Computing the Algebraic Immunity Efficiently

Frédéric Didier and Jean-Pierre Tillich

Projet CODES, INRIA Rocquencourt, Domaine de Voluceau,
78153 Le Chesnay cédex
{frederic.didier, jean-pierre.tillich}@inria.fr

Abstract. The purpose of algebraic attacks on stream and block ciphers is to recover the secret key by solving an overdefined system of multivariate algebraic equations. They become very efficient if this system is of low degree. In particular, they have been used to break stream ciphers immune to all previously known attacks. This kind of attack tends to work when certain Boolean functions used in the ciphering process have either low degree annihilators or low degree multiples. It is therefore important to be able to check this criterion for Boolean functions. We provide in this article an algorithm of complexity $O\left(m^d\right)$ (for fixed d) which is able to prove that a given Boolean function in m variables has no annihilator nor multiple of degree less than or equal to d. This complexity is essentially optimal. We also provide a more practical algorithm for the same task, which we believe to have the same complexity. This last algorithm is also able to output a basis of annihilators or multiples when they exist.

Keywords: Algebraic attacks, Algebraic immunity, Stream ciphers, Boolean functions, Annihilator, Low degree multiple.

1 Introduction

Algebraic attacks have proved to be a powerful class of attacks which might threaten both block and stream ciphers [CM03, Cou03, CP02, CDG05, Arm04]. The idea is to set up an algebraic system of equations verified by the key bits and to try to solve it. For instance, this kind of approach can be quite effective [CM03] on stream ciphers which consist of a linear pseudo-random generator hidden with non-linear combining functions acting on the outputs of the generator to produce the final output. For such an attack to work, it is crucial that the combining functions have low degree multiples or low degree annihilators. The reason for this is that it ensures that the algebraic system of equations verified by the secret key is also of small degree, which is in general essential for being able to solve it. This raises the fundamental issue of determining whether or not a given function has non-trivial low degree multiples or annihilators [Car04, MPC04, DGM04, BP05, DMS05]. The smallest degree for which this happens is called the *algebraic immunity* of the function.

Here we are going to address this issue for Boolean functions. Note that this is the main case of interest in this setting and that the algorithms presented here can be generalized to fields of larger size. This problem has already been

M.J.B. Robshaw (Ed.): FSE 2006, LNCS 4047, pp. 359–374, 2006.

considered in the literature. In [MPC04] three algorithms are proposed to decide whether or not a Boolean function in m variables has a non-trivial annihilator of degree at most d. Note that it is readily checked that finding a multiple of degree d for f amounts to finding an annihilator of degree d for $1+f$. Therefore it is sufficient for computing the algebraic immunity to be able to test whether or not a Boolean function has annihilators of a certain degree. All three algorithms have complexities of order $O\left(\binom{m}{d}^3\right)$. Gröbner bases have also been suggested to perform this task (see [FA03]) but it is still unclear if they perform better than the aforementioned algorithms.

Recall that a Boolean function g of degree d in m variables is an annihilator of a Boolean function f with support of size N iff the $k \stackrel{\text{def}}{=} \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{d}$ coefficients of the monomials of g verify a system of N linear equations (the equations express the fact that g has to be equal to zero at points for which f is equal to 1). We will suppose here that computing a value of f is in $O(1)$ and that it is the only thing we can do. This implies that the complexity of checking whether or not a Boolean function has an annihilator of degree at most d is at least of order $\Omega(k)$ since we need to check the value of f on at least k points.

We are going to present here a new algorithm (namely Algorithm 1 in this paper) which is able to prove the non-existence of annihilators of a certain degree efficiently. More precisely, we prove that for fixed d, the expected running time for our algorithm to prove that there is no non-trivial annihilator of maximum degree d is of order $O(k)$. This algorithm might fail to prove such a property for certain Boolean functions which have actually annihilators of this kind, we prove however that the proportion of such functions is negligible. In view of the previous lower bound, this algorithm is essentially optimal.

We also present another algorithm, namely Algorithm 2, which computes a basis for the annihilators of degree $\leq d$. We conjecture that the average running time of this algorithm is not worse than the average running time of Algorithm 1 when the latter succeeds in proving that there is no annihilator (which would then be of order $O(k)$), but we are only able to prove that its average running time is in this case of order $O(k(\log m)^2)$. Several remarks can be made here.

- It should be noted that the case of small d and rather large m is definitely interesting in cryptography. We wish to emphasize that it implies that we are far from checking all the entries of f to perform such a task. For instance, we are able to test that a random function in 64 variables has no non-trivial annihilator of degree 5 in a few minutes.
- It should be stressed here that our proof does not rely on any assumption about the linear system of equations which arises when a degree d annihilator is sought after. In particular, we do not assume that it behaves like a random system of linear equations, but we rely on a general result about the probability of not being able to recover erasures for linear codes for which the generalized weights distribution is known [Did05].
- The complexity of Algorithm 2 is not the same when there is an annihilator of the specified degree. We note that in this case, if we want to be sure that the output is indeed an annihilator, then the complexity of such an

algorithm is necessarily at least of order the number of entries of f. We present in Section 4 a way to modify this algorithm to make it faster in this case. Its output is in general only a basis of a space which contains the space of annihilators we look after. This algorithm and Algorithm 2 itself have been implemented. In our experiments, we found no example for which the output was not the space of annihilators itself.

2 Algebraic Immunity of a Boolean Function

In this section we recall basic facts about Boolean functions and algebraic immunity. We also introduce the tools we will need to analyze the complexity of our algorithms. In all this paper, we consider the binary vector space $\mathcal{B}(m)$ of m-variable Boolean functions, that is the space of functions from $\{0,1\}^m$ to $\{0,1\}$. It will be convenient to view $\{0,1\}$ as the field over two elements, what we denote by $\mathbf{F}_2$. It is well known that such a function f can be written in an unique way as an m-variable polynomial over $\mathbf{F}_2$ where the degree in each variable is at most 1 using the Algebraic Normal Form (ANF) :

$$f(x_0,\ldots,x_{m-1}) = \sum_{u\in\mathbf{F}_2^m} \alpha_u \left(\Pi_{i=0}^{m-1} x_i^{u_i}\right) \text{ where } \alpha_u \in \mathbf{F}_2, u=(u_0,\ldots,u_{m-1}).$$

By *monomial*, we mean in what follows, a polynomial of the form $\Pi_{i=0}^{m-1} x_i^{u_i}$ with $(u_0,\ldots,u_{m-1}) \in \mathbf{F}_2^m$. The *degree* of f is the maximum weight of the u's for which $\alpha_u \neq 0$. By listing the images of a Boolean function f over all possible values of the variables, that is $(f(x))_{x\in\mathbf{F}_2^m}$ (with some particular order over $\mathbf{F}_2^m$) we can also view it as a binary word of length $n \stackrel{\text{def}}{=} 2^m$. The *weight* of a Boolean function f is denoted by $|f|$ and is equal to $\sum_{x\in\mathbf{F}_2^m} f(x)$ (the sum being performed over the integers). We also denote in the same way the (Hamming) weight of a binary 2^m-tuple. A *balanced* Boolean function is a function with weight equal to half its length, that is $n/2$.

Dealing with algebraic immunity, we will be interested in the subspace $\mathcal{B}(d,m)$ of $\mathcal{B}(m)$ formed by all Boolean functions of degree $\leq d$. Note that the set of monomials of degree $\leq d$ forms a basis of $\mathcal{B}(d,m)$, we call it the *monomial basis*. By counting the number of such monomials we obtain that the dimension k of $\mathcal{B}(d,m)$ is given by $k = \sum_{i=0}^{d} \binom{m}{i}$.

As mentioned in the introduction, the algebraic immunity quantifies the immunity of a cryptosystem to some recent algebraic attacks. These attacks try to break a cryptosystem by solving an algebraic system involving the key bits. The equations involved often depend on a Boolean function f used in the ciphering process. The idea is that f can be replaced by its annihilators or multiples to obtain a new system of lower degree. The complexity of the attack depends on the degree of this system which is nothing but the algebraic immunity of f. Considering the pointwise product of two m-variable Boolean functions f and g, $\forall x \in \mathbf{F}_2^m, \quad f.g(x) = f(x)g(x)$, a function g is an *annihilator* of f if and only if $f.g$ is equal to 0, and a function g is a multiple of f if and only if there exists a Boolean function h such that $f.h$ is equal to g. Note that in the latter case, since

the existence of such a function is equivalent to the fact that the set of zeros of f is included in the set of zeros of g, we also have $f.g = g$, which is equivalent to $(f+1)g = 0$, i.e. g is an annihilator of $1+f$. By definition, the algebraic immunity of f is the smallest degree d such that f admits a non-trivial annihilator or multiple of degree d. By the previous remark, this is also the smallest degree of a non-trivial annihilator of f or $1+f$. In this paper, we will be interested in computing efficiently the algebraic immunity of a Boolean function. For achieving this aim, it is clearly sufficient to be able to find efficiently the smallest degree of a non-trivial annihilator of a Boolean function. Note that a Boolean function f admits a non-trivial annihilator of degree $\leq d$ iff the following set of equations with unknowns $\beta_u \in \mathbf{F}_2$, where $u = (u_0, \ldots, u_{m-1})$ ranges over all binary m-tuples of weight $\leq d$, has a non-zero solution

$$\left(\sum_{|u| \leq d} \beta_u x_0^{u_0} \ldots x_{m-1}^{u_{m-1}} = 0 \right)_{x=(x_0,\ldots,x_{m-1}) \in \mathbf{F}_2^m : f(x)=1} \tag{1}$$

This just expresses the fact that a Boolean function g of degree $\leq d$ (that is a function $g(x_0, \ldots, x_{m-1}) = \sum_{u=(u_0,\ldots,u_{m-1}) \in \mathbf{F}_2^m : |u| \leq d} \beta_u x_0^{u_0} \ldots x_{m-1}^{u_{m-1}}$) annihilates f if and only if for all points x at which f evaluates to 1, g is equal to 0. Our task will be to solve this linear system efficiently.

To estimate the complexity of the algorithms given in the following sections, we will use this result

Theorem 1. *Let λ be the fraction of m-variable Boolean functions of weight w with a non-trivial annihilator of degree $\leq d$, $k = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{d}$ and $n = 2^m$. Then $\lambda \leq \exp\left[\frac{n}{2^d}\left(\frac{k}{n}(d \ln 2 + 3) + \ln\left(1 - \frac{w}{n}\right)\right)\right]$.*

Proof. See [Did05]. □

3 Proving the Non-existence of an Annihilator Efficiently

We provide in this section an efficient algorithm which proves that a given Boolean function has no non-trivial annihilator up to a given degree. We hasten to say that we are going to present an improved version of this algorithm in the next section which will also be able to output a basis of the annihilator space up to a certain degree. The purpose of the algorithm presented here is that it can be analyzed rigorously and has average complexity of order $O(m^d)$ to prove that an m-variable Boolean function has no non-trivial annihilator of degree $\leq d$.

This algorithm makes heavily use of Gaussian elimination in its basic subroutines. What we call *lazy Gaussian elimination* in what follows simply consists in solving system (1) by inserting one by one the equations in a matrix that is kept reduced all the time.

When our algorithm looks for annihilators of degree $\leq d$ for an m-variable Boolean function f, it will apply lazy Gaussian elimination on subfunctions of f and will look for annihilators of lower degree. More precisely, it will apply lazy Gaussian elimination for other values of d and m, that is degrees d' in the

range $1 \leq d' \leq d$ and number of variables m' of the form $2d + 1 + \lceil \log m \rceil$. We need a bound on the probability that an m'-variable Boolean function has no non-trivial annihilator in $\mathcal{B}(d', m')$. We will focus on *random balanced Boolean function* which is one of the most significant random model in a cryptographic context. The analysis carried over here can of course be applied to more general probabilistic models.

Lemma 1. *Let f be a random balanced Boolean function in m variables. Consider an m'-variable Boolean function f' obtained by fixing $m - m'$ variables of f. Let $m' = 2d + 1 + \lceil \log m \rceil$ and $1 \leq d' \leq d$, then as m tends to infinity, the probability that there is a non-trivial degree d' annihilator for f' is upper bounded by $e^{-2^{2d-d'}(1+o(1))m}$.*

Proof. See appendix, Section A.

Moreover, we also need to give an upper bound on the expected running time of the previous lazy Gaussian elimination. It is quite likely that it is of order $O(k^3)$, but we are only able to prove the slightly weaker statement :

Lemma 2. *The expected running time of lazy Gaussian elimination applied to the f' of the previous lemma is upper bounded by $O(d'k'^3)$, where $k' = \binom{m'}{0} + \cdots + \binom{m'}{d'}$.*

Proof. See appendix, Section B.

Our algorithm is based on the the classical $(u, u+v)$ decomposition of a Boolean function. Given a Boolean function f of degree r in algebraic normal form, one can write $f(x_0, \ldots, x_{m-1}) = u(x_0, \ldots, x_{m-2}) + x_{m-1}v(x_0, \ldots, x_{m-2})$ where u and v are Boolean functions in $m-1$ variables. Note that for $x_{m-1} = 0$ we have $f(x_0, \ldots, x_{m-1}) = u(x_0, \ldots, x_{m-2})$ whereas for $x_{m-1} = 1$ we have $f(x_0, \ldots, x_{m-1}) = u(x_0, \ldots, x_{m-2}) + v(x_0, \ldots, x_{m-2})$. In other words, u is the restriction of f to the space $x_{m-1} = 0$ and $u + v$ is the restriction of f for $x_{m-1} = 1$. Moreover, the degree of u is at most r and the degree of v is at most $r - 1$.

Now, if there exists a non-trivial annihilator g for f of degree $\leq d$, then by decomposing g in $(u', u' + v')$ we either have:
- $u' \neq 0$ and $u'.u = 0$. This yields a non-trivial annihilator of u of degree $\leq d$.
- u' is zero but not v'. Then $v'.(u + v) = 0$ and we get a non-trivial annihilator of $u + v$ of degree $\leq d - 1$.

This simple remark is the underlying idea of our algorithm : to check that f has no non-trivial annihilator of degree $\leq d$ we check that neither u (or what is the same the restriction of f to $x_{m-1} = 0$) has a non-trivial annihilator of degree $\leq d$ nor $u + v$ (i.e. the restriction of f to $x_{m-1} = 1$) has a non-trivial annihilator of degree $\leq d - 1$. We perform this task recursively by decomposing u and $u+v$ further, up to the time the number of variables in the decomposition is equal to $2d + 1 + \lceil \log m \rceil$. This is illustrated by Figure 1. The first number in each couple represents the maximum annihilator degree to consider. The second

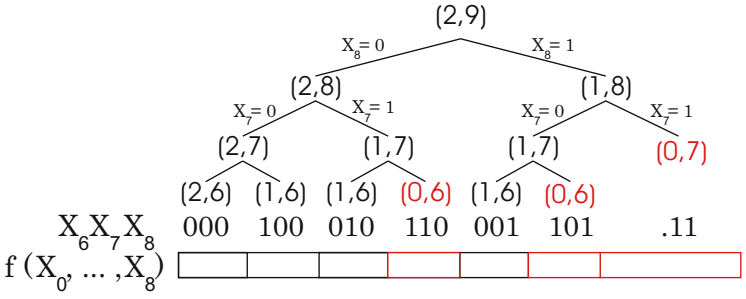

Fig. 1. Recursive decomposition of a 9-variable Boolean function used to find an annihilator of degree smaller than or equal to 2. The decomposition is performed up to 6-variable subfunctions. The bottom squares represent the domains of the subfunctions which are leaves of the decomposition.

number corresponds to the number of variables in the corresponding subfunction. We will continue the decomposition up to a certain depth as indicated above. Note that the decomposition may also stop because the degree of the annihilator to consider is zero.

It is important to notice the following fact

Fact 1. *The first level of decomposition of a Boolean function* $f(x_0, \ldots, x_{m-1})$ *corresponds for the left part to the restriction of* f *to* $x_{m-1} = 0$*, whereas the right part corresponds to the restriction of* f *to* $x_{m-1} = 1$*. More generally, consider a subfunction* f' *at the* l*-th level of the decomposition. It is associated to a path* $p = (p_1, \ldots, p_l) \in \mathbf{F}_2^l$ *starting from the root (that is* f*) by choosing either left or right children at each level. Here the value of* p_i *is* 0 *if we go from level* $i-1$ *to level* i *by choosing the left children and* 1 *otherwise. Notice that* f' *is the restriction of* f *to* $x_{m-1} = p_1, \ldots, x_{m-l} = p_l$*. Moreover, if we look for an annihilator of degree* d *for* f*, we search for an annihilator of degree* $d - |p|$ *for* f'*.*

More formally, our algorithm is the following

Algorithm 1. (Immunity verification) The input is an m-variable Boolean function f and a parameter d. The output is `Yes` if it can prove that there is no non-trivial annihilator of degree $\leq d$, `No` otherwise.

1. [decomposition] Recursively decompose the variable space of f according to d and this up to subfunctions in $2d + 1 + \lceil \log m \rceil$ variables. For each leaf, execute step 2.
2. [Subfunction verification] Use lazy Gaussian elimination with the correct degree to compute the annihilator space of the considered subfunction. If there is a non-trivial annihilator then go directly to step 4, otherwise continue.
3. [Immune] Output `Yes`: f has no non-trivial annihilator of degree $\leq d$.
4. [Unknown] Output `No`: we cannot prove the immunity of f.

Since in most cases of interest in cryptography, d is small, we can continue the decomposition quite far and still have with high probability that each subfunction does not admit any annihilator of the required degree. The point is that verifying with a classical algorithm that there is no annihilator for each subfunction is much faster than verifying it for the whole function. The real gain comes actually from the fact that the decomposition often ends at subfunctions for which we just have to check that the constant function 1 is not an annihilator. For this, we just have to find a corresponding value of the subfunction which is equal to 1. This can be done in $O(1)$ expected time. Hence, the more we decompose f, the faster we get but the more chance we have to output "`No`", since the probability that at least one of the subfunctions admits an annihilator increases. Choosing to decompose the functions up to a number of variables equal to $2d + 1 + \lceil \log m \rceil$ is a good tradeoff. For this choice we are able to prove the following

Theorem 2. *Let f be a random balanced m-variable Boolean function. Let d be fixed, then as m tends to infinity, Algorithm 2 runs in $O(k) = O\left(m^d\right)$ expected time and proves that there is no non-trivial annihilator up to degree d, except for a proportion of functions which is at most $O\left(e^{-2^d m(1+o(1))}\right)$.*

Proof. See Appendix C.

4 An Efficient Algorithm for Finding Annihilators

It the previous section, we considered a fixed depth decomposition, but the depth of decomposition may vary. For a given function f, the best decomposition for our problem is actually the smallest decomposition such that each subfunction has no non-trivial annihilator of the required degree.

By using a specific order on the monomials it is possible to work on this best decomposition. In practice, this new algorithm (Algorithm 2) will perform really well as shown by the running time presented later and will even be able to output an annihilator when it exists. However, analyzing directly Algorithm 2 seems to be more involved. We can only get a good upper bound on its complexity when there is a decomposition for which Algorithm 1 outputs `Yes`, and even in this case, the bound is slightly larger than the $O(k)$ bound of Algorithm 1.

4.1 The Algorithm

We describe here the new algorithm. Its relation with Algorithm 1 is not clear at first sight but will be explained in the next subsection. We will heavily use a specific order on the monomials and on elements of $\mathbf{F}_2^m$. This order is induced by the integer order on the set of integers $[\![0, 2^m - 1]\!]$ by viewing an m-tuple x (or a monomial X) as a binary representation of an integer. More precisely an element $(x_0, \ldots, x_{m-1}) \in \mathbf{F}_2^m$ or the corresponding monomial $X_0^{x_0} \ldots X_{m-1}^{x_{m-1}}$ are both associated to the integer $\sum_{i=0}^{m-1} x_i 2^i$. This identification allows us to compare monomials with points in $\mathbf{F}_2^m$ and to speak about intervals. For instance

for $a, b \in \mathbf{F}_2^m$ with $a < b$ we define $[a, b) \stackrel{\text{def}}{=} \{y \in \mathbf{F}_2^m : a \leq y < b\}$. We also index from 0 to $k-1$ the m-variable monomials of degree $\leq d$ arranged in increasing order.

We are now ready to describe Algorithm 2. The idea behind it is just to compute incrementally (for an x varying in $\mathbf{F}_2^m$) an annihilator basis of f restricted to the first x points.

Algorithm 2. (Incremental algorithm) The input is a Boolean function f in m variables and a number d. The output is a basis of the subspace of annihilators for f of degree $\leq d$.

1. [Initialization] Set the basis stack S to the constant function 1. Set the current monomial index i to 0. Initialize x to the first value in $\mathbf{F}_2^m$ such that $f(x) = 1$.
2. [Add basis element?] While the monomial of index $(i+1)$ is smaller than or equal to x, push it onto the top of S and increment i.
3. [Remove basis element?] If any, find the element of S closest to the top that evaluates to 1 on x. XOR it with all the other elements of S that evaluate to 1 on x and remove it from S.
4. [Skip to next monomial?] If the annihilator space is trivial, increment i, push the monomial of index i onto the top of S and set x to be this monomial.
5. [Loop?] If possible, increment x until $f(x) = 1$ and go back to step 2.
6. [end] Execute a last time step 2 and return the current basis stored in S.

The correctness of this algorithm follows from the two following lemmas

Lemma 3. *Given a monomial $X_0^{x_0} \dots X_{m-1}^{x_{m-1}}$ of degree d, the associated monomial function is zero on all entries strictly smaller than $x = (x_0, \dots, x_{m-1})$. Moreover, this function is equal to 1 on the interval $[x, x')$ where x' is the first point in $\mathbf{F}_2^m$ greater than x of weight $\leq d$.*

Proof. The aforementioned monomial function evaluates to 1 on an m-tuple $y = (y_0, \dots, y_{m-1})$ iff for all i such that $x_i = 1$ we also have $y_i = 1$. In such a case, we necessarily have $\sum_{i=0}^{m-1} y_i 2^i \geq \sum_{i=0}^{m-1} x_i 2^i$ which concludes the first part of the proof. Now, let us define i_0 to be the minimum of the set $\{i, x_i \neq 0\}$. Denote by x' the first point greater than x of weight less than or equal to d. The second assertion follows from the fact that all y' strictly between x and x' coincide with x for all positions greater than or equal to i_0. This implies that the aforementioned monomial evaluates to 1 at such y's. □

Lemma 4. *Let $A_{<x}(d)$, respectively $A_{\leq x}(d)$ be the set of m-variable Boolean functions g generated by the monomials of degree $\leq d$ which are smaller than or equal to x that satisfy $f(y)g(y) = 0$ for all entries $y \in \mathbf{F}_2^m$ smaller than x, resp. smaller than or equal to x. We use the same notation when x is a monomial by identifying x with the corresponding point in $\mathbf{F}_2^m$. For a value x chosen by the algorithm, we denote by x^- its previously chosen value. Then*

- *The set S obtained after completing step 2 is a basis for $A_{<x}(d)$.*
- *The set S obtained after completing step 3 is a basis of $A_{\leq x}(d)$.*

- *If a monomial X has been added in step 4, then just after completing step 4, $A_{<X}(d) = \{0, X\}$.*
- *For step 5, if no monomial has been added in step 4 just before, then after completing step 5 we have that $f(x) = f(x^-) = 1$ and $f(y) = 0$ for all y such that $x^- < y < x$. If a monomial X has been added in step 4 just before, we have after step 5 has been completed, that $A_{<X}(d) = \{0, X\}$.*

Proof. The key point is that by Lemma 3 the value of a Boolean function on x depends only on the monomials in its ANF smaller than or equal to x.

Let us check by induction on the number of times we perform step 2 that S is a basis of $A_{<x}(d)$ after step 2 has been completed. This is obviously true the first time we perform step 2. The assertions on step 3,4,5 are also clearly true the first time they are performed. Consider now a further step 2 for which all assertions about the previous steps 2,3,4,5 hold. There are two possibilities depending on whether the previous step 4 has added or not a monomial to S.
Case 1 : Assume that step 4 has added a monomial X.
Then $A_{<X}(d) = \{0, X\}$. All monomials added in step 2 are precisely the monomials X' of degree $\leq d$ such that $x \geq X' > X$. Note that for all these X' we have $X'(y)f(y) = 0$ for all $y < x$. This is also true for monomial X. Therefore all elements in S belong to $A_{<x}(d)$. All the elements in S are monomials and are necessarily independent. Consider now an element g in $A_{<x}(d)$. Let us first note that the ANF of g contains no monomial $< X$, otherwise the Boolean function formed by the part of the ANF of g involving only monomials $< X$ would belong to $A_{<X}(d)$, which is impossible. g is then necessarily a sum of monomials in S since we have put in S all monomials of degree $\leq d$ between X and x. S is therefore a basis of $A_{<x}(d)$.
Case 2 : Assume that step 4 has added no monomial. In this case, we have added to S all monomials between x^-(exclusive) and x(inclusive) of degree $\leq d$. Note that $f(y) = 0$ for all y such that $x^- < y < x$. From this, it is easy to check that all elements in S belong to $A_{<x}(d)$. They are also clearly independent. As before, consider a $g \in A_{<x}(d)$. Write $g = g_{\leq x^-} + g_{>x^-}$, where the ANF of $g_{\leq x^-}$ consists in the monomials of the ANF of g which are smaller than or equal to x^-. Notice from Lemma 3 that $g_{\leq x^-}$ belongs to $A_{\leq x^-}(d)$. It is therefore generated by elements in S. $g_{>x^-}$ is also clearly generated by the monomials added in step 2. S is therefore a basis of $A_{<x}(d)$.

The assertion on step 3 follows at once from the assertion on step 2, and the assertions about step 4 and 5 are straightforward. □

Remarks

1. Step 4 might seem odd at first sight, because it could be omitted without changing the correctness of the algorithm. However it is an essential step for having a low complexity algorithm. The point is that it avoids checking a lot of values of x satisfying $f(x) = 1$ which are useless for the algorithm.
2. In step 3, taking the element of S closest to the top of the stack is a heuristic which really helps in being fast. Define the trailing monomial of a Boolean function as the smallest monomial appearing in its ANF, the closer we are

to the top of the stack, the larger the trailing monomial will be. Taking elements the closest possible to the top of the stack will therefore tend to reduce the number of monomials in its ANF form. XOR'ing such a function to the other elements of the stack will therefore be more efficient.

3. Observe that Algorithm 2 necessarily outputs the empty-set if f evaluates to 1 in at least one point in any interval $[X_1, X_2)$ where X_1 and X_2 are two consecutive monomials of degree $\leq d$ (the monomials being ordered as explained above). This result comes from the second part of Lemma 3 and implies that such a function has no non-trivial annihilator in $\mathcal{B}(d, m)$. In particular, this applies to the inverse m-variable majority function which is defined by $f(x) = 1$ iff $|x| \leq \lfloor m/2 \rfloor$. This function is equal to 1 on all monomials of degree $\leq \lfloor m/2 \rfloor$ and has therefore no non-trivial annihilator of degree $\leq \lfloor m/2 \rfloor$.

4.2 Complexity Issues

Let us now explain the relationship between Algorithm 1 and Algorithm 2 when there exists a decomposition for which the output of Algorithm 1 is "`YES I can prove that there are no annihilator of degree` $\leq d$". Indeed let Algorithm 1^* be a modified version of Algorithm 1 where we use Algorithm 2 on subfunctions instead of lazy Gaussian elimination. Then we have

Proposition 1. *The complexity of Algorithm 2 is upper-bounded by the smallest*[1] *complexity of Algorithm* 1^* *when it outputs* `YES`*.*

Proof. Assume that we apply Algorithm 1^* to a certain decomposition for which the corresponding output is `YES`. Consider the first restriction of f for which we look for an annihilator of degree $\leq d$. Notice that this subfunction is the restriction of f on $[0, a)$, where the integer associated to a is equal to $2^{m'}$, m' being the number of variables on which the restriction depends. Algorithm 1^* coincides with Algorithm 2 on this subfunction and outputs that there are no annihilator. Notice now that the second subfunction considered by Algorithm 1^* is a restriction of f over an interval of the form $[a, b)$, and so on for all the other subfunctions which correspond to restrictions of f. Note that these intervals are consecutive and that they are examined by Algorithm 1^* in a consecutive order. Since Algorithm 1^* notices that every subfunction has no annihilator of the corresponding degree, Algorithm 2 also reduces in applying itself to each subfunction separately. The reduction is done implicitly because in this case an element in S will have only non zero coordinates on monomials in the current subfunction interval. The proposition follows. □

However, the problem is that this does not show that Algorithm 2 is as efficient as Algorithm 1. If we use Algorithm 2 instead of lazy Gaussian elimination on the subfunctions in Algorithm 1, we are only able to use the weak bound

[1] Recall that this means that the minimum is taken over all decompositions of f for which all subfunctions have no non-trivial annihilators.

$E(m', d') \leq n'k'^2$ instead of the bound of Lemma 2 ($E(m', d')$ representing the expected running time of Algorithm 2 on the corresponding subfunction). But we still have $E(m', 0) = O(1)$ and we obtain

Proposition 2. *The complexity of Algorithm* 1^* *applied to the decomposition of the previous section is* $O\left(k(\log m)^2\right)$.

Proof. Let us first bound the expected complexity of Algorithm 2 when applied to a subfunction f' on m' variables to find annihilators of degree $\leq d'$. When $d' \geq 1$ we will simply use the worst case complexity where we consider all n' points and for each of them, step 3 and 4 run in k'^2. Thus we have $E(m', d') = O(n'k'^2)$. In the case $d' = 0$, because of step 4, we will stop as soon as we get a point where $f' = 1$ and we have the same expected running time as the lazy Gaussian algorithm, that is $E(m', 0) = O(1)$.

In the end, with the new $E(m', d') = O(n'k'^2)$ we obtain a complexity in $O(k(\log m)^2)$ for Algorithm 1^* which is only slightly larger than the bound of complexity of Algorithm 1. □

4.3 Benchmarks

We have implemented a version of Algorithm 2 in C language running on a Pentium 4 at 2.6GHz with 1Gb memory. The (d, m) entry in the following tables means that we calculated the annihilator subspace in $\mathcal{B}(d, m)$ of a balanced m-variable Boolean function.

First of all, we ran some computations to find high degree annihilators. The results are presented in Figure 2. Notice that even for such large values of d corresponding to m, Algorithm 2 performs much better than lazy Gaussian elimination. Notice that the memory usage is better too, allowing to deal with cases where lazy Gaussian elimination ran out of memory.

d,m	2,6	3,8	4,10	5,12	6,14	7,16	8,18	9,20
k	22	93	386	1586	6476	26333	106762	431910
Lazy G	0s	0s	0s	0.1s	5s	2m30s	oom	oom
Algo 2	0s	0s	0s	0.01s	0.5s	20s	15m	12h

Fig. 2. Running time to check a high degree immunity for random balanced functions. The abbreviation `oom` means `Out Of Memory`. The lowest degree of a non-trivial annihilator of all the tested functions appeared to be the maximum possible, that is $\lceil \frac{m}{2} \rceil$ for an m-variable balanced Boolean function.

After this, we ran Algorithm 2 for small values of d and large values of m. The table in figure 3 clearly reflects the theoretical complexity in $O(k)$.

We have also checked the performances of our algorithm when there is an annihilator. The results are displayed in Figure 4. To make sure that the balanced Boolean function in m variables had an annihilator of a specified degree d, we took a random g in $\mathcal{B}(m)$ of degree d and weight w greater than 2^{m-1}. Then

d,m	4,32	5,32	6,32	7,32	3,64	4,64	5,64	2,128	3,128	2,256
k	4.10^4	2.10^5	1.10^6	4.10^6	4.10^4	6.10^5	8.10^6	8.10^3	3.10^5	3.10^4
Algo 2	1s	5s	30s	2m40s	1s	32s	8m	0.1s	32s	0.3s

Fig. 3. Running time to check low degree immunity of functions with a large number of variables. The functions were chosen randomly among balanced function and no non-trivial annihilators were found.

d,m	2,30	3,30	4,30	5,30	6,30
k	466	4526	3.10^4	2.10^5	8.10^5
Algo 2	13m	1h	3h45	-	-
Algo 2^*	1s	1s	4s	31s	4m34s

Fig. 4. Running times for Algorithm 2 and 2^* when $m = 30$ and a non-trivial annihilator of degree d exists

we chose $w - 2^{m-1}$ random values of g equal to 1 and made them equal to 0 to obtain a balanced function f. The f obtained this way has obviously $1 + g$ as an annihilator of degree d. We chose $m = 30$ for our experiments. In this case, Algorithm 2 has to check all 2^{30} points and this can be really long. It is possible to modify the algorithm to obtain a new algorithm (Algorithm 2^*) which runs much faster by avoiding to consider all points of f. Since we know that there is an annihilator, the first modification is to skip to the next monomial in step 4 as soon as the annihilator space is of dimension 1. To speed up the process even more, another modification is to consider (at random) only a fraction of all the points where f is equal to 1 (one half for example). However, with this last modification, the output of the new algorithm is just a candidate set for the annihilator subspace : it always contains the right annihilator space, but it could be much larger. In spite of this fact, we found an annihilator space of dimension 1 at the end of all our experiments.

5 Conclusion

This article presents two algorithms of low complexity which allow to detect whether or not a given Boolean function has low degree multiples or low degree annihilators. They allow to check this kind of property even for Boolean functions with a large number of variables. They can be used to build cryptographically strong functions by devising such functions with respect to other criterions (non-linearity, being balanced, resiliency, ...) and by checking afterwards that such functions are also immune against algebraic attacks.

The analysis of the complexity of Algorithm 2 is probably somehow pessimistic and it is likely that it can be improved. It should also be interesting to improve this algorithm when we apply it to a Boolean function admitting a non-trivial annihilator of the specified degree.

References

[Arm04] Frederik Armknecht. On the existence of low-degree equations for algebraic attacks. 2004. http://eprint.iacr.org/2004/185/.

[BP05] An Braeken and Bart Preneel. On the algebraic immunity of symmetric Boolean functions. 2005. http://eprint.iacr.org/2005/245/.

[Car04] Claude Carlet. Improving the algebraic immunity of resilient and nonlinear functions and constructing bent functions. 2004. http://eprint.iacr.org/2004/276/.

[CDG05] Nicolas Courtois, Blandine Debraize, and Eric Garrido. On exact algebraic [non-]immunity of S-boxes based on power functions. Cryptology ePrint Archive, Report 2005/203, 2005. http://eprint.iacr.org/2005/203.

[CM03] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. *Advances in Cryptology – EUROCRYPT 2003*, LNCS 2656:346–359, 2003.

[Cou03] Nicolas Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology-CRYPTO 2003*, volume 2729 of *LNCS*, pages 176–194. Springer Verlag, 2003.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. *Asiacrypt 2002*, LNCS 2501, 2002. http://eprint.iacr.org/2002/044.

[DGM04] Deepak Kumar Dalai, Kishan Chand Gupta, and Subhamoy Maitra. Results on algebraic immunity for cryptographically significant Boolean functions. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2004.

[Did05] Frédéric Didier. A new bound on the block error probability after decoding over the erasure channel. *Submited to IEEE IT*, July 2005. http://www-rocq.inria.fr/codes/Frederic.Didier/papers/Didier05.pdf.

[DMS05] Deepak Kumar Dalai, Subhamoy Maitra, and Sumanta Sarkar. Basic theory in construction of Boolean functions with maximum possible annihilator immunity. 2005. http://eprint.iacr.org/2005/229/.

[FA03] J.-C. Faugère and G. Ars. An algebraic cryptanalysis of nonlinear filter generator using Gröbner bases. *Rapport de Recherche INRIA*, 4739, 2003.

[MPC04] Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of Boolean functions. *Lecture Notes in Computer Science*, 3027:474–491, April 2004.

A Proof of Lemma 1

Let $n' = 2^{m'}$, $k' = \binom{m'}{0} + \binom{m'}{1} + \cdots + \binom{m'}{d'}$ and $n = 2^m$. Note that $n' \geq 2^{2d+1}m$. We start the proof by noticing that the probability that the weight of f' is w is given by

$$\frac{\binom{n-n'}{n/2-w}\binom{n'}{w}}{\binom{n}{n/2}}.$$

By Stirling's formula this probability can be shown to be of the form $O(e^{n'(h(w/n')-1)})$, with $h(x) = -x\ln x - (1-x)\ln(1-x)$. Using Theorem 1

we obtain that the probability that f' has an annihilator of degree $\leq d$ is of the form

$$\sum_{w=o}^{n'} O\left(e^{2^{2d+1}m\left[h(w/n')-1+\frac{1}{2^{d'}}\left(\ln(1-w/n')+\frac{k'(d'\ln 2+3)}{n'}\right)\right]}\right).$$

It is straightforward to check that $\frac{k'}{n'} = o(1)$ as m' tends to infinity. This implies that the $\frac{k'(d'\ln 2+3)}{n'}$ term in the exponent tends to zero. Moreover, we have a linear (in m) number of terms in the sum. Hence, we can upperbound it by its largest term times an $e^{mo(1)}$ factor. Putting all these facts together, we obtain that the probability that f' has an annihilator of degree d' is upperbounded by an expression of the form

$$e^{2^{2d+1}m\sup_{x\in[0,1]} g(x,d')(1+o(1))},$$

where $g(x,d') \stackrel{\text{def}}{=} h(x) - 1 + \frac{1}{2^{d'}}\ln(1-x)$. It can be checked that for $d' \geq 1$ we have that $\sup_{x\in[0,1]} g(x,d') \leq -\frac{1}{2^{d'+1}}$. The upper-bound given in Lemma 1 follows directly from this last remark.

B Proof of Lemma 2

Let $n' = 2^{m'}$. First of all, the maximum complexity of lazy Gaussian elimination is $O(n'k'^2)$. Moreover, by using Theorem 1, the processing of $w_0 = k'(\lceil d'\ln 2\rceil + 5)$ equations in lazy Gaussian elimination fails to prove the immunity of f' in $O(w_0k'^2)$ with a probability smaller than

$$e^{\frac{n'}{2^{d'}}\left(\frac{k'}{n'}(d'\ln(2)+3)+\ln\left(1-\frac{w_0}{n'}\right)\right)} \leq e^{-\frac{k'}{2^{d'-1}}(1+o(1))}$$

We can split the expected running time E in two. One part for the case where the Hamming weight of f' is greater than w_0 and one part where this is not the case. We get

$$E \leq \left[E_{w_0} + O(w_0k'^2) + e^{-\frac{k'}{2^{d'-1}}(1+o(1))}O(n'k'^2)\right] + P(|f'| < w_0)O(n'k'^2)$$

where E_{w_0} is the expected number of function evaluations we have to perform before actually finding w_0 points where f' is equal to 1. Notice that in this setting $k' = O(m'^{d'})$ and $e^{-\frac{k'}{2^{d'-1}}(1+o(1))}n' = O(1)$. The probability that we have to check i points to find w_0 points for which an m-variable balanced Boolean function evaluates to 1 is given by

$$\binom{i-1}{w_0-1}\frac{\binom{n-i}{n/2-w_0}}{\binom{n}{n/2}} = \binom{i-1}{w_0-1}\frac{(n)!(n/2)!(n/2)!}{(n-i)!(n/2-w_0)!(n/2+w_0-i)!}$$

We can upper-bound this expression by

$$\binom{i-1}{w_0-1}\frac{n^i}{2^i(n-i)^i} \leq \frac{\binom{i-1}{w_0-1}}{2^i}\frac{1}{(1-\frac{n'}{n})^{n'}}$$

Therefore we have for the expected number of function evaluations

$$E_{w_0} = \frac{1}{(1-\frac{n'}{n})^{n'}} \sum_{i=w_0}^{n'} \binom{i-1}{w_0-1} \frac{i}{2^i} = O(w_0)$$

In the same way, we have

$$P(|f'| < w_0) \leq \sum_{i=0}^{w_0-1} \frac{\binom{n'}{i}\binom{n-n'}{n/2-i}}{\binom{n}{n/2}} \leq w_0 \binom{n'}{w_0} \frac{\binom{n-n'}{\frac{n-n'}{2}}}{\binom{n}{n/2}} \leq \frac{w_0 {n'}^{w_0}}{2^{n'}} \frac{1}{(1-\frac{n'}{n})^{n'}}$$

So $P(|f'| < w_0)n' = O(1)$ which concludes the proof.

C Proof of Theorem 2

The proof of this theorem starts by first evaluating $N(m, d, m', d')$ which is the number of times we obtain an m'-variable subfunction for which we have to determine whether or not it has a degree $\leq d'$ annihilator.

Lemma 5. *We have* $N(m, d, m', d') = \binom{m-m'}{d-d'}$.

Proof. We use Fact 1 (and also the notation introduced there) and notice that such subfunctions correspond to paths $p = (p_1, p_2, \ldots, p_{m-m'}) \in \mathbf{F}_2^{m-m'}$ for which $|p| = d - d'$. The number of paths of this kind is obviously $\binom{m-m'}{d-d'}$. □

Let $E(m', d')$ be the expected running time of checking whether or not the m'-variable subfunction has a non-trivial annihilator of degree $\leq d'$. The crux of obtaining a low-complexity algorithm is that

Lemma 6. *For any* m', $E(m', 0) = O(1)$.

Proof. The idea is that in this case we just have to check whether or not the constant function 1 is an annihilator for the subfunction. This can be done by finding a point of the subfunction which evaluates to 1. Let $n = 2^m$ and $n' = 2^{m'}$. The probability that we have to check i points to find a point for which an m-variable balanced Boolean function evaluates to 1 is equal to $\frac{\binom{n-i}{n/2-1}}{\binom{n}{n/2}}$. Therefore the expected time is equal to

$$\sum_{i=1}^{n'} i \frac{\binom{n-i}{n/2-1}}{\binom{n}{n/2}} + n' \frac{\binom{n-n'}{n/2}}{\binom{n}{n/2}}.$$

The last term corresponds to the probability that the subfunction is the all-zero function and tends to 0 as n' tends to infinity. For the sum, we notice that $\frac{\binom{n-i}{n/2-1}}{\binom{n-i-1}{n/2-1}} = \frac{n/2-i+1}{n-i} \leq \frac{1}{2}$ and therefore the expected time is smaller than

$$\sum_{i=1}^{n'} i \left(\frac{1}{2}\right)^i + o(1) = O(1).$$ □

We finish the proof of the first assertion in Theorem 2 by noticing that the expected running time E of our algorithm is equal to

$$\sum_{i=1}^{d} N(m, d, 2d+1+\lceil \log m \rceil, i) E(2d+1+\lceil \log m \rceil, i) + E_0$$

where :

- $2d+1+\lceil \log m \rceil$ is the number of variables at which we stop the decomposition.
- E_0 is the expected time spent in checking the subfunctions for which $d'=0$.

From Lemma 6, E_0 depends on the number of subfunctions for which $d'=0$. It corresponds to paths of weight exactly d in the decomposition tree and we get

$$E_0 = O\left(\binom{m-(2d+1+\lceil \log m \rceil)}{d}\right).$$

We use Lemma 2 to bound the $E(2d+1+\lceil \log m \rceil, i)$'s (and by noticing that the k''s involved in this lemma are of the form $O\left((2d+1+\log m)^i\right)$ and we obtain that

$$\begin{aligned} E &= O\left(\binom{m-2d-1-\lceil \log m \rceil}{d} + \sum_{i=1}^{d} \binom{m-2d-1-\lceil \log m \rceil}{d-i} i(2d+1+\log m)^{3i}\right) \\ &= O\left(m^d + \sum_{i=1}^{d} m^{d-i} i(2d+1+\log m)^{3i}\right) \\ &= O(m^d) = O(k). \end{aligned}$$

It should be noted that it is really the time spent in proving that subfunctions have not the constant 1 as annihilator which is responsible for the $O(k)$ term in the average complexity. The time spent in proving that the other subfunctions have annihilators of larger degree is negligible.

To prove that there is only a negligible part of functions for which Algorithm 1 answers "`No`", we proceed as follows. The probability P_1 that Algorithm 2 exits for values of d' greater than 0 can be bounded by using lemmas 5 and 1 by

$$\begin{aligned} P_1 &\leq \sum_{d'=1}^{d} \binom{m-2d-1-\lceil \log m \rceil}{d-d'} e^{-2^{2d-d'}(1+o(1))m} \\ &\leq O(m^d) e^{-2^{d+1}(1+o(1))m} + e^{-2^d(1+o(1))m} \\ &\leq e^{-2^d(1+o(1))m} \end{aligned}$$

The probability P_2 that Algorithm 1 exits for values of d' equal to 0 is clearly upper-bounded by the number of subfunctions for which d' equals 0 times the probability that such a subfunction evaluates to 0 on a domain of size $2^{2d+1+\lceil \log m \rceil}$. This last probability is clearly of the form $O\left(e^{-2^{2d+1}m(\ln 2+o(1))}\right)$. Summing P_1 and P_2 yields an upper-bound for the probability we wish to bound and leads to the second assertion in Theorem 2.