Computing the algebraic immunity efficiently

Frédéric Didier, Jean-Pierre Tillich
INRIA, projet CODES

Outline

1. Introduction
2. A first algorithm
3. A more “efficient” version
4. Benchmarks

Part 1

Introduction

Boolean functions and annihilator space

Let
- f be an m-variable Boolean function ($\mathbf{F}_2^m \rightarrow \mathbf{F}_2$)
- d be a given degree

Goal : compute the annihilator space of degree $\leq d$ for f

$$\mathcal{A}_d(f) \stackrel{\text{def}}{=} \{g : \quad \deg g \leq d, \quad f(x)g(x) = 0 \ \forall x \in \mathbf{F}_2^m\}$$

Reason : Algebraic immunity of f $\stackrel{\text{def}}{=}$

smallest d such that $\mathcal{A}_d(f) \cup \mathcal{A}_d(1+f) \neq \{0\}$

Basic algorithm : Gaussian elimination

ANF : $$g(x) = \sum_{|y| \leq d} g_y \prod_{i=1}^{m} x_i^{y_i} \qquad g_y \in \mathbf{F}_2,\ y \in \mathbf{F}_2^m$$

- Number of coeffs (g_y) : $k \stackrel{\text{def}}{=} \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{d}$
- $\forall x$ such that $f(x) = 1 \ \longrightarrow\ $ linear equation $g(x) = 0$
- Number of equations : $|f| \stackrel{\text{def}}{=} \sum_x f(x)$
- GE complexity for the $|f| \times k$ linear system : $O(2^{2m}k)$

State of the art

Lazy Gaussian elimination worst $O(2^m k^2)$

average $O(k^3)$

1. Take the equations 1 by 1 (randomly)
2. Update a basis of the generated space so far
3. Stop as soon as we get a space of dimension k

[Meier Pasalic Carlet 04] $O(k^3)$

[Armknecht Carlet Gaborit Künzli Meier Ruatta 06]
(Accepted in eurocrypt 2006) $O(k^2)$

Our Algorithm complexity

- Efficient for d small and m large

For d fixed and $m \to \infty$
our algorithm can prove that $\mathcal{A}_d(f) = \{0\}$ in $O(k)$
except for a vanishing proportion $\mathbf{P}$ of balanced functions f

f balanced ($|f| = 2^{m-1}$) $\supset$ $\mathcal{A}_d(f) = \{0\}$ $\supset$ verifiable in $O(k)$

General case : no results but good practical behavior

Part 2

A first algorithm

Order on $\mathbf{F}_2^m$

$$\begin{array}{ccc} \mathbf{F}_2^m & \longrightarrow & [0, 2^m) \\ (x_1, \ldots, x_m) & \longmapsto & \sum_{i=1}^m x_i 2^{i-1} \end{array}$$

Representation of f

$f(0)f(1)\ldots f(2^m - 1) \quad \Leftrightarrow \quad \boxed{\quad f \quad}$

In particular, if we split the interval in two

$f(x_1, \ldots, x_{m-1}, 0)$	$f(x_1, \ldots, x_{m-1}, 1)$

(u,u+v) decomposition

$$f(x_1, \ldots, x_m) = u(x_1, \ldots, x_{m-1}) + x_m v(x_1, \ldots, x_{m-1})$$

f	u	$u+v$
$g \in \mathcal{A}_d(f) \setminus \{0\}$	u'	$u'+v'$

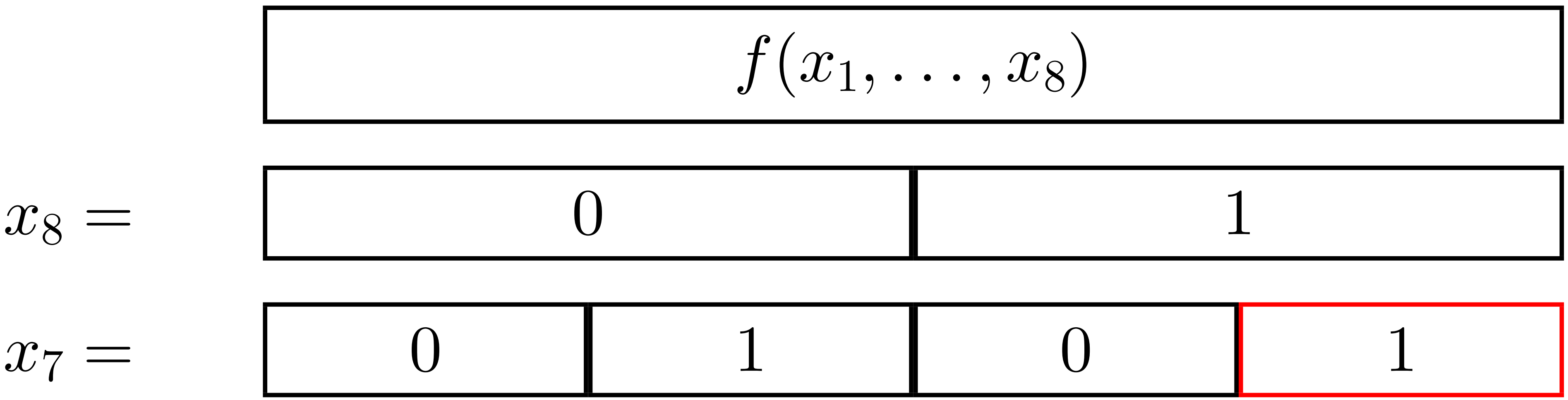
We have either

$u' \neq 0$ and $u.u' = 0$ $\qquad u' \in \mathcal{A}_d(u)$

or

$u' = 0$, $v' \neq 0$ and $(u+v).v' = 0$ $\qquad v' \in \mathcal{A}_{d-1}(u+v)$

Recursive decomposition ($d = 2, m = 8$)



For each subfunction ($\stackrel{\text{def}}{=} f'$) we want $\mathcal{A}_{d'}(f') = \{0\}$.

We stop at a certain depth or when $d' = 0$ (in red here).

Recursive decomposition ($d = 2, m = 8$)

<table>
<tr><td></td><td colspan="8">$f(x_1, \ldots, x_8)$</td></tr>
<tr><td>$x_8 =$</td><td colspan="4">0</td><td colspan="4">1</td></tr>
<tr><td>$x_7 =$</td><td colspan="2">0</td><td colspan="2">1</td><td colspan="2">0</td><td colspan="2">1</td></tr>
<tr><td>$x_6 =$</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td colspan="2"></td></tr>
</table>

For each subfunction ($\stackrel{\text{def}}{=} f'$) we want $\mathcal{A}_{d'}(f') = \{0\}$.

We stop at a certain depth or when $d' = 0$ (in red here).

Immunity verification, input = f,m,d

1. [Decomposition] recursively decompose f until subfunctions in $m' = 2d + 1 + \lceil \log(m) \rceil$ variables or with $d' = 0$ and for each of them, Apply step 2.

2. [Subfunction verification] execute Lazy Gaussian Elimination with the correct degree d'. If the subfunction admits an annihilator, go to step 4.

3. [Immune] return `Yes`, we proved that $\mathcal{A}_d(f) = \{0\}$.

4. [Failure] return `No`, we cannot prove the immunity of f.

Complexity analysis, f balanced, $m \to \infty$

Counting the subfunctions of each kind (d', m') we get

$$\mathbf{C} = \sum_{d'>0} O(m^{d-d'})\mathbf{C}_{\mathsf{LGE}}(d', m') + O(m^d)\mathbf{C}_0$$

Using a result in [Didier 05], $\mathbf{C}_{\mathsf{LGE}}(d', m') = O((\log m)^{3d'})$

Key point is that for $d' = 0$ subfunctions, $\mathbf{C}_0 = O(1)$

- We get the final complexity $\boxed{\mathbf{C} = O(m^d) = O(k)}$
- And in the same way $\boxed{\mathbf{P} \leq e^{-2^d m(1+o(1))}}$

Remarks

In our model :

- Only evaluations of f are allowed
- Computing $f(x)$ is in $O(1)$

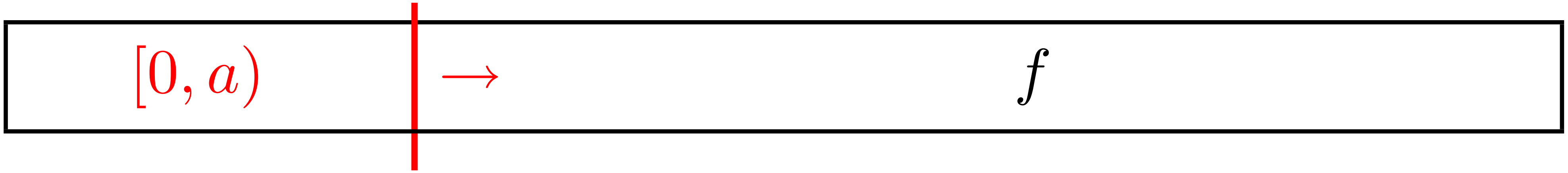
we have

- The $O(k)$ is optimal to show that $\mathcal{A}_d(f) = \{0\}$
- Far from checking all the points ($k \ll 2^m$)
- If there is an annihilator, minimal complexity $\Omega(2^m)$

Part 3

A more “efficient” version

Basic idea



$$f_{<a} \stackrel{\text{def}}{=} \text{ restriction of } f \text{ to } [0, a)$$

The Algorithm will compute incrementally the space

$$\mathcal{A}_d(f_{<a}) \stackrel{\text{def}}{=} \{g_{<a}, \quad \forall x \in [0, a), \quad f(x)g(x) = 0\}$$

Connection with the previous algorithm

Theorem $\quad g_{<a}(x) = \sum_{|y| \leq d,\ y<a} g_y \prod x_i^{y_i}$ in an unique way

For any decomposition on which Algorithm 1 returns `Yes` :

$\boxed{[0, a) \sim f'} \quad \Rightarrow \quad g_y = 0 \quad \forall y \in [0, a)$

$\boxed{[0, a)}\boxed{[a, b) \sim f'}$ And so on ...

Incremental Algorithm, input = f,m,d

For a **from** 0 **to** $2^m - 1$ **do**

1.[Fact] S is a stack containing a basis of $\mathcal{A}_d(f_{<a})$.

2.[Add element ?] **if** $(|a| \leq d)$ push $\prod x_i^{a_i}$ onto the top of S.

3.[Remove element ?] **if** $(f(a) = 1)$ XOR the element closest to the top that evaluates to 1 on a (if any) with all the other elements that evaluate to 1 on a. Remove it from S.

4.[Skip to next monomial ?] **if** $(S$ is empty$)$ we can go directly to the next a such that $|a| \leq d$.

Remarks

Drawback :

- Difficult to analyze the real complexity
- $\mathbf{C}_{\mathsf{LGE}}(d', m') = O(d'k'^3) \rightarrow \mathbf{C}_{\mathsf{Inc}}(d', m') = O(2^{m'}k'^2)$

  for the same $\mathbf{P}$, complexity in $O(k(\log m)^2)$

But in practice :

- It works on the optimal decomposition
- It performs well even when we don't have $d \ll m$
- It can find annihilators efficiently

Part 4

Benchmarks

Immunity verification (P4/2.6Ghz/1Gb)

Time for $\mathcal{A}_d(f)$ and m-variable random balanced f

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	1.10^5	4.10^5
LGE	0s	0s	0s	0.1s	5s	2m30s	oom	oom
Inc	0s	0s	0s	0.01s	0.5s	20s	15m	12h

We found $\mathcal{A}_d(f) = \{0\}$ in all our experiments

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

Computing an annihilator (P4/2.6Ghz/1Gb)

Time for random balanced function having at least one degree d annihilator, we found it back in all our experiment

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

Algorithm Inc* : same as Inc except that in step 4,
we skipped to the next monomial if $(|S| \leq 1)$

Conclusion

Analysis without any assumption on the linear system

Allows the construction of cryptographically strong functions

- Devise such functions with respect to other criteria
- Check afterwards their immunity to algebraic attacks

May be used in some cases to find an annihilator efficiently