

# Efficient Computation of Algebraic Immunity for Algebraic and Fast Algebraic Attacks

Frederik Armknecht<sup>1</sup>, Claude Carlet<sup>2</sup>, Philippe Gaborit<sup>3</sup>, Simon Künzli<sup>4</sup>, Willi Meier<sup>4</sup>, and Olivier Ruatta<sup>3</sup>

<sup>1</sup> Universität Mannheim, 68131 Mannheim (Germany),

armknecht@th.informatik.uni-mannheim.de

<sup>2</sup> INRIA, 78153 Le Chesnay Cedex (France),

claude.carlet@inria.fr

<sup>3</sup> Université de Limoges, 87060 Limoges (France),

{gaborit,olivier.ruatta}@unilim.fr

<sup>4</sup> FH Nordwestschweiz, 5210 Windisch (Switzerland),

{simon.kuenzli,willi.meier}@fhnw.ch

**Abstract.** In this paper we propose several efficient algorithms for assessing the resistance of Boolean functions against algebraic and fast algebraic attacks when implemented in LFSR-based stream ciphers. An algorithm is described which permits to compute the algebraic immunity  $d$  of a Boolean function with  $n$  variables in  $\mathcal{O}(D^2)$  operations, for  $D \approx \binom{n}{d}$ , rather than in  $\mathcal{O}(D^3)$  operations necessary in all previous algorithms. Our algorithm is based on a multivariate polynomial interpolation and is extended to compute the whole annihilator space which can be used in some cases to efficiently determine the resistance against fast algebraic attacks. For assessing the vulnerability of arbitrary Boolean functions with respect to fast algebraic attacks, an efficient generic algorithm is presented that is not based on interpolation. This algorithm is demonstrated to be particularly efficient for symmetric Boolean functions. As an application it is shown that large classes of symmetric functions are very vulnerable to fast algebraic attacks despite their proven resistance against conventional algebraic attacks.

**Key words.** Algebraic Attacks, Algebraic Degree, Boolean Functions, Fast Algebraic Attacks, Stream Ciphers, Symmetric Functions.

## 1 Introduction

Many keystream generators consist of combining several linear feedback shift registers (LFSRs) and possibly some additional memory. One example is the  $E_0$  keystream generator which is part of the Bluetooth standard. LFSRs are very efficient in hardware and can be designed such that the produced bitstream has maximum period and good statistical properties. Various approaches to the cryptanalysis of LFSR-based stream ciphers were discussed in literature (e.g., time-memory-tradeoff, fast correlation attacks or BDD-based attacks). For some

keystream generators, algebraic attacks and fast algebraic attacks outmatched all previously known attacks [3, 12, 13].

For LFSR-based filter or combining generators their security mainly relies on a nonlinear Boolean output function  $f$  filtering the contents of one LFSR or combining the outputs of several ones. The present paper studies the resistance of this kind of stream ciphers to (fast) algebraic attacks.

In view of algebraic attacks, the notion of algebraic immunity (or annihilator immunity) has been introduced (the algebraic immunity  $\mathcal{AI}$  of a Boolean function  $f$  is the minimum value of  $d$  such that  $f$  or  $f + 1$  admits a function  $g$  of degree  $d$  such that  $fg = 0$ ). The construction of Boolean functions for LFSR-based stream ciphers with large algebraic immunity achieved much attention recently, [5–7, 15, 16]. However, many of these functions do not allow for other good cryptographic properties like large non-linearity or large orders of resiliency, and as will be shown later, have undesirable properties with regard to fast algebraic attacks. It seems therefore relevant to be able to efficiently determine the immunity of existing and newly constructed Boolean functions against algebraic and fast algebraic attacks.

Until now, the best algorithms known for computing the algebraic immunity  $d$  of a function with  $n$  variables work roughly in  $\mathcal{O}(D^3)$  operations, where  $D \approx \binom{n}{d}$ . This is impractical for many situations in the design of stream ciphers where the number of variables is 20 or larger. In addition, two problems arise when computing the algebraic immunity of a Boolean function: firstly guess what is potentially its exact value and secondly determine efficiently whether a function has a desired algebraic immunity. In this paper we answer to these questions in one step only, by giving an algorithm which computes the algebraic immunity  $d$  of a function in  $\mathcal{O}(D^2)$  operations and this without having to guess  $d$ . The algorithm is based on the notion of multivariate polynomial interpolation. This algorithm is applied to two particular families of Boolean power functions: the inverse functions and the Kasami type functions. The quadratic nature of the algorithm is experimentally verified, and for the first time the algebraic immunity  $\mathcal{AI}$  of a function  $f$  with 20 variables is computed to be  $\mathcal{AI}(f) = 9$ . For a comparison, the computational as well as the memory complexity of the algorithms known thus far for determining the algebraic immunity for a function with this number of variables and with such a high  $\mathcal{AI}$  would be about  $2^{50}$ .

Resistance against fast algebraic attacks is not fully covered by algebraic immunity, as has been demonstrated, e.g., by a fast algebraic attack on the eS-stream candidate SFINKS, [11]. For determining immunity against fast algebraic attacks, two independent methods are given in this paper. The first one is based on computing all the annihilators of minimal degree of a polynomial  $f$ , which enables to quickly determine in some cases the resistance of a Boolean function against fast algebraic attacks. The second one is a new algorithm that is based on methods different from interpolation, and that for general Boolean functions allows to efficiently assess immunity against fast algebraic attacks. The complexity of our second algorithm is in  $\mathcal{O}(DE^2)$ , where  $E \approx \binom{n}{e}$  and  $e$  in many

cases of interest is much smaller than  $d$ . This compares favorably with the known algorithms, which are in  $\mathcal{O}(D^3)$ .

The algorithm is applied to several of the above mentioned classes of Boolean functions with optimal algebraic immunity, including symmetric Boolean functions, like the majority functions. Symmetric functions are attractive as the hardware complexity grows only linearly with the number of input variables. However, it is shown in this paper that the specific structure of these functions can be exploited in a much refined algorithm for determining resistance against algebraic attacks that is particularly efficient. It is concluded that large classes of symmetric functions are very vulnerable to fast algebraic attacks despite their optimal algebraic immunity. A symmetric function would not be implemented by itself but rather in combination with other nonlinear components in stream ciphers. It seems nevertheless essential to know the basic cryptographic properties of each component used.

The paper is organized as follows. In Section 2 the basics of algebraic and fast algebraic attacks are described. Section 3 derives an algorithm for efficient computation of the algebraic immunity as well as a modified algorithm to determine all minimal degree annihilators. In Section 4, an algorithm for efficient computation of immunity against fast algebraic attacks is presented. In Section 5 the algorithm is adapted and improved for symmetric functions, and it is proven that the class of majority functions which have maximum  $\mathcal{AI}$  is very vulnerable to fast algebraic attacks. We finally conclude in Section 6.

## 2 Algebraic Attacks and Fast Algebraic Attacks

### 2.1 Algebraic Attacks

For an LFSR  $L$  with  $N$  entries filtered by a Boolean function  $f$  with  $n$  variables, algebraic attacks consist of two steps [12]:

- **First step.** Finding functions  $g$  of low degree  $d$  such that  $fg = 0$  or  $(f + 1)g = 0$ . Until this paper the complexity of this step was roughly in  $D^3$ , for  $D := \sum_{i=0}^d \binom{n}{i}$  (which is about  $\binom{n}{d}$  for  $d < n/2$ ) and where 3 is taken for the exponent of the matrix inversion.
- **Second step.** Solving a nonlinear system of multivariate equations  $g(L^i(x_1, \dots, x_N)) = 0$  for adequate  $i$ , induced by the functions  $g$  of the annihilator set  $\text{Ann}(f)$  of  $f$ . Usually this system is solved by linearization with a complexity of  $D_N^3$  for  $D_N := \sum_{i=0}^d \binom{N}{i}$ . In this case, one needs  $D_N/\dim(\text{Ann}(f))$  bits of keystream, since experimentally independent functions  $g$  lead to independent equations. Alternatively, this system can be solved by Gröbner basis, but then the complexity of solving is difficult to evaluate, see [4, 18].

The lowest degree of the function  $g \neq 0$  for which  $fg = 0$  or  $(f + 1)g = 0$  is called the algebraic (or annihilator) immunity  $\mathcal{AI}$  of  $f$ . In [12] it has been shown that for any function  $f$  with  $n$  input variables, functions  $g \neq 0$  and  $h$  exist with  $fg = h$  such that  $e$  and  $d$  are at most  $\lceil n/2 \rceil$ . This implies that  $\mathcal{AI}(f) \leq \lceil n/2 \rceil$ .

## 2.2 Fast Algebraic Attacks

Fast algebraic attacks were introduced by Courtois in [13]. They were confirmed and improved later by Armknecht in [3] and Hawkes and Rose in [19]. A prior aim of fast algebraic attacks is to find a relation  $fg = h$  with  $e := \deg g$  small and  $d := \deg h$  larger. In classical algebraic attacks, the degree  $d$  of  $h$  would necessarily lead to considering  $D_N$  unknowns. In fast algebraic attacks, one considers that the sequence of the functions  $h(L^i(x_1, \dots, x_N))$  can be obtained as an LFSR with linear complexity  $D_N$ . One uses then the Berlekamp-Massey algorithm to eliminate all monomials of degree superior to  $e$  in the equations, such that eventually one only needs to solve a system in  $E_N := \sum_{i=0}^e \binom{N}{i}$  unknowns. The complexity of fast algebraic attacks can be summarized in these four steps:

- **Relation search step.** One searches for functions  $g$  and  $h$  of low degrees such that  $fg = h$ . For  $g$  and  $h$  of degrees  $e$  and  $d$  respectively, with associated values  $D := \sum_{i=0}^d \binom{n}{i}$  and  $E := \sum_{i=0}^e \binom{n}{i}$ , such  $g$  and  $h$  can be found when they exist by solving a linear system with  $D + E$  equations, and with complexity  $\mathcal{O}((D + E)^3)$ . Usually one considers  $e < d$ .
- **Pre-computation step.** In this step, one searches for particular linear relations which permit to eliminate monomials with degree greater than  $e$  in the equations. This step needs a sequence of  $2D_N$  bits of stream and has a complexity of  $\mathcal{O}(D_N \log^2(D_N))$ , see [19].
- **Substitution step.** At this step, one eliminates the monomials of degrees greater than  $e$ . This step has a natural complexity in  $\mathcal{O}(E_N^2 D_N)$  but using discrete Fourier transform, it is claimed in [19] that a complexity  $\mathcal{O}(E_N D_N \log(D_N))$  can be obtained.
- **Solving step.** One solves the system with  $E_N$  linear equations in  $\mathcal{O}(E_N^3)$ .

Notice that for arbitrary nonzero functions  $f, g, h$ , we have  $d \geq \mathcal{AI}(f)$ . Moreover,  $fg = h$  implies  $fh = h$ , thus we can restrict to values  $e$  with  $e \leq d$ . Fast algebraic attacks are always more efficient than conventional algebraic attacks if  $d = \mathcal{AI}(f)$  and  $e < d - 1$ . In case that  $e$  turns out to be large for this  $d$ , it is of interest to determine the minimum  $e$  where  $d$  is slightly larger than  $\mathcal{AI}(f)$ .

## 3 Efficient computation of the algebraic immunity

The best algorithms known so far for computing the algebraic immunity  $d$  of a function with  $n$  variables works roughly in  $\mathcal{O}(D^3)$  operations. Two problems arise when computing the algebraic immunity  $\mathcal{AI}(f)$  of a Boolean function  $f$ : firstly guess the value of  $\mathcal{AI}(f)$  and secondly check whether the guess is correct.

In this section these questions are answered in a single step, by giving an algorithm which computes the algebraic immunity  $d$  of a function in  $\mathcal{O}(D^2)$  operations, without necessitating a prior guess of  $d$ . The algorithm is based on the notion of multivariate polynomial interpolation.

We show that the interpolation can be translated in a linear algebra problem that can be solved in an incremental way. This insures us that the arithmetic complexity of our method depends only on the lowest degree of an annihilator. This degree is an intrinsic bound of the problem.

### 3.1 Lagrange interpolation

Before stating what is the multivariate Lagrange interpolation problem, when it is specified to binary polynomials, we need to introduce some notation. We denote by  $\mathbb{F}$  the finite field  $\text{GF}(2)$  and by  $\mathbb{F}^k$  the vector space of dimension  $k$  over  $\mathbb{F}$ . Consider  $x = x_1, \dots, x_k$  a set of  $k$  binary variables,  $\alpha = (\alpha_1, \dots, \alpha_k) \in \{0, 1\}^k$  a multi-index,  $z = (z_1, \dots, z_k)$  an element of  $\mathbb{F}^k$ . We denote  $x^\alpha = x_1^{\alpha_1} \cdots x_k^{\alpha_k}$  and  $z^\alpha = z_1^{\alpha_1} \cdots z_k^{\alpha_k}$ . Let  $E = \{\alpha_1, \dots, \alpha_D\} \subset \{0, 1\}^k$  be a set of multi-indices, then we denote by  $x^E = \{x^{\alpha_1}, \dots, x^{\alpha_D}\}$  the set of associated monomials. We recall that we identify the ring of boolean functions in  $n$  variables with  $\mathbb{F}[x]/\langle x_i^2 - x_i, i = 1, \dots, n \rangle$ , the quotient ring of the ring of polynomials with coefficients in  $\mathbb{F}$  by the ideal generated by the relations  $x_i^2 - x_i, i \in \{1, \dots, n\}$ . We will use, explicitly or not, several times this identification. In our framework, the Lagrange formula can be stated as follows:

*Problem 1.* Let  $E = \{\alpha_1, \dots, \alpha_D\} \subseteq \mathbb{F}^n$ ,  $\mathcal{Z} = \{z_1, \dots, z_D\} \subseteq \mathbb{F}^n$  and  $\bar{v} = (v_1, \dots, v_D) \in \mathbb{F}^D$ . Does there exist a polynomial  $g \in \mathbb{F}[x_1, \dots, x_n]$  whose monomial support is included in  $x^E$  and such that  $g(z_i) = v_i, \forall i \in \{1, \dots, D\}$ ?

*Remark 1.* The general multivariate Lagrange interpolation problem has been addressed in [22], but the proposed algorithm has cubic complexity (on the number of monomials). We will present an algorithm with a quadratic complexity over  $\mathbb{F}$  instead.

To answer in terms of existence and uniqueness, we need to introduce the following object (in the following we consider  $0^0 = 1$ ):

**Definition 1.** Let  $\mathcal{Z} = \{z_1, \dots, z_D\} \subseteq \mathbb{F}^n$  and  $E = \{\alpha_1, \dots, \alpha_D\} \subseteq \mathbb{F}^n$ , we define the Vandermonde matrix as follows:

$$V_{\mathcal{Z}, E} = \begin{pmatrix} z_1^{\alpha_1} & \cdots & z_1^{\alpha_D} \\ \vdots & \ddots & \vdots \\ z_D^{\alpha_1} & \cdots & z_D^{\alpha_D} \end{pmatrix} \quad (1)$$

and we define the Vandermonde determinant to be  $v_{\mathcal{Z}, E} = \det(V_{\mathcal{Z}, E})$ .

The following proposition answers to the existence and uniqueness of solution of problem 1:

**Proposition 1.** Let  $\mathcal{Z} = \{z_1, \dots, z_D\} \subseteq \mathbb{F}^n$  and  $E = \{\alpha_1, \dots, \alpha_D\} \subseteq \mathbb{F}^n$ . There exists a solution  $g \in \mathbb{F}[x]$  to Problem 1 if and only if  $v_{\mathcal{Z}, E} \neq 0$ . Furthermore, the solution  $g(x)$  is given by:

$$g(x) = (g_{\alpha_1}, \dots, g_{\alpha_D}) \begin{pmatrix} x^{\alpha_1} \\ \vdots \\ x^{\alpha_D} \end{pmatrix}$$

where the vector  $g = (g_{\alpha_1}, \dots, g_{\alpha_D})^t$  is the only solution of the following system:

$$V_{\mathcal{Z}, E} g = v. \quad (2)$$

*Remark 2.* Given  $\mathcal{Z} = \{z_1, \dots, z_D\} \subseteq \mathbb{F}^n$ , the existence of a set  $E = \{\alpha_1, \dots, \alpha_D\} \subseteq \mathbb{F}^n$  such that  $v_{\mathcal{Z}, E} \neq 0$  is insured since it is enough to take for  $E$  the set of monomial whose are not in the monomial ideal generated by the leading monomials of a Gröbner basis of the ideal of the polynomials vanishing at each point of  $\mathcal{Z}$  (this ideal contains the ideal generated by the  $x_i^2 - x_i$ ). Of course, several set  $E$  are possible since two Gröbner bases for different monomial order may lead to different such sets.

The following proposition show that the minimal annihilator problem can be reduced to a Lagrange interpolation problem :

**Proposition 2.** *Let  $f$  be a boolean function, let  $\mathcal{Z} = f^{-1}(1)$ . Let  $E$  be such that  $x^E$  is the complementary of the monomial ideal generated by the leading monomials of Gröbner basis for a graduated order of the ideal of the polynomials vanishing at each point of  $\mathcal{Z}$ , then, if  $\beta \notin E$  is of lowest weight, the function  $R_\beta$  define below is a minimal algebraic degree annihilator of  $f$  :*

$$R_\beta = \det \begin{pmatrix} x^\beta & x^{\alpha_1} & \dots & x^{\alpha_D} \\ z_1^\beta & z_1^{\alpha_1} & \dots & z_1^{\alpha_D} \\ \vdots & \vdots & \ddots & \vdots \\ z_D^\beta & z_D^{\alpha_1} & \dots & z_D^{\alpha_D} \end{pmatrix}$$

Furthermore,  $R_\beta = x^\beta - g$  where  $g$  is the solution of the Problem 1 taking  $v = (z_1^\beta, z_2^\beta, \dots, z_D^\beta)$ .

The idea is then to solve the linear system (2) to compute the solution of Problem 1 and so to the minimal degree annihilator problem. In the next subsection, we describe an algorithm solving the Problem 1, generalizing the incremental univariate Newton interpolation scheme, taking  $\mathcal{O}(D^2)$  operations. The method can be easily adapted in order to compute all the annihilators of a given algebraic degree.

### 3.2 Computing a minimal degree annihilator

We need to introduce some more notation. We denote by  $E_d$  the set of all  $\alpha$  of weight equal to  $d$  and by  $E_{\leq d} := E_0 \cup \dots \cup E_d$ , ordered by the weight. Let  $\mathcal{Z} := f^{-1}(1) \subseteq \mathbb{F}^n$  and  $\mathcal{Z}_i := \{z_1, \dots, z_i\}$ . We assume that for all  $i \in \{1, \dots, t\}$ , we have  $v_{\mathcal{Z}_i, E^i} \neq 0$  (such kind of ordered sets of points and exponents always exists and can be pre-computed in quadratic time once for all, see [22], and so we do not lose any generality). Take  $V_i = V_{\mathcal{Z}_i, E^i}$  and  $\bar{v}_i := (v_1, \dots, v_i)$ . The algorithm proposed here relies on a multivariate generalization of the Newton interpolation scheme. The computation of both the Newton polynomials and the coefficients in this basis are given in a quadratic number of arithmetic operations, depending on the number of interpolation node. This is an improvement of the classical Lagrange interpolation scheme leading to a cubic number of operations, essentially because one must compute the Lagrange interpolation polynomials. The Newton

interpolation scheme has an important property : interpolation nodes are introduced one by one in order to compute the interpolation polynomial. Classically the coefficient in the Newton basis are computed using divided differences. The approach proposed here is based on an interpretation of the Newton interpolation scheme in term of linear algebra (see [23]). Roughly speaking, the only property we really need is that the Newton basis is a basis where the matrix associated to the Vandermonde is triangular. The algorithm relies on the following remark:

*Remark 3.* Make an  $LU$ -decomposition of  $V_i$ , i.e. compute  $V_i = L_i U_i$ , where  $U_i$  is triangular superior and  $L_i$  is triangular inferior. Then a solution of  $V_i \bar{g}_i = \bar{v}_i$  is a solution of  $U_i \bar{g}_i = L_i^{-1} \bar{v}_i$ . In this way, we have two triangular systems to solve (computing the inverses of  $U_i$  and  $L_i$ ). Imagine now that the computed solution  $\bar{g}_i$  does not give an annihilator. Then, instead of computing a complete  $LU$ -decomposition of  $V_{i+1}$ , we write

$$V_{i+1} = \begin{pmatrix} V_i & C_{i+1} \\ R_i & z_{i+1}^{\alpha_{i+1}} \end{pmatrix} \quad \text{with} \quad C_{i+1} = \begin{pmatrix} z_1^{\alpha_{i+1}} \\ \vdots \\ z_i^{\alpha_{i+1}} \end{pmatrix}, \quad R_i = (z_{i+1}^{\alpha_1}, \dots, z_{i+1}^{\alpha_i})$$

and set  $v_{i+1} = \begin{pmatrix} \bar{v}_i \\ v_{i+1} \end{pmatrix}$ . We have  $V_{i+1} = \begin{pmatrix} L_i & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} U_i & L_i^{-1} C_{i+1} \\ R_i & z_{i+1}^{\alpha_{i+1}} \end{pmatrix}$  in a way that we do not have to do a complete  $LU$ -decomposition of  $V_{i+1}$  since we know an almost  $LU$ -decomposition from the decomposition of  $V_i$ . This is a basic fact usually exploited to design efficient  $LU$ -factorization algorithms.

We do not want to compute the factors  $L_i$  of the above remark. This is done using the following updating process, in order to find  $g$ . Denote  $T = (t_1, \dots, t_i)^t$  where the  $t_i$  are considered as indeterminate,  $P_i(t_1, \dots, t_i) := L_i^{-1} T$ , we have :

$$V_{i+1} = \begin{pmatrix} L_i & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} U_i & P_i(z_1^{\alpha_{i+1}}, \dots, z_i^{\alpha_{i+1}}) \\ z_{i+1}^{\alpha_1} \dots z_{i+1}^{\alpha_i} & z_{i+1}^{\alpha_{i+1}} \end{pmatrix}. \quad (3)$$

and hence  $\begin{pmatrix} v_1 \\ \vdots \\ v_{i+1} \end{pmatrix} = \begin{pmatrix} L_i & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} P_i(v_1, \dots, v_i) \\ v_{i+1} \end{pmatrix}$ . In order to solve  $V_{i+1} \bar{g}_{i+1} = \bar{v}_{i+1}$ , we only have to solve the following equation:

$$\begin{pmatrix} U_i & P_i(z_1^{\alpha_{i+1}}, \dots, z_i^{\alpha_{i+1}}) \\ z_{i+1}^{\alpha_1} \dots z_{i+1}^{\alpha_i} & z_{i+1}^{\alpha_{i+1}} \end{pmatrix} \bar{g}_{i+1} = \begin{pmatrix} P_i(v_1, \dots, v_i) \\ v_{i+1} \end{pmatrix}. \quad (4)$$

Then we triangulate the left matrix (i.e. we eliminate the  $i^{\text{th}}$  first terms of the last row) by row operations. This is an easy task since  $U_i$  is triangular, doing the same operations on  $\begin{pmatrix} P_i(v_1, \dots, v_i) \\ v_{i+1} \end{pmatrix}$ . This gives us both  $U_{i+1}$  and  $P_{i+1}(t_1, \dots, t_{i+1})$ . Then we solve the system  $U_{i+1} \bar{g}_{i+1} = P_{i+1}(v_1, \dots, v_{i+1})$  and

if the polynomial associated to  $\bar{g}_{i+1}$  is not an annihilator, we add the next monomial in  $E_{\leq d}$  and so on. The method terminates because the degree of the annihilator is bounded. To test that the computed polynomial is an annihilator, we only have to check that the computed polynomial vanishes at points where the input polynomial does not vanish. The vectors  $P_i$  is not formally computed but its evaluation is computed using an updating even if we use the vector in notations for simplicity. As an input of the algorithm, we do not take a monomial expansion of  $f$  but the vector of its evaluation at the points  $z_i$ . If  $t := |\mathcal{Z}|$ , then this can be computed with asymptotically  $\mathcal{O}(t \log(t))$  operations using a method based on fast Fourier transform and more easily in  $\mathcal{O}(tN_m)$  operations over the ground field for  $N_m$  the number of monomials in the algebraic normal form of  $f$  simply by adding the evaluation of each monomial at the  $t$  points. Let us now introduce the algorithm:

---

**Algorithm 1** Computation of an annihilator of minimal degree

---

**Input:**  $f, \mathcal{Z} := f^{-1}(1), E_{\leq \lceil n/2 \rceil}$ .

**Output:** An annihilator of  $f$  of minimal degree.

- 1: Initialization:  $V := (z_1^{\alpha_1}), g := 1, v_1 := f(z_1) \oplus 1 = 0, P := (t_1), i := 1$ .
  - 2: **while** the polynomial associated to  $g$  is not an annihilator of  $f$  **do**
  - 3:  $i \leftarrow i + 1$ .
  - 4:  $V \leftarrow \begin{pmatrix} V & C \\ R & z_i^{\alpha_i} \end{pmatrix} := \begin{pmatrix} V & P(z_1^{\alpha_1}, \dots, z_{i-1}^{\alpha_{i-1}}) \\ z_i^{\alpha_1} \dots z_i^{\alpha_{i-1}} & z_i^{\alpha_i} \end{pmatrix}$ .
  - 5:  $P(t_1, \dots, t_i) \leftarrow \begin{pmatrix} P(t_1, \dots, t_{i-1}) \\ t_i \end{pmatrix}$  (in fact, update  $P(z_i^{\alpha_1} \dots z_i^{\alpha_i})$  and  $P(v_1, \dots, v_i)$ ).
  - 6: Triangulate  $V$  and update  $P$  (see below).
  - 7: Solve  $V\bar{g} = P(v_1, \dots, v_i)$  (see below).
  - 8: **end while**
  - 9: Output  $g := \bar{g}^t(x^{\alpha_0}, \dots, x^{\alpha_i})$ .
- 

At the  $i^{\text{th}}$  step of the algorithm, the most costly operations are:

- The triangulation of the new extended matrix  $V$ : As the previous matrix  $V$  is already a triangular matrix, we need only to eliminate the entries in  $R$  and to update the entry in the bottom right corner. This is done by substituting  $z_i^{\alpha_1}, \dots, z_i^{\alpha_{i-1}}$  by zero and  $z_i^{\alpha_i}$  by  $z_i^{\alpha_i} - \sum_{j=1}^{i-1} z_i^{\alpha_1} \cdot P_{i,j}$  where  $(P_{i,1}, \dots, P_{i,i-1}) = P(z_1^{\alpha_1}, \dots, z_{i-1}^{\alpha_{i-1}})$ . This step requires  $i$  arithmetic operations.
- The updating of  $P$  is done with the same number of arithmetic operations. Recalling that we do not compute  $P$  itself but the two vectors representing  $P$  evaluated at  $(z_i^{\alpha_1} \dots z_i^{\alpha_i})$  and  $(v_1, \dots, v_i)$ .
- The solving of the system is basically done in  $i^2$  arithmetic operations but this is also amenable in  $i$  arithmetic operations by remarking that:

$$V_{i+1}\bar{g}_{i+1} = \begin{pmatrix} V_i & P_i(z_1^{\alpha_{i+1}}, \dots, z_i^{\alpha_{i+1}}) \\ 0 & * \end{pmatrix} \begin{pmatrix} \bar{g}_i \\ v_{i+1} \end{pmatrix} = \begin{pmatrix} P_i(v_1, \dots, v_i) \\ \bullet \end{pmatrix},$$

allowing to correct  $\bar{g}_i$  in order to compute  $\bar{g}_{i+1}$ .

Again, let us recall that the stopping test consists in checking that  $g$  vanishes where  $f$  does not. This does not introduce any new complex computations. The point is to compute the values of  $g$  at the points that are not already introduced during the algorithm. This can be also be done by an updating approach by updating a vector storing the evaluations of  $g$  at each points considered so far, where each new step leads to as many additions as coordinates of this vector, i.e. to a linear number of operations. Again, the total coast of this computation is quadratic on the number of points.

To summarize, the arithmetic complexity  $T(N)$  of the proposed algorithm is given by the following formula:  $T(N) = T(N - 1) + a * i + \mathcal{O}(D)$ . An easy computation shows that  $T(N) = \mathcal{O}(t^2 + tD)$ . This is summarized in the following proposition:

**Proposition 3.** *The arithmetic complexity of the above algorithm to compute the degree of the annihilator of a polynomial of degree  $d$  in  $\mathbb{F}[x]$  is  $\mathcal{O}(t^2 + D)$  arithmetic operations, where  $t$  is the number of monomials occurring in a minimal degree annihilator of  $f$ .*

Since  $t$  has the same order of magnitude as  $D$ , we have the following corollary:

**Corollary 1.** *The arithmetic complexity of the above algorithm to compute the degree of the annihilator of a polynomial of degree  $d$  in  $\mathbb{F}[x]$  is  $\mathcal{O}(D^2)$  arithmetic operations.*

One can remark that this method can be used in order to construct functions with high algebraic immunity. Notice also that in order to obtain the quadratic behavior, it is necessary to handle memory allocation with care since the management of the “extension” operations on the matrix are difficult and that a bad memory allocation leads to an implementation cubic in space (and so unfortunately in time also because we need to parse the matrices for copies).

### 3.3 Computing all the minimal degree annihilators

In this section, we explain how to modify the algorithm of the previous section in order to compute all the annihilators of minimal degree for a polynomial  $f$ . First remark that the space  $\text{Ann}(f, d) = \{g \in \langle E_{\leq d} \rangle \subseteq \mathbb{F}[x] \mid f \cdot g = 0\}$  is a vector space and hence that no linearly dependent two annihilators of the same algebraic degree exist. In order to compute a basis of  $\text{Ann}(f, d)$ , we start the algorithm 1 by we change the halting state in the ”while” loop. When the least annihilator had been found we set a new local variable (say `degreeReach`) to ”true”, store the computed annihilator and remove the last monomial introduced and introduce the next one in the list of the monomials of the same degree, if this lead to a new annihilator store it and remove the monomial from the list, go to the next monomial of the same degree and so one until we have introduced all the monomials of this degree. Then stop. Since the number of monomial to test

at this degree is of the same order of magnitude than the number of monomials introduce before to find the first annihilator, we do not change the asymptotic performance of the algorithm. All this is resumed in the following proposition :

**Proposition 4.** *The above modifications of algorithm 1 allows to compute a basis of  $\text{Ann}(f, d)$ , where  $d$  is the minimal degree of an annihilator of  $f$ , using  $\mathcal{O}(D^2)$  arithmetic operations.*

### 3.4 Examples and computations of the algebraic immunity of some functions

In this section we apply our algorithm to two particular families of Boolean power functions (see [10]): the inverse functions and the Kasami type functions. We checked that the implementation in  $\mathbb{C}$  of the algorithm followed the announced quadratic complexity as the number of variables increased.

The inverse function is of particular interest since this function is used in the case of 8 variables in the S-box of the AES and almost directly in the new cipher SFINKS [6] as a filtering function. Table 1 and Table 2 give respectively the power exponent  $d$  of the function  $f$ , its weight, its algebraic degree, its nonlinearity and its algebraic immunity.

The Kasami functions with  $n$  variables have exponents of the form  $2^{2k} - 2^k + 1$  with  $\text{gcd}(k, n) = 1$  and  $k \leq n/2$ . These functions are of interest since they usually have a high algebraic immunity. In Table 2 we considered Kasami type exponents where  $\text{gcd}(k, n)$  may be different of 1 and for  $n = 12, 16, 20$  (a '\*' in the table) we started from non-balanced functions that we made balanced by turning the first '0' in '1' until we obtained a balanced function. We obtain for the first time the computation of a function with 20 variables and  $\mathcal{AI}$  9.

**Table 1.** Computation of the nonlinearity and algebraic immunity for the inverse function for  $12 \leq n \leq 20$ .

$n$	$d$	weight	degree	nonlinearity	$\mathcal{AI}$
12	-1	2048	11	1984	5
13	-1	4096	12	4006	6
14	-1	8192	13	8064	6
15	-1	$2^{14}$	14	16204	6
16	-1	$2^{15}$	15	$2^{15} - 2^8$	6
17	-1	$2^{16}$	16	65174	7
18	-1	$2^{17}$	17	$2^{17} - 2^9$	7
19	-1	$2^{18}$	18	261420	7
20	-1	$2^{19}$	19	$2^{19} - 2^{10}$	7

**Table 2.** Computation of the nonlinearity and algebraic immunity for some Kasami type power functions for  $12 \leq n \leq 20$ .

$n$	$d$	weight	degree	nonlinearity	$\mathcal{AI}$
12	993	2048*	11	1984	5
13	993	4096	6	$2^{12} - 2^6$	6
14	4033	8192	6	$2^{13} - 2^7$	6
15	4033	$2^{14}$	7	$2^{14} - 2^8$	7
16	$2^{14} - 2^7 + 1$	$(2^{15})^*$	15	$2^{15} - 2^7$	7
17	$2^{14} - 2^7 + 1$	$2^{16}$	8	$2^{16} - 2^8$	8
18	$2^{16} - 2^8 + 1$	$2^{17}$	9	$2^{17} - 2^9$	8
19	$2^{16} - 2^8 + 1$	$2^{18}$	9	$2^{18} - 2^9$	9
20	$2^{18} - 2^9 + 1$	$(2^{19})^*$	19	$2^{19} - 2^9$	9

### 3.5 Application to other types of algebraic attacks

In the following, we argue why Alg. 1 and 2 are helpful in other scenarios. Recall that Alg. 1 is an efficient method to find a Boolean function  $g$  which is zero on a given set  $\mathcal{Z}$  and has the lowest possible degree. As it has been shown in [2], the equations usable in the case of block ciphers or combiners with memory are all given by functions which are zero on a given set. Hence, Alg. 1 can be directly applied to check the existence of low-degree equations in these settings.

Furthermore, the algorithm can be used in the context of fast algebraic attacks. An important step for fast algebraic attacks is to find  $g$  and  $h$  of degrees  $e$  and  $d$ , respectively, such that  $fg = h$ . A simple algorithm to know whether there exist such  $g$  and  $h$  is to check if non-trivial polynomials exist in the intersection of the vector spaces  $\langle x^\alpha \cdot f : |\alpha| \leq e \rangle$  and  $\langle x^\alpha : |\alpha| \leq d \rangle$ . This can be done by solving a system with  $D + E$  unknowns, hence with a complexity of  $\mathcal{O}((D + E)^3)$ .

An obvious observation is also that, if  $fg = h$  and  $h \neq 0$ , then we have  $ffg = fh = h$  and  $h$  is an annihilator of  $f \oplus 1$ . Hence, it suffices to check whether the intersection of  $\langle x^\alpha \cdot f : |\alpha| \leq e \rangle$  and  $\text{Ann}(f \oplus 1, d)$  contains only the all-zero polynomial. Hereby, Alg. 2 can be helpful which permits to compute a basis of  $\text{Ann}(f \oplus 1, d)$  in  $\mathcal{O}(D^2)$ .

Note that, very often, the dimension of the annihilator space is much lower than  $D$ . In fact, experimentally, it is frequent that the annihilator space has a very small dimension  $\text{Dim}(\text{Ann}(f \oplus 1, d))$ ;  $d = \mathcal{AI}(f)$ . In that case, the system determining whether  $g$  and  $h$  exist with  $E + D$  unknowns can be replaced by a system with only  $E + \text{dim}(\text{Ann}(f \oplus 1, d))$  unknowns. This is particularly interesting for small  $e$  (which is the interesting case) and small value of  $\text{dim}(\text{Ann}(f \oplus 1, d))$ .

In the next section, we will present an alternative algorithm with a complexity of  $\mathcal{O}(D \cdot E^2)$  and optimize it for the case of symmetric functions  $f$ .

## 4 Efficient Computation of Immunity against Fast Algebraic Attacks

Let us first introduce some notation for this section. Any Boolean function  $f$  with  $n$  input variables  $\mathbf{x} := x_1, \dots, x_n$  can be characterized by its truth table  $\mathbf{T}(f) := (f(0), \dots, f(2^n - 1)) \in \mathbb{F}^{2^n}$  or by its algebraic normal form  $f(\mathbf{x}) = \bigoplus_{\alpha} f_{\alpha} \mathbf{x}^{\alpha}$ , with coefficients  $f_{\alpha} \in \mathbb{F}$ , multi-indices  $\alpha \in \mathbb{F}^n$  (which can also be identified by their integers  $\alpha$ ) and the abbreviation  $\mathbf{x}^{\alpha} := x_1^{\alpha_1} \cdot \dots \cdot x_n^{\alpha_n}$ . Consequently, we define the coefficient vector of  $f$  by  $\mathbf{C}(f) := (f_0, \dots, f_{2^n-1}) \in \mathbb{F}^{2^n}$ .

Given a Boolean function  $f$  with  $n$  input variables, the goal is to decide whether  $g$  of degree  $e$  and  $h$  of degree  $d$  exist, such that  $fg = h$ . The known function  $f$  is represented by the truth table  $\mathbf{T}(f)$ , which allows efficient access to the required elements, and the unknown functions  $g$  and  $h$  are represented by the coefficient vectors  $\mathbf{C}(g)$  and  $\mathbf{C}(h)$ , which leads to the simple side conditions  $g_{\beta} = 0$  for  $|\beta| > e$  and  $h_{\gamma} = 0$  for  $|\gamma| > d$ . In order to decide if  $g$  and  $h$  exist, one has to set up a number of linear equations in  $g_{\beta}$  and  $h_{\gamma}$ . Such equations are obtained, e.g., by evaluation of  $f(\mathbf{z}) \cdot \bigoplus_{\beta} g_{\beta} \mathbf{z}^{\beta} = \bigoplus_{\gamma} h_{\gamma} \mathbf{z}^{\gamma}$  for some values of  $\mathbf{z}$ . There are  $D + E$  variables, so one requires at least the same number of equations. The resulting system of equations can be solved by Gaussian elimination with time complexity  $\mathcal{O}((D + E)^3) = \mathcal{O}(D^3)$ . If any  $D + E$  equations are linearly independent, then no nontrivial  $g$  and  $h$  of corresponding degree exist. Otherwise, one may try to verify a nontrivial solution. Certainly, there are more sophisticated algorithms, namely we are able to express a single coefficient  $h_{\gamma}$  as a linear combination of coefficients  $g_{\beta}$ . If these relations hold for any value of  $\gamma$ , one may choose  $\gamma$  with  $|\gamma| > d$  such that  $h_{\gamma} = 0$ , in order to obtain relations in  $g_{\beta}$  only. Consequently, equations for coefficients of  $g$  can be completely separated from equations for coefficients of  $h$ . As there are only  $E$  variables  $g_{\beta}$ , one requires at least  $E$  equations, and the system of equations can be solved in  $\mathcal{O}(E^3)$ . Depending on the parameters  $n, d, e$  and on the structure of  $f$ , there are different strategies how to efficiently set up equations.

### 4.1 Setting up Equations

In this section, we consider the product  $fg = h$  where  $f, g$  and  $h$  are arbitrary Boolean functions in  $n$  variables. Here are some additional notational conventions: For  $\alpha, \beta, \gamma \in \mathbb{F}^n$ , let  $\alpha \subseteq \beta$  be an abbreviation for  $\text{supp}(\alpha) \subseteq \text{supp}(\beta)$ , and let  $\alpha \vee \beta := \alpha_1 \vee \beta_1, \dots, \alpha_n \vee \beta_n$ . For  $\mathbf{B}, \mathbf{C} \in \mathbb{F}^{2^n}$ , we define the scalar product  $\mathbf{B} \cdot \mathbf{C} := \bigoplus_{k=0}^{2^n-1} [\mathbf{B}]_k \cdot [\mathbf{C}]_k$ . All expressions are modulo 2 here. With the following theorem, we are able to express a single coefficient  $h_{\gamma}$  as a linear combination of coefficients  $g_{\beta}$ , where the linear combination is computed either with  $\mathbf{T}(f)$  or with  $\mathbf{C}(f)$ .

**Theorem 1.** *Let  $f(\mathbf{x}) = \bigoplus_{\alpha} f_{\alpha} \mathbf{x}^{\alpha}$  and  $g(\mathbf{x}) = \bigoplus_{\beta} g_{\beta} \mathbf{x}^{\beta}$ . Set  $h(\mathbf{x}) = \bigoplus_{\gamma} h_{\gamma} \mathbf{x}^{\gamma} := f(\mathbf{x}) \cdot g(\mathbf{x})$ . Then, with  $A_{i,j} \in \mathbb{F}$  and  $\mathbf{B}_{i,j} \in \mathbb{F}^{2^n}$ , we have for each  $\gamma$*

$$h_{\gamma} = \bigoplus_{\beta} \binom{\gamma}{\beta} A_{\gamma,\beta} \cdot g_{\beta} \quad (5)$$

$$A_{i,j} := \mathbf{B}_{i,j} \cdot \mathbf{T}(f) = \mathbf{B}_{i,i-j} \cdot \mathbf{C}(f) \quad (6)$$

$$[\mathbf{B}_{i,j}]_k := \binom{i}{k} \cdot \binom{k}{j}. \quad (7)$$

*Proof.* The binary Moebius transform relates the ANF of a Boolean function with the corresponding truth table, namely  $f(\mathbf{k}) = \bigoplus_{\alpha} \binom{\mathbf{k}}{\alpha} f_{\alpha} = \bigoplus_{\alpha \subseteq \mathbf{k}} f_{\alpha}$  and  $f_{\mathbf{k}} = \bigoplus_{\alpha} \binom{\mathbf{k}}{\alpha} f(\alpha) = \bigoplus_{\alpha \subseteq \mathbf{k}} f(\alpha)$ . We obtain the relation  $h_{\gamma} = \bigoplus_{\alpha \subseteq \gamma} f(\alpha) g_{\alpha}$ . With  $g(\alpha) = \bigoplus_{\beta \subseteq \alpha} g_{\beta}$ , this becomes  $h_{\gamma} = \bigoplus_{\alpha \subseteq \gamma} \bigoplus_{\beta \subseteq \alpha} g_{\beta} f(\alpha)$ . Rearranging the coefficients and considering Lucas' theorem, we finally have the product  $h_{\gamma} = \bigoplus_{\beta \subseteq \gamma} g_{\beta} \bigoplus_{\alpha \subseteq \beta} f(\alpha) = \bigoplus_{\beta} \binom{\gamma}{\beta} g_{\beta} \mathbf{B}_{\gamma,\beta} \cdot \mathbf{T}(f)$ . In order to prove the second relation, we multiply the ANF of both functions and obtain  $h_{\gamma} = \bigoplus_{\alpha \vee \beta = \gamma} f_{\alpha} g_{\beta}$ . The sum can be partitioned according to  $h_{\gamma} = \bigoplus_{\beta \subseteq \gamma} g_{\beta} \bigoplus_{\alpha \subseteq \gamma, \alpha \vee \beta = \gamma} f_{\alpha}$ . With Lucas' theorem again, we have the relation  $h_{\gamma} = \bigoplus_{\beta \subseteq \gamma} g_{\beta} \bigoplus_{\gamma - \beta \subseteq \alpha \subseteq \gamma} f_{\alpha} = \bigoplus_{\beta} \binom{\gamma}{\beta} g_{\beta} \mathbf{B}_{\gamma, \gamma - \beta} \cdot \mathbf{C}(f)$ .  $\square$

## 4.2 Determining the Existence of Solutions

We propose an efficient algorithm to determine the existence of  $g$  and  $h$  with corresponding degrees. The algorithm is based on the equation  $h_{\gamma} = \bigoplus_{\beta \subseteq \gamma} g_{\beta} \bigoplus_{\beta \subseteq \alpha \subseteq \gamma} f(\alpha)$ , which is a variant of Th. 1.

---

**Algorithm 2** Determine the existence of  $g$  and  $h$  for any  $f$

---

**Input:** A Boolean function  $f$  with  $n$  input variables, functions  $g$  and  $h$  with  $\deg g = e$  and  $\deg h = d$ .

**Output:** Determine if  $g$  and  $h$  exist such that  $fg = h$ .

- 1: Initialize an  $E \times E$  matrix  $G$ , and let each entry be zero.
  - 2: Compute an ordered set  $\mathcal{I} \leftarrow \{\beta : |\beta| \leq e\}$ .
  - 3: **for**  $i$  from 1 to  $E$  **do**
  - 4:   Choose a random  $\gamma$  with  $|\gamma| = d + 1$ .
  - 5:   Determine the set  $\mathcal{B} \leftarrow \{\beta : \beta \subseteq \gamma, |\beta| \leq e\}$ .
  - 6:   **for all**  $\beta$  in  $\mathcal{B}$  **do**
  - 7:     Determine the set  $\mathcal{A} \leftarrow \{\alpha : \beta \subseteq \alpha \subseteq \gamma\}$ .
  - 8:     Compute  $A \leftarrow \bigoplus_{\alpha \in \mathcal{A}} f(\alpha)$ .
  - 9:     Let the entry of  $G$  in row  $i$  and column  $\beta$  (in respect to  $\mathcal{I}$ ) be 1 if  $A = 1$ .
  - 10:   **end for**
  - 11: **end for**
  - 12: Solve the linear system of equations, and output **no**  $g$  and  $h$  of corresponding degree if there is only a trivial solution.
- 

Let us discuss the complexity of Alg. 3. Initialization of  $G$  takes at most  $\mathcal{O}(E^2)$  time and memory, and  $\mathcal{I}$  can be constructed in  $\mathcal{O}(E)$  time. Iteration initiates by choosing a fixed  $\gamma$  of weight  $d + 1$ , this step will be repeated  $E$  times to set up the same number of equations. Notice that the set  $\{\gamma : |\gamma| = d + 1\}$  is sufficient to choose  $E$  different values of  $\gamma$ , as  $E < \binom{n}{d+1}$  in the case of  $e \ll d$  and  $d \approx n/2$

(which is the typical scope of fast algebraic attacks). Thereafter, one chooses a fixed  $\beta$  of weight  $b$ . This step will be repeated for all  $\binom{d+1}{b}$  elements of weight  $b$ , and for all  $b = 0, \dots, e$ . Given this choice of  $\gamma$  and  $\beta$ , we find  $|\mathcal{A}| = 2^{d+1-b}$ , which corresponds to the number of operations to compute  $A$ . Overall complexity of the iteration becomes  $E \sum_{b=0}^e \binom{d+1}{b} 2^{d+1-b} < E(e+1) \binom{d+1}{e} 2^{d+1} < DE^2$ , where the last inequality holds in the specified range of parameters. Time complexity of the final step of Alg. 3 is  $\mathcal{O}(E^3)$ . The dominating term, and hence complexity of Alg. 3 corresponds to  $\mathcal{O}(DE^2)$ . Compared to the complexity  $\mathcal{O}(D^3)$  of Alg. 2 in [20], Alg. 3 is very efficient for  $g$  of low degree.

### 4.3 Experimental Results

In [15], a class of (non-symmetric) Boolean functions  $f$  with maximum algebraic immunity is presented; these functions will be referred here as DGM functions. Application of Alg. 3 on their examples for  $n = 5, 6, 7, 8, 9, 10$  reveals that  $h$  and  $g$  exist with  $d = \mathcal{AI}(f) = \lceil n/2 \rceil$  and  $e = 1$ . We point out that this is the most efficient situation for a fast algebraic attack. Explicit functions  $g$  with corresponding degree are also obtained by Alg. 3, see Tab. 3 (where  $\dim$  denotes the dimension of the solution space for  $g$  of degree  $e$ ). A formal expansion of  $f(\mathbf{x}) \cdot g(\mathbf{x})$  was performed to verify the results.

**Table 3.** Degrees of the functions  $h$  and  $g$  for DGM functions  $f$  with  $n$  input variables.

$n$	$\deg f$	$\deg h$	$\deg g$	$g$	$\dim$
5	4	3	1	$1 + x_4$	4
6	4	3	1	$1 + x_6$	4
7	5	4	1	$1 + x_4 + x_5$	1
8	5	4	1	$1 + x_5 + x_6$	1
9	8	5	1	$x_4 + x_5 + x_6 + x_7$	1
10	8	6	1	$x_5 + x_6 + x_7 + x_8$	1

## 5 Efficient Computation of Immunity for Symmetric Functions

Consider the case that  $f(\mathbf{x})$  is a symmetric Boolean function. This means that  $f(\mathbf{x}) = f(x_1, \dots, x_n)$  is invariant under changing the variables  $x_i$ . Therefore, it is  $f(\mathbf{y}) = f(\mathbf{y}')$  if  $|\mathbf{y}| = |\mathbf{y}'|$  and we can identify  $f$  with its (abbreviated) truth table  $\mathbf{T}^s(f) := (f^s(0), \dots, f^s(n)) \in \mathbb{F}^{n+1}$  where  $f^s(i) := f(\mathbf{y})$  for a  $\mathbf{y}$  with  $|\mathbf{y}| = i$ . Let  $\sigma_i(\mathbf{x}) := \bigoplus_{|\alpha|=i} \mathbf{x}^\alpha$  denote the elementary symmetric polynomial of degree  $i$ . Then, each symmetric function  $f$  can be expressed by  $f(\mathbf{x}) = \bigoplus f_i^s \sigma_i(\mathbf{x})$  with  $f_i^s \in \mathbb{F}$ . Similarly to the non-symmetric case,  $f$  can be identified with its coefficient vector  $\mathbf{C}^s(f) := (f_0^s, \dots, f_n^s) \in \mathbb{F}^{n+1}$ .

In this section, we present a general analysis of the resulting system of equations for symmetric functions and propose a generic and a specific algorithm in order to determine the existence of  $g$  and  $h$  of low degree.

### 5.1 Setting up Equations

One can derive a much simpler relation for the coefficients  $h_\gamma$  in the case of symmetric functions  $f$ .

**Corollary 2.** *Let  $f(\mathbf{x}) = \bigoplus_{i=0}^n f_i^s \sigma_i(\mathbf{x})$  a symmetric function and  $g(\mathbf{x}) = \bigoplus_{\beta} g_\beta \mathbf{x}^\beta$ . Set  $h(\mathbf{x}) = \bigoplus_{\gamma} h_\gamma \mathbf{x}^\gamma := f(\mathbf{x}) \cdot g(\mathbf{x})$ . Then, with  $A_{i,j}^s \in \mathbb{F}$  and  $\mathbf{B}_{i,j}^s \in \mathbb{F}^{n+1}$ , we have for each  $\gamma$*

$$h_\gamma = \bigoplus_{\beta} \binom{\gamma}{\beta} A_{|\gamma|,|\beta|}^s \cdot g_\beta \quad (8)$$

$$A_{i,j}^s := \mathbf{B}_{i,j}^s \cdot \mathbf{T}^s(f) = \mathbf{B}_{i,i-j}^s \cdot \mathbf{C}^s(f) \quad (9)$$

$$[\mathbf{B}_{i,j}^s]_k := \binom{i-j}{i-k}. \quad (10)$$

*Proof.* Notice that Th. 1 holds for any function  $f$ , including symmetric functions. Computation of  $A_{\gamma,\beta} = \mathbf{B}_{\gamma,\beta} \cdot \mathbf{T}(f)$  for symmetric functions may be simplified by collecting all terms of the truth table with the same weight. Therefore, let  $i := |\gamma|$  and  $j := |\beta|$  and define  $[\mathbf{B}_{i,j}^s]_k := \bigoplus_{|\alpha|=k} [\mathbf{B}_{\gamma,\beta}]_\alpha$ , such that  $A_{\gamma,\beta} = A_{i,j}^s := \mathbf{B}_{i,j}^s \cdot \mathbf{T}^s(f)$ . For  $j \leq i$  we have  $\bigoplus_{|\alpha|=k} \binom{\gamma}{\alpha} \binom{\alpha}{\beta} = \bigoplus_{|\alpha|=k; \beta \subseteq \alpha \subseteq \gamma} 1$ . Counting the number of choices of the  $k$  elements of the support of  $\alpha$ , we find that the above sum equals  $\binom{i-j}{k-j}$ . The proof of  $A_{i,j}^s = \mathbf{B}_{i,i-j}^s \cdot \mathbf{C}^s(f)$  is similar.  $\square$

### 5.2 Determining the Existence of Solutions

Given a symmetric function  $f$ , the existence of  $g$  and  $h$  with corresponding degrees can be determined by an adapted version of Alg. 3 (which will be referred as Alg. 3<sup>s</sup>): step 7 is omitted, and step 8 is replaced by  $A \leftarrow A_{i,j}^s$ . The discussion of this slightly modified algorithm is similar to Sect. 4.2. However, computation of  $A_{i,j}^s$  requires only  $n+1$  evaluations of the function  $f$ , which can be neglected in terms of complexity. Consequently, time complexity to set up equations is only about  $\mathcal{O}(E^2)$ , and overall complexity of Alg. 3<sup>s</sup> becomes  $\mathcal{O}(E^3)$ .

Next, we will derive a method of very low (polynomial) complexity to determine the existence of  $g$  and  $h$  of low degree for a symmetric function  $f$ , but with the price that the method uses only sufficient conditions (i.e. some solutions may be lost). More precisely, we constrict ourselves to homogeneous functions  $g$  of degree  $e$  (i.e.  $g$  contains monomials of degree  $e$  only), and Eq. 8 becomes

$h_\gamma = A_{|\gamma|,e}^s \bigoplus_{|\beta|=e} \binom{\gamma}{\beta} g_\beta$ . Remember that  $h_\gamma = 0$  for  $|\gamma| > d$ , so the homogeneous function  $g$  is determined by the corresponding system of equations for all  $\gamma$  with  $|\gamma| = d+1, \dots, n$ . In this system, the coefficient  $A_{|\gamma|,e}^s$  is constant for  $\binom{n}{|\gamma|}$  equations. If  $A_{|\gamma|,e}^s = 0$ , then all these equations are linearly dependent (i.e. of type  $0 = 0$ ). On the other hand, if  $A_{|\gamma|,e}^s = 1$ , then a number of  $\binom{n}{|\gamma|}$  additional equations is possibly linearly independent. Consequently, if the sum of all possibly linearly independent equations for  $|\gamma| = d+1, \dots, n$  is smaller than the number of variables  $\binom{n}{e}$ , then non-trivial homogeneous functions  $g$  exist. This sufficient criterion is formalized by

$$\sum_{i=d+1}^n A_{i,e}^s \cdot \binom{n}{i} < \binom{n}{e}. \quad (11)$$

Given some degree  $e$ , the goal is to find the minimum value of  $d$  such that Eq. 11 holds. This can be done incrementally, starting from  $d = n$ . We formalized Alg. 4 of polynomial complexity  $\mathcal{O}(n^3)$ . This algorithm turned out to be very powerful (but not necessarily optimal) in practice, see Sect. 5.4 for some experimental results.

---

**Algorithm 3** Determine the degrees of  $g$  and  $h$  for symmetric  $f$

---

**Input:** A symmetric Boolean function  $f$  with  $n$  input variables.

**Output:** Degrees of specific homogeneous functions  $g$  and  $h$  such that  $fg = h$ .

- 1: **for**  $e$  from 0 to  $\lceil n/2 \rceil$  **do**
  - 2:   Let  $d \leftarrow n$ , number of equations  $\leftarrow 0$ , number of variables  $\leftarrow \binom{n}{e}$
  - 3:   **while** number of equations < number of variables **and**  $d+1 > 0$  **do**
  - 4:     Compute  $A \leftarrow A_{d,e}^s$ .
  - 5:     Add  $A \cdot \binom{n}{d}$  to the number of equations.
  - 6:      $d \leftarrow d - 1$ .
  - 7:   **end while**
  - 8:   Output  $\deg g = e$  and  $\deg h = d + 1$ .
  - 9: **end for**
- 

For a specified class of symmetric Boolean functions  $f$ , it is desirable to prove some general statements concerning the degrees of  $g$  and  $h$  for any number of input variables  $n$ . In the next section, we apply technique based on Alg. 4 in order to prove a theorem for the class of majority functions.

### 5.3 Fast Algebraic Attacks on the Majority Function

We denote by  $f$  the symmetric Boolean majority function with  $n \geq 2$  input variables, defined by  $f^s(i) := 0$  if  $i \leq \lfloor n/2 \rfloor$  and  $f^s(i) := 1$  otherwise. For example,  $\mathbf{T}^s(f) := (0, 0, 1)$  for  $n = 2$ , and  $\mathbf{T}^s(f) := (0, 0, 1, 1)$  for  $n = 3$ . The algebraic degree of this function is  $2^{\lfloor \log_2 n \rfloor}$ . In [7] and [16], it could be

proven independently that  $f$  has maximum algebraic immunity<sup>1</sup>. However, in the following theorem, we disclose the properties of  $f$  (and related functions) with respect to fast algebraic attacks.

**Theorem 2.** *Let  $f$  be the majority function with any  $n \geq 2$  input variables. Then there exist Boolean functions  $g$  and  $h$  such that  $fg = h$ , where  $d := \deg h = \lfloor n/2 \rfloor + 1$  and  $e := \deg g = d - 2^j$ , and where  $j \in \mathbb{N}^0$  is maximum so that  $e > 0$ .*

*Proof.* According to Eq. 8 for symmetric functions, we set up a system of equations in the coefficients of  $g$  only. The coefficients  $A_{i,j}^s$  of Eq. 9 have a simple form in the case of the majority function, namely  $A_{i,j}^s = \bigoplus_{k \geq d} \binom{i-j}{k-j} = \bigoplus_{k \geq d} \binom{i-j-1}{k-j-1} + \bigoplus_{k \geq d} \binom{i-j-1}{k-j} = \binom{i-j-1}{d-j-1} + 2 \bigoplus_{k \geq d} \binom{i-j-1}{k-j} = \binom{i-j-1}{d-j-1}$  for  $i > d$ . Additionally, we assume that  $g$  is homogeneous of degree  $e := d - 2^j$  where  $j$  is chosen maximum such that  $e \geq 1$ . According to Lucas' theorem, we find  $A_{d+i,e}^s = 0$  for  $1 \leq i < d - e$ . Consequently, only equations with  $|\gamma| = 2d - e, \dots, n$  may impose conditions on the coefficients  $g_\beta$ . As we can show that  $\sum_{i=0}^{e-1} \binom{n}{i} < \binom{n}{e}$ , the sufficient criterion (11) is satisfied, and nontrivial solutions exist.  $\square$

Algebraic and fast algebraic attacks are invariant with regard to binary affine transformations in the input variables. Consequently, Th. 2 is valid for all Boolean functions which are derived from the majority function by means of affine transformations. We notice that such a class of functions was proposed in a recent paper, discussing design principles of stream ciphers [5, 6].

## 5.4 Experimental Results

Application of Alg. 3<sup>s</sup> reveals that Th. 2 is optimal for the majority function where  $d = \lfloor n/2 \rfloor + 1$  (verification for  $n = 5, 6, \dots, 16$ ). An explicit homogeneous function  $g$  can be constructed according to  $g(\mathbf{x}) = \prod_{i=1}^e (x_{2i-1} + x_{2i})$ . We verified that Alg. 4 can discover the solutions of Th. 2.

In [7], a large pool of symmetric Boolean functions with maximum algebraic immunity is presented (defined for  $n$  even). One of these functions is the majority function, whereas the other functions are nonlinear transformations of the majority function. Application of Alg. 4 brings out that Th. 2 is valid for all functions  $f$  (verification for  $n = 6, 8, \dots, 16$ ). For some functions  $f$ , Alg. 4 finds better solutions than predicted by Th. 2 (e.g. for  $\mathbf{T}^s(f) := (0, 0, 0, 1, 1, 0, 1)$  where  $d = 3$  and  $e = 1$ ), which means that Th. 2 is not optimal for all symmetric functions. All solutions found by Alg. 4 can be constructed according to the above equation. Furthermore, Alg. 3<sup>s</sup> finds a few solutions which are (possibly) better than predicted by Alg. 4 (e.g. for  $\mathbf{T}^s(f) := (0, 0, 0, 1, 1, 1, 0)$  where  $d = 3$  and  $e = 2$ ), which means that Alg. 4 is not optimal for all symmetric functions.

<sup>1</sup> Notice that for  $n$  odd, it is verified in [16] up to  $n = 11$  that the majority function is the only symmetric Boolean function with maximum  $\mathcal{AI}$ .

## 6 Conclusions

In this paper, several efficient algorithms have been derived for assessing resistance of LFSR-based stream ciphers against conventional as well as fast algebraic attacks. This resistance is directly linked to the Boolean output function used. In many recent proposals, the number of inputs for this function is about 20 or larger. For such input sizes, verification of immunity against (fast) algebraic attacks by existing algorithms is infeasible. Due to improved efficiency of our algorithms, provable resistance of these stream ciphers against conventional and fast algebraic attacks has become amenable. Our algorithms have been applied to various classes of Boolean functions. In one direction the algebraic immunity of two families of Boolean power functions, the inverse function and Kasami type functions, have been determined. For the first time, the algebraic immunity  $\mathcal{AI}$  of a function with 20 variables is computed to be as large as  $\mathcal{AI} = 9$ . In another direction, our algorithms have been applied to demonstrate that large classes of Boolean functions with optimal algebraic immunity are very vulnerable to fast algebraic attacks. This applies in particular to classes of symmetric functions including the majority functions.

## Acknowledgments

This work is supported in part by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under grant number 5005-67322. The fifth author also receives partial funding through GEBERT RÜF STIFTUNG. We would like to thank Subhamoy Maitra for valuable discussions.

## References

1. F. Armknecht, and G. Ars. Introducing a New Variant of Fast Algebraic Attacks and Minimizing Their Successive Data Complexity. In *Progress in Cryptology - Mycrypt 2005*, LNCS 3715, pages 16–32. Springer Verlag, 2005.
2. F. Armknecht. Algebraic Attacks and Annihilators. In *WEWoRC 2005*, volume P-74 of *LNI*, pages 13–21. Gesellschaft für Informatik, 2005.
3. F. Armknecht. Improving Fast Algebraic Attacks. In *Fast Software Encryption 2004*, LNCS 3017, pages 65–82. Springer Verlag, 2004.
4. G. Ars. Application des Bases de Gröbner à la Cryptographie. Thèse de l'Université de Rennes, 2005.
5. A. Braeken, and J. Lano. Design Principles for LFSR-Based Stream Ciphers. To appear in *Selected Areas in Cryptography - SAC 2005*. Springer Verlag, 2005.
6. A. Braeken, J. Lano, N. Mentens, B. Preneel, and I. Verbauwhede. SFINKS: A Synchronous Stream Cipher for Restricted Hardware Environments. In *eSTREAM, ECRYPT Stream Cipher Project*, Report 2005/026. Available at <http://www.ecrypt.eu.org/stream>.
7. A. Braeken, and B. Preneel. On the Algebraic Immunity of Symmetric Boolean Functions. In *Progress in Cryptology - INDOCRYPT 2005*, LNCS 3797, pages 35–48. Springer Verlag, 2005.

8. P. Camion, C. Carlet, P. Charpin, and N. Sendrier. On Correlation-Immune Functions. In *Advances in Cryptology - CRYPTO 1991*, LNCS 576, pages 86–100. Springer Verlag, 1991.
9. A. Canteaut and M. Videau. Symmetric Boolean Functions. In *IEEE Transactions on Information Theory*, 2005, volume 51/8, pages 2791–2811.
10. C. Carlet, and P. Gaborit. On the Construction of Boolean Functions with a Good Algebraic Immunity. In *Boolean Functions: Cryptography and Applications - BFCA 2005*. See also the extended abstract entitled “On the Construction of Balanced Boolean Functions with a Good Algebraic Immunity” in the proceedings of *ISIT 2005*.
11. N. Courtois. Cryptanalysis of SFINKS. To appear in *Information Security and Cryptology - ICISC 2005*. Available at <http://eprint.iacr.org/2005/243>.
12. N. Courtois, and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - EUROCRYPT 2003*, LNCS 2656, pages 345–359. Springer Verlag, 2003.
13. N. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - CRYPTO 2003*, LNCS 2729, pages 176–194. Springer Verlag, 2003.
14. N. Courtois, and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *Advances in Cryptology - ASIACRYPT 2002*, LNCS 2501, pages 267–287. Springer Verlag, 2002.
15. D. K. Dalai, K. C. Gupta, and S. Maitra. Cryptographically Significant Boolean Functions: Construction and Analysis in Terms of Algebraic Immunity. In *Fast Software Encryption 2005*, LNCS 3557, pages 98–111. Springer Verlag, 2005.
16. D. K. Dalai, S. Maitra, and S. Sarkar. Basic Theory in Construction of Boolean Functions with Maximum Possible Annihilator Immunity. To appear in *Design, Codes and Cryptography*. Springer Verlag, 2006. Available at <http://eprint.iacr.org/2005/229>.
17. N. J. Fine. Binomial Coefficients Modulo a Prime. In *The American Mathematical Monthly*, 1947, volume 54, pages 589–592.
18. J.-C. Faugère, and G. Ars. An Algebraic Cryptanalysis of Nonlinear Filter Generators using Gröbner bases. In *Rapport de Recherche INRIA*, volume 4739, 2003.
19. P. Hawkes and G. G. Rose. Rewriting Variables: The Complexity of Fast Algebraic Attacks on Stream Ciphers. In *Advances in Cryptology - CRYPTO 2004*, LNCS 3152, pages 390–406. Springer Verlag, 2004.
20. W. Meier, E. Pasalic, and C. Carlet. Algebraic Attacks and Decomposition of Boolean Functions. In *Advances in Cryptology - EUROCRYPT 2004*, LNCS 3027, pages 474–491. Springer Verlag, 2004.
21. W. Meier, and O. Staffelbach. Nonlinearity Criteria for Cryptographic Functions. In *Advances in Cryptology - EUROCRYPT 1989*, LNCS 434, pages 549–562. Springer Verlag, 1990.
22. B. Mourrain, and O. Ruatta. Relations Between Roots and Coefficients, Interpolation and Application to System Solving. In *J. Symb. Comput.*, volume 33/5, 2002, pages 679–699.
23. Olver, P.J. On multivariate interpolation. In *Stud. Appl. Math.* 116 (2006) 201–240.
24. T. Siegenthaler. Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications. In *IEEE Transactions on Information Theory*, volume 30/5, 1984, pages 776–780.
25. T. Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. In *IEEE Transactions on Computer*, volume 34/1, 1985, pages 81–85.