\right) \cap S$ and $E = \{\alpha_1, \cdots, \alpha_{D_n^n}\}$

Output: Degree of minimal degree non-zero-annihilator restricted on S of $f^{(o)}$.

$V \leftarrow (1)$;
$ops \leftarrow [\,]$;
$k \leftarrow 2$;
while $k \leq |Z|$ **do**
    $V_{previous} \leftarrow V$;
    $(v_1, \cdots, v_{k-1}) \leftarrow$ APPLY_OPERATIONS $\left(\left(z_1^{\alpha_k}, \cdots, z_{k-1}^{\alpha_k}\right), ops\right)$[3];
    $M \leftarrow \begin{pmatrix} & & z_k^{\alpha_1} \\ & V & z_k^{\alpha_2} \\ & & \vdots \\ & & z_k^{\alpha_{k-1}} \\ v_1\ v_1 \cdots v_{k-1} & & z_k^{\alpha_k} \end{pmatrix}$;
    $V, ops_k \leftarrow$ EF_LAST_COL(M)[4];
    if $rank(V) < k$ **then**
        $k \leftarrow kernel\,(V)\,.base()$;
        $\hat{g} \leftarrow k[0]$;
        $nonVanishingPointIndex, vanishingFlag \leftarrow Null, True$;
        for all $z_{idx} \in (Z \backslash Z_k)$ **do**
            $vanishCheck \leftarrow$ VANISHES_ON$(\hat{g}, z_{idx})$[5];
            if $vanishCheck = False$ **then**
                $vanishingFlag \leftarrow False$;
                $nonVanishingPointIndex \leftarrow idx$
                break;
            end if
        end for
        if $vanishingFlag = True$ **then**
            for all $z_c \in S \backslash Z$ **do**
                $vanishCheckOnS \leftarrow$ VANISHES_ON$(\hat{g}, z_c)$;
                if $vanishCheckOnS = False$ **then**
                    return $w_H\,(\alpha_i)$;
                end if
            end for
            $V \leftarrow V_{previous}$;
            REMOVE(E, α_i);
            continue; *// Go to the next iteration.*
        else
            $zIndex \leftarrow k + nonVanishingPointIndex$;
            $Z \leftarrow$ SWAP(z_{k+1}, z_{zIndex})[6];
        end if
        $ops \leftarrow ops \,||\, ops_k$;
        $k \leftarrow k + 1$;
    end if
end while
// The algorithm continues in the next page.

40: $S, ops_S \leftarrow \text{EF}\left(\begin{pmatrix} s_1^{\alpha_1} & s_2^{\alpha_1} & \cdots & s_{|S|}^{\alpha_1} \\ \vdots & \vdots & \ddots & \vdots \\ s_1^{\alpha_k} & s_2^{\alpha_k} & \cdots & s_{|S|}^{\alpha_k} \end{pmatrix}\right)$;
41: $r \leftarrow rank(S)$;
42: **if** $rank(V) < r$ **then**
43: **return** $w_H(\alpha_k)$;
44: **end if**
45: $r_V \leftarrow rank(V)$;
46: $fullRank \leftarrow False$;
47: **while** $r \leq \lfloor |S|/2 \rfloor$ **do**
48: **if** $fullRank = False$ and $r_v! = k$ **then**
49: $(v_1, \cdots, v_{k-1}) \leftarrow$ APPLY_OPERATIONS $\left(\left(z_1^{\alpha_k}, \cdots, z_{k-1}^{\alpha_k}\right), ops\right)$
50: $V \leftarrow \begin{pmatrix} V \\ v_1\ v_2 \cdots v_{k-1}\ z_k^{\alpha_k} \end{pmatrix}$;
51: $r_V \leftarrow rank(V)$;
52: $fullRank \leftarrow True$;
53: **end if**
54: $(v_1, \cdots, v_{k-1}) \leftarrow$ APPLY_OPERATIONS $\left(\left(s_1^{\alpha_k}, \cdots, s_{k-1}^{\alpha_k}\right), ops_S\right)$;
55: $S \leftarrow \begin{pmatrix} S \\ s_1\ s_2 \cdots s_{k-1}\ s_k^{\alpha_k} \end{pmatrix}$;
56: $r \leftarrow rank(S)$;
57: **if** $r_V < r$ **then**
58: **return** $w_H(\alpha_k)$;
59: **end if**
60: **end while**

Remark 11. To keep the pseudocode in line with the theory, the elements in the vector or sets are often parsed by name instead of by index, namely let $Z = (z_1, z_2, \cdots, z_k)$ we consider that $Z[i-1] = z_i$. Therefore, we have to clarify our notation in the case in which two vector elements get swapped. In the case of Algorithm 2, we consider that the swap generates a new vector with the name of the variables adapted with their index position in the vector to keep satisfying the condition $Z[i-1] = z_i$. Meaning that

$$\text{SWAP}(z_i, z_j) \rightarrow (z_{\sigma(1)}, \cdots, z_{\sigma(i)}, \cdots, z_{\sigma(j)}, \cdots, z_{\sigma(k)}) = Z' = (z_1', \cdots, z_i', \cdots, z_j', \cdots, z_k')$$

where σ is a permutation describing the swap:

$$\sigma = \begin{pmatrix} 1 \cdots i \cdots j \cdots k \\ 1 \cdots j \cdots i \cdots k \end{pmatrix}.$$

[3] APPLY_OPERATIONS is a function taking a vector and the list of column operations described by P_{k-1} in Remark 9

[4] The implementation of EF_LAST_COL is different from the implementation of EF, as it uses the fact that it only needs to column-echelon the last row and column added instead of the full matrix.

[5] VANISHED_ON is a function taking as input the vector of the ANF coefficient of a Boolean function $g \in \mathcal{B}_n$ and a point z on which to evaluate the corresponding Boolean function g. If g vanished is z it returns $True$ otherwise it returns $False$.

[6] See Remark 11

Algorithm 3 Algorithm to compute the algebraic immunity of a function $f \in \mathcal{B}_n$ restricted to S

Input: $x \in \mathbb{F}_2^{2^n}$ evaluation of the truth table of $f \in \mathcal{B}_n$ and the restricted set $S \subseteq \mathbb{F}_2^n$.
Output: $\mathsf{AI}_S(f)$

```
Z ← [];
Z_c ← [];
for k ← 0 to 2^n − 1 do
    if k ∉ S then
        continue
    end if
    if x[k] = 1 then
        Z ← Z||(x[k])_2;
    else
        Z_c ← Z_c||(x[k])_2;
    end if
end for
if Z = [ ] or Z_c = [ ] then
    return 0;
end if

E_{D_n^n} ← CONSTRUCT_MONOMIALS(n = n, d = n)[7];
r ← |Z|/|Z_c| if |Z| < |Z_c| else |Z_c|/|Z|;
if r ≥ ratio then
    immunity_f, immunity_{f+1} ← (
 FIND_DEG_SMALLEST_S_ANNIHILATOR(Z = Z, E = E_{D_n^n}, S = S)
 | FIND_DEG_SMALLEST_S_ANNIHILATOR(Z = Z_c, E = E_{D_n^n}, S = S)
 );
    return MIN(immunity_f, immunity_{f+1});
else
    return FIND_DEG_SMALLEST_S_ANNIHILATOR_SEQ(Z = Z, Z_c, E = E_{D_n^n}, S = S))[8];
end if
```

5 Experiments and applications to W(A)PB functions

In this section, we apply the algorithms to the weightwise perfectly balanced (WPB) Boolean functions with the intent to study their algebraic immunity restricted to subsets of constant Hamming weight. In particular, we determine the probability distributions of the AI_k of WPB functions and we compute the AI_k of some WPB Boolean functions already studied. The source code for the implementation of Algorithm 1

[7] CONSTRUCT_MONOMIALS generates all monomials up to the gree d with n variables. $E_{D_n^n}$ may also be pre-computed and the function CONSTRUCT_MONOMIALS removed from the algorithm.

[8] The implementation of the function FIND_DEG_SMALLEST_S_ANNIHILATOR_SEQ is similar to the function FIND_DEG_SMALLEST_S_ANNIHILATOR described in Algorithm 2 with the difference that the degree of the minimal degree restricted non-zero annihilator of f or of $f+1$ is found sequentially instead of in parallel.

and Algorithm 3, along with the probability distributions and the values of AI_k for the published functions, can be found in the `https://github.com/LucaBonamino/restricted_algebraic_immunity`.

5.1 Comparison between Algorithm 1 and Algorithm 3 on WAPB functions

In this part, we compare the execution times of Algorithm 1 and Algorithm 3 by running both algorithms on multiple WAPB functions restricted to the largest set $E_{\lceil n/2 \rceil, n}$, with different numbers of variables. Specifically, the experiment is done by running the algorithms with the number of variables n going from 1 to 12. For both algorithms, if $n \leq 4$, we calculate the average over the $\mathsf{AI}_{\lceil n/2 \rceil}$ of all functions in $\mathcal{WAPB}_n$, whereas for $n > 4$, we calculate the average over the $\mathsf{AI}_{\lceil n/2 \rceil}$ of a random sample of 10^4 $\mathcal{WAPB}_n$ functions.

Figure 1 shows the mean execution time per value of n of the two algorithms. We denote by $T\left(G^{\leq n_{max}}\right) = T\left(G^{\leq 12}\right)$ the necessary time to pre-compute all the necessary Reed-Muller codes generator matrices following Equation (3): we recall that it is the necessary time to compute the sequence $(G_{n,n})_{n \in [1, 12 = n_{max}]}$, not only the generator matrix $G_{n,n}$ for the current iteration n. We notice in fact that $T\left(G^{\leq 12}\right)$ is constant, that is because the computation occurs before beginning the experiment since it is expected that the Algorithm 1 is able to run for any number of variable n up to an upper bound n_{max}, without having to compute the Reed-Muller code for it.

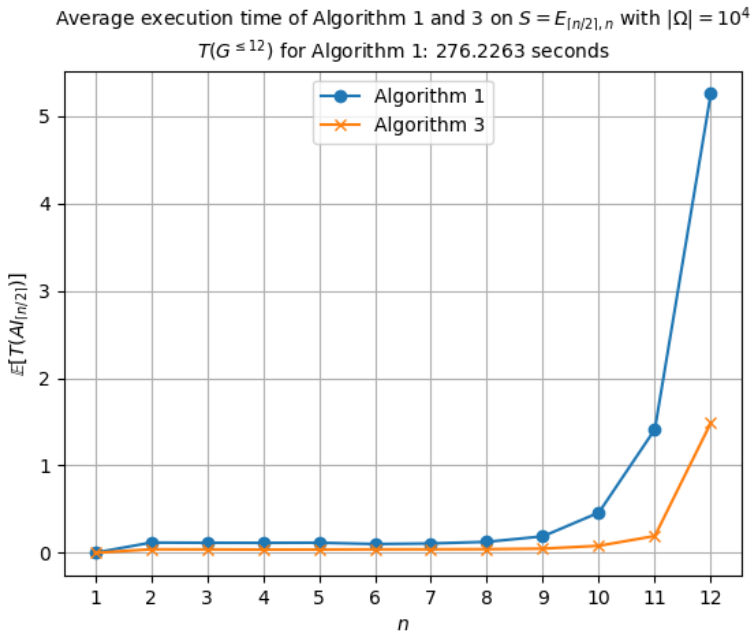

Fig. 1: Average time execution per number of variables of Algorithms 1 and 3 over 10^4 random $\mathcal{WAPB}_n$ functions on the set $E_{\lceil n/2 \rceil, n}$. $T\left(G^{\leq 12}\right)$ gives the time of the pre-computation of the sequence of all Reed-Muller matrices up to $G_{12,12}$.

The pre-computation times of $G^{\leq n_{max}}$ for Algorithm 1 are given in the Table 1.

From Figure 1, we observe that Algorithm 3 outperforms Algorithm 1 for $n \leq 12$, even without considering the pre-computation time required for $G^{\leq n_{max}}$ and $D^{\leq n_{max}}$ in Algorithm 1. Furthermore,

$G^{\leq 1}$	$G^{\leq 2}$	$G^{\leq 3}$	$G^{\leq 4}$	$G^{\leq 5}$	$G^{\leq 6}$	$G^{\leq 7}$	$G^{\leq 8}$	$G^{\leq 9}$	$G^{\leq 10}$	$G^{\leq 11}$	$G^{\leq 12}$
0.0126	0.0042	0.0017	0.004	0.0149	0.0470	0.1938	0.806	3.5456	14.9168	65.5014	276.2263

Table 1: Pre-computation time (in seconds) for $G^{\leq n_{max}}$ for n_{max} from 1 to 12.

Algorithm 1 becomes impractical for $n > 12$ due to the excessive complexity of pre-computation. In contrast, Algorithm 3 remains feasible for larger values of n.

Both implementations of the algorithms used to generate the plot in Figure 1 have been developed in *Python*, with the following enhancements:

- The linear algebra operations in Algorithm 1 are performed using the *SageMath* library, while those in Algorithm 3 are executed using an ad-hoc custom *Python* package implemented in *Rust*. This setup ensures comparability between the two implementations, as the *SageMath* library is highly optimized for linear algebra operations.
- The computation of $G^{\leq n_{max}}$ for Algorithm 1 is performed using the *generator_matrix* method of the *SageMath* object *BinaryReedMullerCodes*. While this computation could be more efficient if implemented in *Rust*, the current overall implementation still faces scalability issues. Specifically, for $n \geq 16$, the algorithm becomes impractical because $G_{16,16}$ cannot be stored in a native *Python* list or a *SageMath* matrix object. However, if the entire algorithm were implemented in *Rust*, it could potentially be feasible. Consequently, for larger values of n, Algorithm 1 is not usable, while Algorithm 3 remains viable.

5.2 Determination of AI_k distribution of WPB functions

Similar to the approach in [GM23a] for analyzing the weightwise nonlinearity NL_k, we determine the discrete probability distribution of AI_k for weightwise perfectly balanced Boolean functions. This distribution describes the probability that a function $f \in \mathcal{WPB}_m$ has a specific algebraic immunity restricted to the slice $\mathsf{E}_{k,n}$. A formal definition is provided in Definition 16:

Definition 16. *Let $m \in \mathbb{N}^*$, $n = 2^m$ and $k \in [1, n-1]$, we define the discrete probability distribution of AI_k as:*

$$p_{\mathsf{AI}_k}(x) = \frac{\left|\{f \in \mathcal{WPB}_m \mid \mathsf{AI}_k(f) = x\}\right|}{|\mathcal{WPB}_m|}.$$

We aim to determine this distribution for $m = 2, 3$, and 4. However, the number of due of WPB functions increases fast with the growth of m, making this computation infeasible for $m > 2$. In fact, as stated in [GM23a], $|\mathcal{WPB}_2| = 720$, while $|\mathcal{WPB}_3| > 2^{243}$ and $|\mathcal{WPB}_4| > 2^{65452}$.
Therefore, we have to take a different approach for $m > 2$ by generating a certain number of samples from the set $\mathcal{WPB}_m$ and calculating the probability over it. The larger the sample size we select, the better we approximate the distribution.

Definition 17. *We define the AI_k distribution over a sample Ω as:*

$$p_{\Omega,\mathsf{AI}_k}(x) = \frac{\left|\{f \in \Omega \mid \mathsf{AI}_k(f) = x\}\right|}{|\Omega|}.$$

Determination of the full distribution for m=2 For $m = 2 \Rightarrow n = 4$, it is possible to determine the full probability distribution from Definition 16 without using the random sampling strategy. The resulting plots can be found on Figure 4 and Figure 2 while the corresponding probability distribution table is presented in Table 2a.

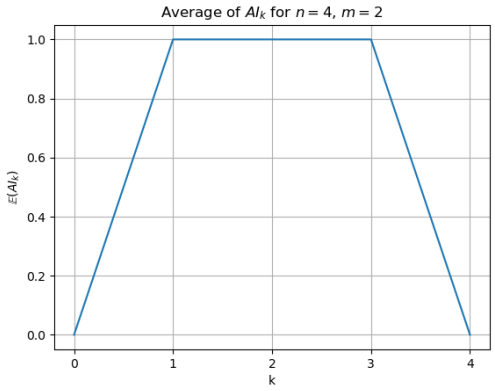

Fig. 2: Mean of AI_k over $\mathcal{WPB}_2$.

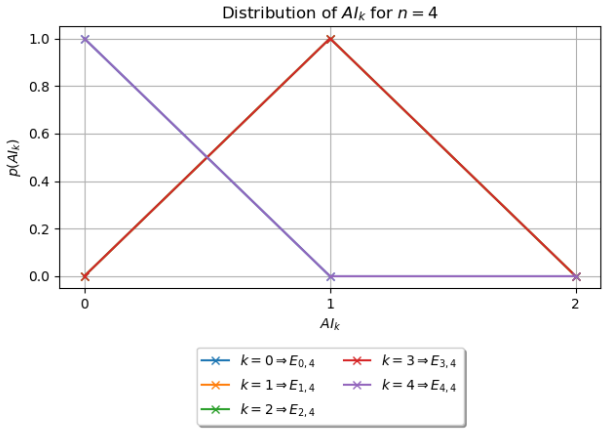

Fig. 3: Probability distribution of AI_k over $\mathcal{WPB}_2$.

Fig. 4: Mean of AI_k and probability distribution of AI_k from Definition 16 of 4-variable WPB functions.

Random Sampling In practice, random sampling is performed by generating a set $\Omega_k = \{\omega_1^{(k)}, \cdots, \omega_{|\Omega|}^{(k)}\}$ consisting of $|\Omega|$ random balanced sub-truth tables for each slice$\mathsf{E}_{k,n}$. Then, for each slice $\mathsf{E}_{k,n}$, the AI_k is computed for each restricted set in Ω_k the frequency of each particular AI_k value is recorded. Depending on the value of m and the chosen sample size, the number of sub-truth tables generated may be too large to fit in memory simultaneously. To address this, it is crucial to avoid storing all elements in memory at once. Instead, they should be processed one by one, either by computing sequentially or by using iterators.
The following procedure is done for each slice $\mathsf{E}_{k,n}$:

1. Initialize a hash table of all possible values of AI_k.
2. For each sub-truth table in $\omega_i^{(k)} \in \Omega_k$ (note that $\omega_i^{(k)}$ is either generated on the fly by an iterator or by having the source code for the generation of $\omega_i^{(k)}$ in this loop)[9]:
   (a) Run Algorithm 3 with $S = \omega_i^{(k)}$.
   (b) Increment the value in the hash table corresponding to the output of the previous step.
3. Divide each value in the hash table by $|\Omega|$.

Results for $n = 8$ and sample size$= 2^{15}$. Figure 6 represents the AI_k distribution, Table 2b shows the corresponding values, and Figure 5 the average values. We observe that for $n = 4$ all WPB functions have the same values of AI_k, for every k. For $n = 8$, there are WPB functions with different restricted AI_k, for example with $k = 4$ we observe functions reaching the lowest value (2, such as families exhibited in [GM23a]) and some reaching 3 which is optimal for this size.

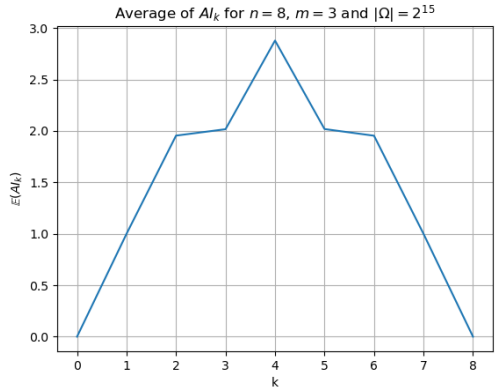

Fig. 5: Estimated average of AI_k over $\mathcal{WPB}_3$.

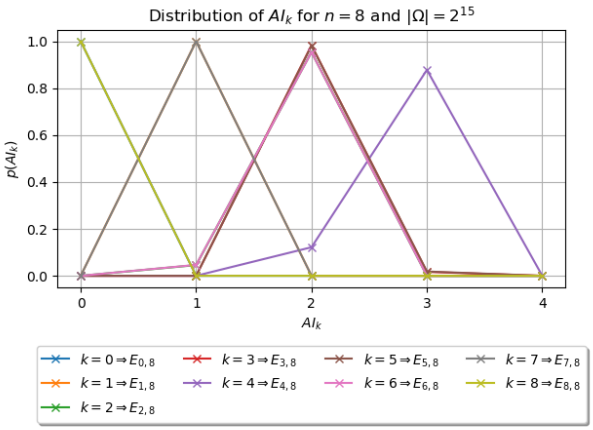

Fig. 6: Estimated Probability distribution of AI_k over $\mathcal{WPB}_3$.

Fig. 7: Estimated average of AI_k and probability distribution of AI_k over 8-variable WPB functions, with 2^{15} samples.

Results for $n = 16$ and sample size$= 2^{10}$. Figure 9 represents the AI_k distribution, Table 2c shows the corresponding values and Figure 8 the average values. We observe that there are WPB functions with different restricted AI_k for this value of n, and that for the central values ok k, all functions we sampled reach the optimal value of AI_k.

The average execution times for the AI_k distributions with $n = 4$, $n = 8$ are $n = 16$ are displayed in Tables 3a, 3b and 3c.

[9] Step 2 may be executed in parallel on the $\omega_i^{(k)}$s.

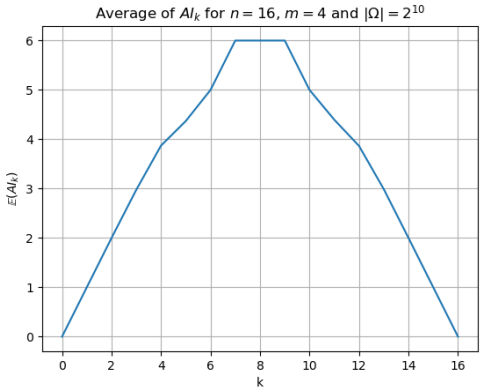

Fig. 8: Estimated average of AI_k over $\mathcal{WPB}_4$.

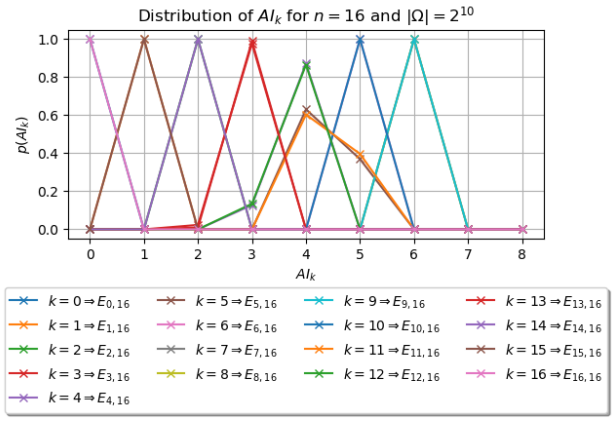

Fig. 9: Estimated Probability distribution of AI_k over $\mathcal{WPB}_4$.

Fig. 10: Estimated average of AI_k and probability distribution of AI_k over 8-variable WPB functions, with 2^{10} samples.

Table 2: AI_k probability distribution of $\mathcal{WPB}_2$, $\mathcal{WPB}_3$ and $\mathcal{WPB}_4$.

(a) Probability distribution p_{AI_k} for $n = 4$, $m = 2$.

$p(x)$, k	0	1	2	3	4
0	1.00	0.00	0.00	0.00	1.00
1	0.00	1.00	0.00	1.00	0.00
2	0.00	0.00	1.00	0.00	0.00

(b) Probability distribution p_{Ω,AI_k} for $n = 8$, $m = 3$, $\Omega = 2^{15}$.

$p(x)$, k	0	1	2	3	4	5	6	7	8
0	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
1	0.00	1.00	0.00	0.00	0.00	0.00	0.05	1.00	0.00
2	0.00	0.00	1.00	0.98	0.00	0.98	0.95	0.00	0.00
3	0.00	0.00	0.00	0.02	0.13	0.02	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.87	0.00	0.00	0.00	0.00

(c) Probability distribution p_{Ω,AI_k} for $n = 16$, $m = 4$, $\Omega = 2^{10}$.

$p(x)$, k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
1	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
2	0.00	0.00	1.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	1.00	0.00	0.00
3	0.00	0.00	0.00	0.98	0.13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13	0.99	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.87	0.63	0.00	0.00	0.00	0.00	0.00	0.60	0.87	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.37	1.00	0.00	0.00	0.00	1.00	0.40	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 3: Averages execution times and $\mathsf{AI}k$ of $\mathcal{WPB}_2$, $\mathcal{WPB}_3$ and $\mathcal{WPB}_4$ functions.

(a) Average execution time and AI_k for $n = 4$, $m = 2$.

k	0	1	2	3	4
$\mathbb{E}[T(AI_k)]$	0.000053	0.052649	0.048703	0.058630	0.000006
$\mathbb{E}[AI_k]$	0.00	1.00	1.00	1.00	0.00

(b) Average execution time and AI_k for $n = 8$, $m = 3$ and sampleSize $= 2^{15}$.

k	0	1	2	3	4	5	6	7	8
$\mathbb{E}[T(AI_k)]$	4.768×10^{-6}	0.0756	0.086	0.094	0.090	0.102	0.0924	0.093818	1.621×10^{-5}
$\mathbb{E}[AI_k]$	0.00	1.00	1.95	2.02	2.89	2.02	1.95	1.00	0.00

(c) Average execution time and AI_k for $n = 16$, $m = 4$ and sampleSize $= 2^{10}$.

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\mathbb{E}[T(AI_k)]$	$0.10 * 10^{-4}$	0.334	0.342	0.41	4.01	91.81	636.37	2838.92	6384.38	3111.12	732.50	118.96	5.23	0.43	0.35	0.35	$0.15 * 10^{-4}$
$\mathbb{E}[AI_k]$	0.00	1.00	2.00	2.98	3.87	4.37	5.00	6.00	6.00	6.00	5.00	4.40	3.87	2.99	2.00	1.00	0.00

5.3 Determination of AI_k of known WPB families

In this part, we determine the AI_ks of the function introduced in [ZJZQ23], [DM24], [TL19] and in [CMR17] for $n = 8$ and 16 variables.

The results are summarized in Table 4. Of the four WPB functions, the one in [TL19] is the one that would resist the best to algebraic attacks (with constant Hamming weight inputs), as it achieves higher AI_k values for both $n = 8$ and $n = 16$. In contrast, the one in [ZJZQ23], would be the least resistant, as it has the overall lower AI_k for both $n = 8$ and $n = 16$. Specifically:

- $n = 8$. The function in [ZJZQ23] has a constant AI_k of 2 for $k \in [2, 6]$ and the functions in [CMR17] a constant AI_k of 2 for $k \in [3, 6]$, whereas the functions in [DM24] and in [TL19] achieve an AI_4 of 3.
- $n = 16$. The function in [ZJZQ23] again has a constant AI_k of 2 for $k \in [2, 14]$. In contrast, the function in [TL19] achieves an AI_k of 3 for $k = 4, 5, 11$ and 12, an AI_k of 4 for $k = 6, 7, 9$ and 10 and an AI_k of 5 for $k = 8$. The functions [DM24] and in [CMR17], both reach an AI_k of 3 but for $k \in [4, 12]$ and $k \in [5, 13]$ respectively.

The maximum execution times for [DM24], [ZJZQ23], [TL19] and [CMR17] with $n = 8$ occur with $k = 4$ and are around 0.014, 0.029, 0.011 and 0.012 seconds, respectively.
For $n = 16$, the maximum execution times for [ZJZQ23], [TL19] and [CMR17] occur with $k = 8$ and are 5.898671, 126.627561 and 12.048051 seconds respectively. In contrast, for [DM24], the maximum execution time occurs for $k = 9$ and is 7.935819 seconds.

Table 4: AI_k of the construction from [DM24], from [ZJZQ23], from [TL19] and from [CMR17] with $m = 3$ ($n = 8$) and $m = 4$ ($n = 16$).

[DM24]																	
n, k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
8	0	1	1	2	3	2	2	1	0								
16	0	1	1	2	3	3	3	3	3	3	3	3	3	2	2	1	0
[ZJZQ23]																	
n, k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
8	0	1	2	2	2	2	2	1	0								
16	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2	1	0
[TL19]																	
n, k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
8	0	1	2	2	3	2	2	1	0								
16	0	1	2	2	3	3	4	4	5	4	4	3	3	2	2	1	0
[CMR17]																	
n, k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
8	0	1	1	2	2	2	2	1	0								
16	0	1	1	2	2	3	3	3	3	3	3	3	3	3	2	1	0

In Table 5, we summarize the results on AI_k for WPB functions with 8 variables, including both our findings and those from the state of the art. Similarly, Table 6 presents the corresponding results for functions with 16 variables. We observe that most of the studied families exhibit lower weightwise algebraic immunity than a randomly chosen WPB function, with the exception of the field order construction [Méa24].

Function	AI_2	AI_3	AI_4	AI_5	AI_6
[DM24]	1	2	2	2	1
[ZJZQ23]	2	2	2	2	2
[TL19]	2	2	3	2	2
[CMR17]	1	2	2	2	2
Estimated average	1.95	2.02	2.87	2.02	1.95
Field order [Méa24]	1	2	3	2	2
Upper bound	2	3	3	3	2

Table 5: AI_k of WPB functions in 8 variables.

Function	AI_2	AI_3	AI_4	AI_5	AI_6	AI_7	AI_8	AI_9	AI_{10}	AI_{11}	AI_{12}	AI_{13}	AI_{14}
[DM24]	1	2	3	3	3	3	3	3	3	3	2	2	0
[ZJZQ23]	2	2	2	2	2	2	2	2	2	2	2	2	2
[TL19]	2	2	3	3	4	4	5	4	4	3	3	2	2
[CMR17]	1	2	2	3	3	3	3	3	3	3	3	3	2
Estimated average	2.00	2.99	3.89	4.37	5.00	6.00	6.00	6.00	5.00	4.40	3.87	2.99	2.00
Field order [Méa24]	2	3	4	4	5	6	6	6	5	4	3	2	2
Upper bound	2	3	4	5	5	6	6	6	5	5	4	3	2

Table 6: AI_k of WPB functions in 16 variables.

6 Conclusion

In this article, we formalized the main theoretical foundations and provided algorithms for computing restricted algebraic immunity over any set $S \subseteq \mathbb{F}_2^n$, using both the Reed-Muller and iterative approaches. We compared the practical complexity of these algorithms and showed that the iterative approach is more efficient. Additionally, we applied our methods to compute or estimate, for the first time, the distribution of AI_k for WPB functions with 4, 8, and 16 variables. Furthermore, we determined the values of AI_k for various families of WPB functions introduced in previous works and showed that, in most cases, these functions exhibit lower weightwise algebraic immunity than a randomly chosen WPB function.

By formalizing the theoretical background and providing both algorithms and implementations for computing restricted algebraic immunity, we hope this work will contribute to further research and the practical implementation of algebraic attack variants.

7 Acknowledgments

Pierrick Méaux was supported by the ERC Advanced Grant no. 787390.

References

ACG+06. Frederik Armknecht, Claude Carlet, Philippe Gaborit, Simon Künzli, Willi Meier, and Olivier Ruatta. Efficient computation of algebraic immunity for algebraic and fast algebraic attacks. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2006.

AL18. Benny Applebaum and Shachar Lovett. Algebraic attacks against random local functions and their countermeasures. *SIAM J. Comput.*, pages 52–79, 2018.

Arm04. Frederik Armknecht. Improving fast algebraic attacks. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2004.

Car07. Claude Carlet. Constructing balanced functions with optimum algebraic immunity. In *IEEE International Symposium on Information Theory, ISIT 2007, Nice, France, June 24-29, 2007*, pages 451–455. IEEE, 2007.

Car21. Claude Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021.

CF08. Claude Carlet and Keqin Feng. An infinite class of balanced functions with optimal algebraic immunity, good immunity to fast algebraic attacks and good nonlinearity. In *Advances in Cryptology - ASIACRYPT 2008*, pages 425–440, 2008.

CM03. Nicolas T. Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

CM13. Claude Carlet and Brahim Merabet. Asymptotic lower bound on the algebraic immunity of random balanced multi-output boolean functions. *Adv. Math. Commun.*, 7(2):197–217, 2013.

CMR17. Claude Carlet, Pierrick Méaux, and Yann Rotella. Boolean functions with restricted input and their robustness; application to the FLIP cipher. *IACR Trans. Symmetric Cryptol.*, 2017(3), 2017.

Cou03. Nicolas T. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer, 2003.

Did06. F. Didier. A new upper bound on the block error probability after decoding over the erasure channel. *IEEE Trans. Inf. Theory*, 52(10):4496–4503, 2006.

DM24. Deepak Kumar Dalai and Krishna Mallick. A class of weightwise almost perfectly balanced boolean functions. *Advances in Mathematics of Communications*, 18(2):480–504, 2024.

DMM24. Deepak Kumar Dalai, Krishna Mallick, and Pierrick Méaux. Weightwise almost perfectly balanced functions, construction from A permutation group action view. *IACR Cryptol. ePrint Arch.*, page 2068, 2024.

DMS06. Deepak Kumar Dalai, Subhamoy Maitra, and Sumanta Sarkar. Basic theory in construction of boolean functions with maximum possible annihilator immunity. *Designs, Codes and Cryptography*, 2006.

DT06. Frédéric Didier and Jean-Pierre Tillich. Computing the algebraic immunity efficiently. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 359–374. Springer, 2006.

GM22a. Agnese Gini and Pierrick Méaux. On the weightwise nonlinearity of weightwise perfectly balanced functions. *Discret. Appl. Math.*, 322:320–341, 2022.

GM22b. Agnese Gini and Pierrick Méaux. Weightwise almost perfectly balanced functions: Secondary constructions for all n and better weightwise nonlinearities. In Takanori Isobe and Santanu Sarkar, editors, *Progress in Cryptology - INDOCRYPT 2022 - 23rd International Conference on Cryptology in India, Kolkata, India, December 11-14, 2022, Proceedings*, volume 13774 of *Lecture Notes in Computer Science*, pages 492–514. Springer, 2022.

GM23a. Agnese Gini and Pierrick Méaux. On the algebraic immunity of weightwise perfectly balanced functions. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *Progress in Cryptology - LATINCRYPT 2023 - 8th International Conference on Cryptology and Information Security in Latin America, LATINCRYPT 2023, Quito, Ecuador, October 3-6, 2023, Proceedings*, volume 14168 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2023.

GM23b. Agnese Gini and Pierrick Méaux. Weightwise perfectly balanced functions and nonlinearity. In Said El Hajji, Sihem Mesnager, and El Mamoun Souidi, editors, *Codes, Cryptology and Information Security - 4th International Conference, C2SI 2023, Rabat, Morocco, May 29-31, 2023, Proceedings*, volume 13874 of *Lecture Notes in Computer Science*, pages 338–359. Springer, 2023.

GM24. Agnese Gini and Pierrick Méaux. S_0-equivalence classes, a new direction to find better weightwise perfectly balanced functions, and more. *Cryptogr. Commun.*, 16(6):1211–1234, 2024.

Gol00. Oded Goldreich. Candidate one-way functions based on expander graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(90), 2000.

GS22. Xiaoqi Guo and Sihong Su. Construction of weightwise almost perfectly balanced boolean functions on an arbitrary number of variables. *Discrete Applied Mathematics*, 307:102–114, 2022.

HR04. Philip Hawkes and Gregory G. Rose. Rewriting variables: The complexity of fast algebraic attacks on stream ciphers. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 390–406. Springer, 2004.

LM19. Jian Liu and Sihem Mesnager. Weightwise perfectly balanced functions with high weightwise nonlinearity profile. *Des. Codes Cryptogr.*, 87(8):1797–1813, 2019.

LS20. Jingjing Li and Sihong Su. Construction of weightwise perfectly balanced boolean functions with high weightwise nonlinearity. *Discret. Appl. Math.*, 279:218–227, 2020.

MCJS19. Pierrick Méaux, Claude Carlet, Anthony Journault, and François-Xavier Standaert. Improved filter permutators for efficient FHE: better instances and implementations. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology - INDOCRYPT*, volume 11898 of *LNCS*, pages 68–91. Springer, 2019.

Méa22. Pierrick Méaux. On the algebraic immunity of direct sum constructions. *Discret. Appl. Math.*, 320:223–234, 2022.

Méa24. Pierrick Méaux. Weightwise (almost) perfectly balanced functions based on total orders. *IACR Cryptol. ePrint Arch.*, page 647, 2024. to appear at Selected Area in Cryptography 2024.

MJSC16. Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In *Advances in Cryptology – EUROCRYPT 2016*, pages 311–343, 2016.

MKCL22. Sara Mandujano, Juan Carlos Ku Cauich, and Adriana Lara. Studying special operators for the application of evolutionary algorithms in the seek of optimal boolean functions for cryptography. In Obdulia Pichardo Lagunas, Juan Martínez-Miranda, and Bella Martínez Seis, editors, *Advances in Computational Intelligence*, pages 383–396, Cham, 2022. Springer Nature Switzerland.

MPC04. Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of boolean functions. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 474–491, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

MPJ+22. Luca Mariot, Stjepan Picek, Domagoj Jakobovic, Marko Djurasevic, and Alberto Leporati. Evolutionary construction of perfectly balanced boolean functions. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, page 1–8. IEEE Press, 2022.

MS21. Sihem Mesnager and Sihong Su. On constructions of weightwise perfectly balanced boolean functions. *Cryptography and Communications*, 2021.

MSLZ22. Sihem Mesnager, Sihong Su, Jingjing Li, and Linya Zhu. Concrete constructions of weightwise perfectly balanced (2-rotation symmetric) functions with optimal algebraic immunity and high weightwise nonlinearity. *Cryptogr. Commun.*, 14(6):1371–1389, 2022.

MW24. Pierrick Méaux and Qingju Wang. Extreme algebraic attacks. *IACR Cryptol. ePrint Arch.*, page 64, 2024.

QFLW09. Longjiang Qu, Keqin Feng, Feng Liu, and Lei Wang. Constructing symmetric boolean functions with maximum algebraic immunity. *IEEE Transactions on Information Theory*, 55:2406–2412, 05 2009.

SW12. Brett Stevens and Aaron Williams. The coolest order of binary strings. In Evangelos Kranakis, Danny Krizanc, and Flaminia L. Luccio, editors, *Fun with Algorithms - 6th International Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, volume 7288 of *Lecture Notes in Computer Science*, pages 322–333. Springer, 2012.

TD11. Ziran Tu and Yingpu Deng. A conjecture about binary strings and its applications on constructing boolean functions with optimal algebraic immunity. *DCC*, 60:1–14, 2011.

TL19. Deng Tang and Jian Liu. A family of weightwise (almost) perfectly balanced boolean functions with optimal algebraic immunity. *Cryptogr. Commun.*, 11(6):1185–1197, 2019.

YCL+23. Lili Yan, Jingyi Cui, Jian Liu, Guangquan Xu, Lidong Han, Alireza Jolfaei, and Xi Zheng. Iga: An improved genetic algorithm to construct weightwise (almost) perfectly balanced boolean functions with high weightwise nonlinearity. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '23, page 638–648, New York, NY, USA, 2023. Association for Computing Machinery.

ZCSH11. X. Zeng, C. Carlet, J. Shan, and L. Hu. More balanced boolean functions with optimal algebraic immunity and good nonlinearity and resistance to fast algebraic attacks. *IEEE Transactions on Information Theory*, 57(9):6310–6320, 2011.

ZJZQ23. Qinglan Zhao, Yu Jia, Dong Zheng, and Baodong Qin. A new construction of weightwise perfectly balanced functions with high weightwise nonlinearity. *Mathematics*, 11(5), 2023.

ZLC+23. Qinglan Zhao, Mengran Li, Zhixiong Chen, Baodong Qin, and Dong Zheng. A unified construction of weightwise perfectly balanced boolean functions. *Discrete Applied Mathematics*, 337:190–201, 2023.

ZS22. Linya Zhu and Sihong Su. A systematic method of constructing weightwise almost perfectly balanced boolean functions on an arbitrary number of variables. *Discrete Applied Mathematics*, 314:181–190, 2022.

ZS23. Rui Zhang and Sihong Su. A new construction of weightwise perfectly balanced boolean functions. *Adv. Math. Commun.*, 17(4):757–770, 2023.