# Computing Boolean Border Bases

Jan Horáček, Martin Kreuzer, and Ange-Salomé Messeng Ekossono
Faculty of Informatics and Mathematics
University of Passau
94030 Passau, Germany
Email: Jan.Horacek,Martin.Kreuzer,Ange-Salome.MessengEkossono@uni-passau.de

***Abstract*—Given a 0-dimensional polynomial system in a polynomial ring over $\mathbb{F}_2$ having only $\mathbb{F}_2$-rational solutions, we optimize the Border Basis Algorithm (BBA) for solving this system by introducing a Boolean BBA. This algorithm is further improved by optimizing the linear algebra steps. We discuss ways to combine it with SAT solvers, optimized methods for performing the combinatorial steps involved in the algorithm, and various approaches to implement the linear algebra steps. Based on our C++ implementation, we provide some timings to compare sparse and dense representations of the coefficient matrices.**

## I. Introduction

Frequently, a cryptographic attack can be formulated as the solution of a large polynomial system over $\mathbb{F}_2$ for which we have important additional information: the input is sparse, the system has only finitely many solutions, and all solutions are defined over the field $\mathbb{F}_2$. Hence we may add the *field equations* $x_i^2 + x_i = 0$ to the polynomial system without changing the set of solutions. Viewed from a different perspective, we are working in the ring $\mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2+x_1, \ldots, x_n^2+x_n\rangle$ whose elements are represented by *Boolean polynomials*, i.e., polynomials with only squarefree terms in their support.

The usage of Gröbner bases for solving systems of Boolean polynomials has received widespread attention (e.g., see [3], [2], [18], [9]). Specialized algorithms have been implemented in the package `PolyBoRi` (see [1]) and in Risa/Asir (see [17]), and they achieved good timings. However, one disadvantage of the Gröbner basis method in this context is that the formation of S-polynomials leads to polynomials of significantly larger degree than the current degree we are working in. Furthermore, the S-polynomials have to be reduced against the currently known part of the Gröbner bases. This involves a potentially exponential number of reduction steps, each of which reduces the sparsity of the input significantly. The result is frequently a rapid increase in memory consumption which breaks down the calculation. To overcome this problem, it has been suggested to use Pommaret bases which restrict the possible multiplications of leading terms (see [7], [8]), but the overall performance was similar.

In this paper we suggest a more fundamental improvement: using the border basis algorithm, we apply only *linear syzygies* to the input polynomials. Thus, by carefully controlling the size of the computational *universe*, i.e., the vector space in which all linear algebra steps take place, and the sizes of the supports of the intermediate polynomials, we keep the memory consumption of the algorithm under control at the cost of a somewhat slower progress.

One way to speed it up further is to input additional sparse polynomials coming from the conflict clauses learned through a parallel execution of a SAT-solver. This conversion has been used for Gröbner basis methods (see [6], [19], [16]), and also in the border basis setting it yields promising prospects. Another way to achieve competitive running times is to optimize the implementation. We describe improved methods for performing the necessary combinatorial operations with order ideals and monomial ideals to reduce their contribution to the total running time of the algorithm. Moreover, we look at different ways to implement terms, polynomials, and coefficient matrices in the Boolean setting and examine how they affect the performance.

The paper is organized as follows. After recalling the definition of border bases and the border basis algorithm (BBA), we construct in Section IV a version of this algorithm which is tailored to systems of Boolean polynomials and call it the Boolean BBA. In Section V we study improvements of the critical linear algebra steps in the Boolean BBA, and in Section VI we combine it with a SAT solver to get additional input polynomials from learned conflict clauses. Next, Section VII contains improvements and speed-ups related to the implementation of the operations with order ideals necessary for the Boolean BBA. Finally, in Section VIII we discuss the implementation of the linear algebra steps of the algorithm and provide some timings to compare the feasibility of sparse and dense representations. Unless specifically stated otherwise, we use the definitions and adhere to the notation introduced in [13] and [14],

## II. Border Bases

In the following we let $K$ be a field and $P = K[x_1, \ldots, x_n]$ a polynomial ring over $K$. An ideal $I$ in $P$ is called *0-dimensional* if $P/I$ is a finite dimensional $K$-vector space. One of the basic ideas of border bases is to find vector space bases of $P/I$ of the following form.

*Definition 2.1:* Let $\mathbb{T}^n = \{x_1^{\alpha_1} \cdots x_n^{\alpha_n} \mid \alpha_i \geq 0\}$ be the monoid of terms in $P$. A finite subset $\mathcal{O}$ of $\mathbb{T}^n$ is called an **order ideal** if $t \in \mathcal{O}$ and $t' \mid t$ imply $t' \in \mathcal{O}$. Given an order ideal $\mathcal{O}$, we call $\partial\mathcal{O} = (x_1\mathcal{O} \cup \cdots \cup x_n\mathcal{O}) \setminus \mathcal{O}$ the **border** of $\mathcal{O}$

For every term ordering $\sigma$, the set $\mathcal{O}_\sigma(I) = \mathbb{T}^n \setminus \mathrm{LT}_\sigma(I)$ is an order ideal. The following type of systems of generators

of $I$ allow us to rewrite all polynomials in terms of an order ideal $\mathcal{O}$.

*Definition 2.2:* Let $\mathcal{O} = \{t_1, \ldots, t_\mu\}$ be an order ideal, and let $\partial\mathcal{O} = \{b_1, \ldots, b_\nu\}$ be its border.

(a) A set of polynomials $G = \{g_1, \ldots, g_\nu\}$ is called an **$\mathcal{O}$-border prebasis** if $g_j = b_j - \sum_{i=1}^{\mu} c_{ij}\, t_i$ with $c_{1j}, \ldots, c_{\mu j} \in K$ for $j = 1, \ldots, \nu$.

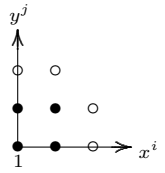
(b) An $\mathcal{O}$-border prebasis $G \subset I$ is called an **$\mathcal{O}$-border basis** of $I$ if the residue classes $\overline{\mathcal{O}} = \{\bar{t}_1, \ldots, \bar{t}_\mu\}$ in $P/I$ form a $K$-basis of $P/I$.

Note that the existence of an $\mathcal{O}$-border prebasis $G \subset I$ implies that $\overline{\mathcal{O}}$ generates the $K$-vector space $P/I$, and that an $\mathcal{O}$-border basis of $I$ is automatically a system of generators of the ideal $I$. One way of constructing an $\mathcal{O}$-border basis is to extend a Gröbner basis as follows.

*Example 2.3:* Let $I$ be a 0-dimensional ideal in $P$. Choose a term ordering $\sigma$, compute the reduced $\sigma$-Gröbner basis $G$ of $I$, let $\mathcal{O}_\sigma(I) = \mathbb{T}^n \setminus \mathrm{LT}_\sigma(I) = \{t_1, \ldots, t_\mu\}$, and let $\partial\mathcal{O}_\sigma(I) = \{b_1, \ldots, b_\nu\}$. Then the set $G' = \{g_1, \ldots, g_\nu\}$, where $g_j = b_j - \mathrm{NF}_G(b_j)$ for $j = 1, \ldots, \nu$, is an $\mathcal{O}_\sigma(I)$-border basis of $I$ which contains $G$ (see [14], Prop. 6.4.18).

Not every border basis is constructed by extending a reduced Gröbner basis, as the next example shows.

*Example 2.4:* Let $K = \mathbb{Q}$, let $P = K[x, y]$, and let $I = \langle x^2 + xy + 1,\, y^2 + 2xy + 1\rangle$. Then $\mathcal{O} = \{1,\, x,\, y,\, xy\}$ is an order ideal and the residue classes of the terms in $\mathcal{O}$ form a $K$-basis of $P/I$. Notice that this order ideal is not of the form $\mathbb{T}^n \setminus \mathrm{LT}_\sigma(I)$ for any term ordering $\sigma$, since $xy \in \mathrm{LT}_\sigma(I)$ for both cases $x >_\sigma y$ and $y >_\sigma x$. This order ideal and its border $\partial\mathcal{O} = \{x^2,\, x^2y,\, y^2x,\, y^2\}$ can be illustrated as follows.




In this setting the set $\{x^2 + xy + 1, x^2y + x - y,\, xy^2 - x + 2y,\, y^2 + 2xy + 1\}$ is an $\mathcal{O}$-border basis of $I$ which is not constructed from a Gröbner basis as above.

Let us see an example of an order ideal $\mathcal{O}$ and an $\mathcal{O}$-border basis $G$ which is not constructed from a Gröbner basis such that the ideal $I = \langle G\rangle$ contains the field ideal.

*Example 2.5:* In the polynomial ring $P = \mathbb{F}_2[w, x, y, z]$, consider the ideal $I = \langle xz + yz + 1,\, yw + xw + 1,\, ywz\rangle + F$ where $F = \langle w^2 + w,\, x^2 + x,\, y^2 + y,\, z^2 + z\rangle$ is the field ideal. Then one can check that $I$ has a border basis for the order ideal $\mathcal{O} = \{1, w, x, y, z, xw, xy, yz, zw\}$. This border basis is not constructed from a Gröbner basis, since $xz >_\sigma yz$ and $yw >_\sigma xw$ cannot hold simultaneously.

## III. The Border Basis Algorithm

Based on suggestions in [15], the following algorithm for computing a border basis of a 0-dimensional ideal $I$ in $P = K[x_1, \ldots, x_n]$ was introduced in [11]. For a list of polynomials $V$, we let $V^+ = V \cup x_1V \cup \cdots \cup x_nV$. Moreover, for a set of terms $S$, we let $\langle S\rangle_{\mathrm{OI}}$ be the order ideal spanned by $S$, i.e., the set of all terms dividing one of the terms in $S$. A finite set of polynomials is called LT-interreduced (with respect to some term ordering) if they are monic and have pairwise distinct leading terms.

*Algorithm 3.1:* **(The Border Basis Algorithm (BBA))**
*Input:* generators $\{f_1, \ldots, f_s\}$ of $I$ and a term ordering $\sigma$
*Output:* the $\mathcal{O}_\sigma(I)$ border basis of $I$

1: Let $U = \langle \mathrm{Supp}(f_1) \cup \cdots \cup \mathrm{Supp}(f_s)\rangle_{\mathrm{OI}}$.
2: Let $V$ be an LT-interreduced basis of $\langle f_1, \ldots, f_s\rangle_K$.
3: **repeat**
4:  **repeat**
5:  Compute a set of polynomials $W'$ such that $V \cup W'$ is an LT-interreduced basis of $\langle V^+\rangle_K$.
6:  **repeat**
7:  $W := \{w \in W' \mid \mathrm{LT}_\sigma(w) \in U\}$
8:  $U' := \langle \bigcup_{w\in W} \mathrm{Supp}(w) \setminus U\rangle_{\mathrm{OI}}$
9:  $U := U \cup U'$
10:  **until** $U' = \emptyset$
11:  $V := V \cup W$
12:  **until** $W = \emptyset$
13:  $\mathcal{O} := U \setminus \langle \mathrm{LT}_\sigma(V)\rangle$
14:  Let $U_{\mathrm{old}} := U$ and $U := U^+$.
15: **until** $\partial\mathcal{O} \subset U_{\mathrm{old}}$
16: Apply `FinalReduction`$(V, \mathcal{O})$ and return the result.

Here the algorithm `FinalReduction` is the one given in [11], Prop, 17. Its purpose is to extract the desired border basis from $\langle V\rangle_K$. The condition $\partial\mathcal{O} \subset U$ ensures that such a border basis exists. The algorithm `FinalReduction` has a low time complexity and will be ignored in the following. Notice that we used already the Improved BBA of [11], Prop. 21. Further improvements are suggested in [12].

The order ideal $U$ in this algorithm is called the (computational) **universe**. Since we are performing linear algebra operations in the vector space $\langle U\rangle_K$, our goal is to keep it as small as possible at all times. The loop in steps (4) - (12) of the algorithm computes the **$U$-stable span** of $V$.

## IV. A Boolean BB Algorithm

For the tasks described in the introduction, we now use $K = \mathbb{F}_2$ and work in the ring of **Boolean polynomials**

$$R = \mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n\rangle.$$

The elements of this ring will be represented by polynomials whose support consists of squarefree terms. The set of squarefree terms will be denoted by $\mathbb{S}^n$. The ideal $F = \langle x_1^2 + x_1, \ldots, x_n^2 + x_n\rangle$ is called the **field ideal**, since it defines the set of points in $\mathbb{A}^n$ with coordinates in the field $\mathbb{F}_2$.

*Definition 4.1:* Let $\mathcal{S}$ be a set of terms in $\mathbb{T}^n$. Then the set $\mathcal{S}^{\,\mathrm{sf}} = \mathcal{S} \cap \mathbb{S}^n$ is called the **squarefree part** of $\mathcal{S}$.

A border basis of an ideal $I$ in $P$ which contains the field ideal $F$ has the following shape.

*Proposition 4.2:* Let $\mathcal{O}$ be an order ideal in $\mathbb{T}^n$, and let $I$ be an ideal in $P$ which contains $F$ and has an $\mathcal{O}$-border basis.

(a) We have $\mathcal{O} \subset \mathbb{S}^n$ and a disjoint union $\partial\mathcal{O} = (\partial\mathcal{O})^{\mathrm{sf}} \cup x_1\mathcal{O}_1 \cup \cdots \cup x_n\mathcal{O}_n$, where $\mathcal{O}_i$ is the set of all terms in $\mathcal{O}$ divisible by $x_i$.
(b) For $i \in \{1, \ldots, n\}$ and $t \in \mathcal{O}_i$, the border basis element corresponding to $x_i\,t$ is $x_i\,t + t$.

*Proof:* To prove (a), it suffices to note that a term of the form $x_i^2\,t$ cannot be in $\mathcal{O}$, since $x_i^2\,t + x_i t$ is in $I$. To prove (b), we observe that the polynomial $x_i\,t + t$ is a multiple of $x_i^2 + x_i$ and hence in $I$. This implies the claim. □

Consequently, while computing a border basis of an ideal containing $F$, we can restrict everything to polynomials having only squarefree terms in their supports. For $t, t' \in \mathbb{S}^n$, we let $t{*}t'$ be the product of $t$ and $t'$ followed by reduction modulo $F$. For a polynomial $f \in P$, the normal form $\mathrm{NF}_F(f)$ is obtained by replying each term in $\mathrm{Supp}(f)$ by its squarefree part. For a list of polynomials $V$ we let $V^{(+)} = V \cup x_1{*}V \cup \cdots \cup x_n{*}V$. Putting these facts into the BBA, we get the following Boolean version.

*Algorithm 4.3:* **(The Boolean BBA)**
*Input:* generators $\{f_1, \ldots, f_s\}$ of $I$ and a term ordering $\sigma$
*Output:* the part of an $\mathcal{O}_\sigma(I)$-border basis of $I$ corresponding to $(\partial\mathcal{O}_\sigma(I))^{\mathrm{sf}}$

1: Let $U = \langle \mathrm{Supp}(\mathrm{NF}_F(f_1)) \cup \cdots \cup \mathrm{Supp}(\mathrm{NF}_F(f_s)) \rangle_{\mathrm{OI}}$.
2: Let $V$ be an LT-interreduced $K$-vector space basis of $\langle \mathrm{NF}_F(f_1), \ldots, \mathrm{NF}_F(f_s) \rangle_K$.
3: **repeat**
4:  **repeat**
5:  Compute a set of polynomials $W'$ such that $V \cup W'$ is an LT-interreduced basis of $\langle V^{(+)} \rangle_K$.
6:  **repeat**
7:  $W := \{w \in W' \mid \mathrm{LT}_\sigma(w) \in U\}$
8:  $U' := \langle \bigcup_{w \in W} \mathrm{Supp}(w) \setminus U \rangle_{\mathrm{OI}}$
9:  $U := U \cup U'$
10:  **until** $U' = \emptyset$
11:  $V := V \cup W$
12:  **until** $W = \emptyset$
13:  $\mathcal{O} := U \setminus \mathrm{LT}_\sigma(V)$
14:  Let $U_{\mathrm{old}} := U$ and $U := U^{(+)}$.
15: **until** $\partial\mathcal{O}^{\mathrm{sf}} \subset U_{\mathrm{old}}$
16: Apply `FinalReduction`$(V, \mathcal{O})$ and return the result.

Notice that, during the course of this algorithm, we always have $U \subset \mathbb{S}^n$ and $V \subset \langle \mathbb{S}^n \rangle$. Since $\mathbb{T}^n$ contains $\binom{n+d-1}{d}$ terms of degree $d$, while $\mathbb{S}^n$ contains $\binom{n}{d}$ terms of degree $d$, the universe and the resulting border basis will typically be much smaller for the Boolean BBA in comparison to the BBA.

## V. Improvements of the Boolean BBA

The main work of the Boolean BBA is clearly done in step (5) where an LT-interreduced basis of $V$ has to be extended to an LT-interreduced basis of $V^{(+)}$. The following observations allow us to speed up this step. As before, we let $\{f_1, \ldots, f_s\}$ be a set of polynomials in $P = \mathbb{F}_2[x_1, \ldots, x_n]$ which generates a 0-dimensional ideal $I$ containing the field ideal. We may assume that the polynomials $f_i$ are in normal form w.r.t. $F$, i.e., their support consists of squarefree terms. By $\mathbb{S}_d$ we denote the set of squarefree terms of degree $d$ in $P$, i.e. $\mathbb{S}_d = \{x_{i_1} \cdots x_{i_d} \mid 1 \le i_1 < \cdots < i_d \le n\}$.

*Proposition 5.1:* In the above setting, let $V_0$ be an LT-interreduced basis of $\langle f_1, \ldots, f_s \rangle$, and let $V_i$ be the set $V$ after the stable span loop of Algorithm 4.3 has performed $i$ iterations. Then we have

$$V_i \subseteq V_0 \cup \mathbb{S}_1 V_0 \cup \cdots \cup \mathbb{S}_i V_0.$$

*Proof:* This follows by induction on $i$ and by noting that, in view of Algorithm 4.3, we have $V_i = V_{i-1} \cup W \subseteq V_{i-1} \cup W'$ and $\langle W' \rangle_K \subseteq \langle V_{i-1}^{(+)} \rangle_K$. □

Notice that $\#\mathbb{S}_i = \binom{n}{i}$. Thus it is not advisable to use this proposition directly to find $V_i$. Instead, we can use this structural insight to improve the calculation of $W'$ in Step (5). Let $W_i$ and $W_i'$ be the values of the sets $W$ and $W'$ after the $i$-th iteration of the stable span loop (steps (4) through (12)). We write $W_i' = W_i \cup B_i$ and call $B_i$ the *unused polynomials* of round $i$.

*Proposition 5.2:* In the above setting, we have

$$V_i = V_{i-1} \cup W_{i-1} \subseteq V_{i-1} \cup B_{i-2} \cup W_{i-2}^{(+)} = V_{i-1}^{(+)}$$

for all $i \ge 2$. Hence $W_{i-1}'$ can be computed by reducing the elements of $B_{i-2} \cup W_{i-2}^{(+)}$ against $V_{i-1}$.
*Proof:* This follows from $V_{i-1}^{(+)} = (V_{i-2} \cup W_{i-2})^{(+)} = V_{i-2}^{(+)} \cup W_{i-2}^{(+)} = V_{i-2} \cup W_{i-2}' \cup W_{i-2}^{(+)} = V_{i-1} \cup B_{i-2} \cup W_{i-2}^{(+)}$ and the definition of $W_{i-1}'$. □

Since the purpose of the stable span loop is to approximate $I \cap \langle U \rangle_K$ by enlarging $V$ repeatedly until the process stabilizes, the number of newly found polynomials $W_i$ decreases rapidly, and thus also the size of $W_i^{(+)}$ decreases rapidly. By the proposition, this improves the computation of the basis extension from $V_i$ to $V_i^{(+)}$ substantially. The calculation of the sets $B_i$ can be inserted into the innermost repeat loop of Algorithm 4.3 as follows.

*Algorithm 5.3:* **(U-Extension Algorithm)**
*Input:* a set $W'$ and the current universe $U$
*Output:* sets $W$, $B$ such that $W' = W \cup B$ and an updated universe $U$

1: Let $W = \emptyset$ and $B := W'$.
2: **repeat**
3:  $A := \{w \in B \mid \mathrm{LT}_\sigma(w) \in U\}$
4:  Append $A$ to $W$ and remove it from $B$.
5:  $U' := \langle \bigcup_{w \in A} \mathrm{Supp}(w) \setminus U \rangle_{\mathrm{OI}}$
6:  $U := U \cup U'$
7: **until** $U' = \emptyset$
8: Return the pair $(W, B)$ and the order ideal $U$.

Now we can formulate the following improved version of the stable span loop in Algorithm 4.3.

*Algorithm 5.4:* **(Stable Span Algorithm)**
*Input:* an index $i \ge 2$, a set $V_{i-1}$, a pair of sets $(B_{i-2}, W_{i-1})$, and the current universe $U$

*Output:* a set $V_i$, a pair of sets $(B_{i-1}, W_{i-1})$, and an updated universe $U$

1: Compute an LT-interreduced basis extension $W'_{i-1}$ for $\langle V_{i-1}\rangle_K \subseteq \langle V_{i-1} \cup B_{i-2} \cup W^{(+)}_{i-2}\rangle_K$.
2: Apply the U-extension Algorithm 5.3 to $W'_{i-1}$ and $U$ and get a pair $(W_{i-1}, B_{i-1})$ and an updated universe $U$.
3: Let $V_i := V_{i-1} \cup W_{i-1}$.
4: Return $V_i$, the pair $(W_{i-1}, B_{i-1})$, and $U$.

The straightforward task to insert this Stable Span Algorithm correctly into Algorithm 4.3 is left to the reader.

Another reason why the sizes of the matrices involved in the basis extension step can increase dramatically is the enlargement $U := U^{(+)}$ in Step (14) of Algorithm 4.3. It was already suggested in [12] that we can use much smaller increases of the universe here if we make sure that occasionally, e.g., after a certain number of rounds, we include all of $\partial U$ to make sure the algorithm terminates. For instance, if we use the above Stable Span Algorithm 5.4, we can use $U := U \cup \langle \mathrm{LT}_\sigma(w) \mid w \in B_{i-1}\rangle_{\mathrm{OI}}$, where $B_{i-1}$ is the last value of the set of unused polynomials in the stable span iteration. Other strategies for enlarging the universe in Step (14) are possible and await further testing.

## VI. Combining the Boolean BBA and SAT Solvers

For large polynomial systems over $\mathbb{F}_2$, one approach to improve the performance of the Boolean BBA is to simultaneously run a SAT solver and to exchange intermediate results between the two processes. Let us briefly discuss the two directions for this exchange.

The input of the SAT solver are propositional logic clauses. When the Boolean BBA discovers a new simple polynomial $f \in I$ in the ideal, we can convert it to a logical formula $\mathcal{F}$ in CNF such that a tuple $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$ is a zero of $f$ if and only if it yields a satisfying assignment for the formula $\mathcal{F}$. Here "simple" means that we require a bound for the number of terms in support of the polynomial, because otherwise the conversion produces an exponentially large set of new clauses or many new indeterminates. Efficient methods for this conversion have been discussed for instance in [4] and [10].

For the purposes of optimizing the performance of the Boolean BBA, the reverse conversion method is the crucial one. Modern SAT solvers are able to produce large numbers of **conflict clauses** in a short time. A clause $\{L_1, \ldots, L_m\}$ consists of literals $L_i$, i.e., logical variables $X_i$ or their negations $\neg X_i$. The corresponding formula $\mathcal{F} = L_1 \vee L_2 \vee \cdots \vee L_m$ is satisfied for the assignment $(a_1, \ldots, a_n) \in \mathbb{F}_2^n$ if and only if the polynomial $f = \ell_1 \cdots \ell_m$ vanishes at the point $(a_1, \ldots, a_m)$, where $\ell_i = x_i + 1$ for $L_i = X_i$ and $\ell_i = x_i$ for $L_i = \neg X_i$. As the degree of $f$ equals the length of the clause, short conflict clauses are the most useful. Moreover, to get sparse polynomials, we prefer clauses with many negative literals.

Both solvers are of different nature. Modern SAT solvers are mainly based on DPLL (choosing the assignment for a literal and possibly backtracking), while the BBA is based on linear algebra elimination. Therefore their cooperation seems to be promising. Let us see how the BBA can profit from intermediate results of a SAT solver. Sometimes a SAT solver finds out the assignment for one indeterminate. For the BBA, this translates to a new polynomial in $I$ of the form $x_i + a_i$, where $a_i \in \mathbb{F}_2$. Normally, the BBA will not use this information very efficiently, as the following example shows.

*Example 6.1:* Suppose that we are computing the border basis of an ideal in $\mathbb{F}_2[x_1, x_2, x_3, x_4]$ and that the SAT solver provides us with the insight that $x_1 \in I$. Moreover, assume that we are currently trying to simplify $x_1x_2x_3x_4 + x_1x_2x_3 + x_3 = 0$. We have to wait for the result of two $V^{(+)}$ computations to eliminate $x_1x_2x_3$, and then one more to eliminate $x_1x_2x_3x_4$. However, a direct substitution of $x_1 \mapsto 0$ immediately shows $x_3 \in I$, and by substituting $x_3 \mapsto 0$, we can simpify other polynomials.

Therefore, whenever the SAT solver discovers useful polynomials in $I$, such as $x_i + a_i$ or $x_i + x_j$, it is better to perform the corresponding substitutions everywhere in the BBA and thus reduce the number of variables involved in the system. An experimental implementation of this SAT solver assisted Boolean BBA indicates promising speed-ups.

## VII. Implementation of Order Ideal Operations

To implement the Boolean BBA efficiently, we have to find suitable data structures for squarefree terms, order ideals and Boolean polynomials. We have implemented them in C++ as separate classes. In the following we look at the problem of implementing order ideals and the necessary order ideal operations efficiently.

*Remark 7.1: (Representation of Terms)*
A squarefree term $t = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ can be implemented in the *dense representation* via the bittuple $\alpha = (\alpha_1, \ldots, \alpha_n)$. The multiplication of terms $t * t'$, i.e., multiplication and subsequent reduction modulo $F$, can then be implemented via `OR` of the bittuples. Specifically, multiplication of $t$ by an indeterminate $x_i$ results in checking if the bit $\alpha_i$ is zero. If this conditions evaluates to true, the bit $\alpha_i$ is flipped. Similarly, a term $t$ with exponent vector $\alpha$ divides a term $t'$ with exponent vector $\alpha'$ if and only if $(\alpha \texttt{ AND } \alpha') = \alpha$ holds.

On the other hand, in practically relevant cases we may have a bound on the maximal degree of the terms that are used at some time during the computation. In this case we can use a *sparse representation:* it suffices to store the indices of the indeterminates appearing in the term. If the maximum degree is $d$ and we have $n = 2^k$ indeterminates, this representation requires $dk$ bits. In our setting, we usually have a large number of indeterminates, while the maximum degree is typically below 8. Thus we have $dk \ll 2^k = n$, and the sparse representation is more memory efficient.

*Example 7.2:* Let $t = x_1x_8$ be a term in $\mathbb{F}_2[x_1, \ldots, x_{10}]$. Then $t$ is represented as $(1,0,0,0,0,0,0,1,0,0)$ in the dense way and by $(1, 8)$ in the sparse way.

For the term ordering, we chose `DegLex`, because it is easily implementable and it reflects the fact that the Boolean BBA is a degree-by-degree algorithm.

To implement order ideals (such as the universe $U$) efficiently, we represent them by cogenerators which are defined as follows.

*Definition 7.3:* Let $\mathcal{O}$ be an order ideal in $\mathbb{T}^n$. A set of terms $\{t_1, \ldots, t_k\} \subseteq \mathcal{O}$ is called a set of **cogenerators** of $\mathcal{O}$ if every term in $\mathcal{O}$ divides one of the terms $t_1, \ldots, t_k$. A set of cogenerators $\{t_1, \ldots, t_k\}$ is called **minimal** if no term $t_i$ divides $t_j$ with $j \neq i$.

Clearly, an arbitrary set of cogenerators can be transformed to a minimal one by removing multiples iteratively. Every set of cogenerators contains a unique minimal one. We shall represent order ideals by their unique minimal set of cogenerators. Thus, after every order ideal operation, we minimalize the resulting set of cogenerators. Membership of a term in an order ideal can be checked by testing if it divides one of the cogenerators.

There are three steps in the Boolean BBA which require operations with order ideals: in Step (13) we have to find $U \setminus \mathrm{LT}_\sigma(V)$, in Step (14) we calculate $U^{(+)}$, and in Step (15) we check $\partial\mathcal{O}^{\mathrm{sf}} \subseteq U_{\mathrm{old}}$. Here the computation of $U^{(+)}$ is straightforward, since it suffices to take the union of the sets of cogenerators of $U$, $x_1U$, ..., $x_nU$, and to minimalize. The following proposition shows how we can calculate cogenerators of an order ideal minus a monomial ideal. This can then be applied to compute $U \setminus \mathrm{LT}_\sigma(V)$, because in [11], Prop. 21 it is shown that this set is an order ideal, and hence equal to $U \setminus \langle \mathrm{LT}_\sigma(V) \rangle$.

*Proposition 7.4:* Let $U$ be an order ideal in $\mathbb{T}^n$, let $C$ be a set of cogenerators of $U$, and let $t \in U$. Define the set $D = \{\frac{t'}{x_i} \mid i \in \{1, \ldots, n\},\ t' \in C,\ t \text{ divides } t',\text{ and } x_i \text{ divides } t'\}$. Then $(C \cup D) \setminus \langle t \rangle$ is a set of cogenerators of the order ideal $U \setminus \langle t \rangle$.

*Proof:* Let $u \in U \setminus \langle t \rangle$. Then $u$ divides a cogenerator $t' \in C$. If $t'$ is not a multiple of $t$, the claim is trivially true. If $t'$ is a multiple of $t$, then $u$ is a proper divisor of $t'$ and hence a divisor of one of the terms $\frac{t'}{x_i}$ in $D$. □

Notice that we can replace the condition that "$x_i$ divides $t'$" in the definition of $D$ by "$x_i$ divides $t$" if $U \subset \mathbb{S}^n$. The remaining task, namely checking whether $\partial\mathcal{O}^{\mathrm{sf}}$ is contained in $U_{\mathrm{old}}$ in Step (15), is dealt with by the following proposition.

*Proposition 7.5:* Let $U$ be an order ideal in $\mathbb{S}^n$, let $\mathcal{O}$ be an order ideal contained in $U$, let $C$ be the minimal set of cogenerators of $\mathcal{O}$, and let $D = x_1 * C \cup \cdots \cup x_n * C$. Then we have $\partial\mathcal{O}^{\mathrm{sf}} \subseteq U$ if and only if $D \subseteq U$.

*Proof:* Clearly, the terms in $D$ are contained in $\mathcal{O}$ or in $\partial\mathcal{O}^{\mathrm{sf}}$, and therefore in $U$. Conversely, assume that $D \subseteq U$. Every term $t \in \partial\mathcal{O}^{\mathrm{sf}}$ is of the form $t = x_i t'$ with $i \in \{1, \ldots, n\}$ and $t' \in \mathcal{O}$. Thus there exists a cogenerator $u \in C$ such that $u = t't''$ for some $t'' \in \mathbb{S}^n$. Now the claim follows from $t''t = x_i * u \in D \subseteq U$ and the fact that $U$ is an order ideal. □

Notice that it is more complicated to compute the border of $\mathcal{O}$ itself. We have to multiply the cogenerators of $\mathcal{O}$ by indeterminates and consider all factors of the resulting squarefree terms. Since we do not need this method here, we merely indicate it by an example.

*Example 7.6:* In $\mathbb{S}^3$ we consider $\mathcal{O} = \{1, x_1, x_3, x_1x_3\}$. By multiplying the cogenerator $x_1x_3$ with indeterminates, we get one squarefree border term, namely $x_1x_2x_3$. Now the divisors of this term provide the squarefree border $\partial\mathcal{O}^{\mathrm{sf}} = \{x_2, x_3, x_1x_2, x_2x_3, x_1x_2x_3\}$.

On the other hand, the test provided by the proposition is easy to implement. Terms in the set $D$ are produced by sequentially flipping zeros to ones in the exponents of the terms of $C$. Furthermore, in applications such as algebraic attacks or algebraic fault attacks in cryptography, we end up with a very small order ideal. Therefore the final computation of the border is much less expensive than the computation of the borders of the intermediate order ideals $\mathcal{O}$.

## VIII. C++ Implementation and Timings

After dealing with the combinatorial part of the Boolean BBA, we now turn to the implementation of polynomials and polynomial linear algebra. Clearly, this is the true core of the algorithm and needs to be optimized most. A Boolean polynomial can be implemented as a list of bittuples, each of which represents a term.

An important aspect of our implementation is that, together with the minimal set of cogenerators of the universe $U$, we keep track of the support of all polynomials that are currently used. The support and the cogenerators describes different aspects of the algorithm. E.g., during the computation of $V^{(+)}$, the support changes, since it is enlarged via multiplications by indeterminates, but the cogenerators remain untouched. On the other hand, in Steps (9) and (14), the universe is enlarged, i.e., the cogenerators are changed, but the support remains the same.

In the dense representation, the set of polynomials in $V$ can be represented by a coefficient matrix whose rows are lists of bittuples, and whose columns are labeled with the terms in the support. This is particularly useful, because the Boolean BBA is at its core a linear algebra algorithm for which the computation of the basis extension $W'$ is the most demanding task. This coefficient matrix has to be dynamic, because we add new rows and new columns during the computation of $V^{(+)}$. Moreover, when we add a new column, i.e., when we introduce a new term, the resulting columns must be ordered via the term ordering from the biggest to the smallest term.

Again we can choose between a sparse and a dense representation of the matrix. The basis extension $W'$ may be found via standard Gaussian elimination (GE). This is a special problem of computing a REF, because we need to know the permutation of the rows (if some swapping has occurred), and many pivot positions are known beforehand. In fact, we are really computing a REF extension.

It has turned out that GE without swapping rows, and optimized to minimize the writing all over the matrix, is the most suitable among different variants of GE. We start with a matrix $V_{old}$ which is already in REF. Then we append new rows. We reduce each of these rows by rows in $V_{old}$. At the end we get either a new pivot and we enlarge $V_{old}$, or we get a zero row which we cancel and we proceed to the next row. Therefore we read only from $V_{old}$ and write only in a single row at each step.

We prefer choosing pivots coming from rows in $V$. Otherwise, we could eliminate an row in $V$ by a row in $V^{(+)} \backslash V$, and thus obtain a bigger extension than necessary. If the matrix is in the dense representation, addition of polynomials is nothing but a `XOR` of two rows. On the other hand, when we use a sparse representation, i.e., when only the non-zero position in a row are remembered, addition of polynomials results in the symmetric difference of two lists.

In the following table we measure some execution times of our implementation of the Boolean BBA in C++ for both sparse and dense representations of the coefficient matrices. The timings were obtained on a personal laptop with a 2.6 GHz Intel(R) Core(TM) i7-5600U CPU and with 16 GB RAM. The program was compiled using the GNU GCC compiler with the `-O2` optimization flag. For benchmarking, we use systems of quadratic polynomial equations coming from the algebraic attacks at Small Scale AES (cf. [5]). The number of rounds is denoted by $r$, the number of rows of the state by $a$, the number of columns of the state by $b$ and the size of the word by $e$. To these parameters we add the number of variables and the number of equations.

| Small scale AES | | | | | | BBA (dense) | BBA (sparse) |
|---|---|---|---|---|---|---|---|
| # var | # eq | $r$ | $a$ | $b$ | $e$ | in seconds | in seconds |
| 20 | 36 | 1 | 1 | 1 | 4 | 0.04 | 0.01 |
| 36 | 60 | 1 | 1 | 2 | 4 | 1.60 | 0.14 |
| 36 | 68 | 2 | 1 | 1 | 4 | 1.17 | 0.27 |
| 40 | 72 | 1 | 2 | 1 | 4 | 12.76 | 0.21 |
| 52 | 100 | 3 | 1 | 1 | 4 | 27.15 | 7.03 |
| 64 | 112 | 2 | 1 | 2 | 4 | 240.23 | 35.03 |
| 68 | 132 | 4 | 1 | 1 | 4 | 953.44 | 17.94 |
| 72 | 120 | 1 | 2 | 2 | 4 | 422.61 | 4.56 |
| 72 | 136 | 2 | 2 | 1 | 4 | >1500 | 299.72 |
| 84 | 164 | 5 | 1 | 1 | 4 | >1500 | 148.92 |
| 100 | 196 | 6 | 1 | 1 | 4 | >1500 | 439.10 |
| 116 | 228 | 7 | 1 | 1 | 4 | >1500 | 1045.49 |

TABLE I
TIMINGS OF THE IMPLEMENTATION OF THE BOOLEAN BBA IN C++

These timings are not yet comparable to other algebraic solvers, many of which profit from optimizations developed over several decades. For instance, by introducing suitable variants of the Buchberger criteria, we expect to improve them significantly. Currently, they show the results of a first Boolean BBA implementation based only on standard C++ libraries (such as `std::bitset`, `boost::dynamic_bitset`, and `std::vector`), basic profiling and a cache-friendly design. Recall that 0/1 coefficients are stored in `std::bitset` and `boost::dynamic_bitset` only as one bit (plus some bytes in a header) in the memory, in contradiction to `bool`, which is usually stored in one byte, because it must be addressable. In fact, this size is platform dependent and can be larger.

From the table one can clearly see that sparse BBA outperforms dense BBA. Dense representation turns out be very effective when performing operations such as `XOR` of two rows. On the other hand, accessing a coefficient is expensive, because the bits are packed in the memory and therefore masking is required. Overall, sparse representation takes advantage of sparse input (this is the case for algebraic attacks), and in this context sparse linear algebra clearly outperforms dense techniques. Time performance does not depend only on the size of the input system, i.e., the number of polynomials and the number of indeterminates, but the shape of the system.

To end of this section, we mention possible optimizations of the linear algebra part. Firstly, the computation of the basis extension, i.e., the REF extension, should be implemented in a specialized algorithm rather than standard GE. Secondly, during the elimination some parts of the coefficient matrix are already very dense. Therefore a good strategy should divide the matrix into sparse/dense blocks and deal with them separately by using specialized libraries.

## IX. CONCLUSION

When it is applied to 0-dimensional polynomial systems over $\mathbb{F}_2$ having $\mathbb{F}_2$-rational solutions, the Border Basis Algorithm (BBA) can be simplified and sped up tremendously. We developed a Boolean BBA for such systems which reduces the universe, i.e., the order ideal of terms in which all computations take place, to a subset of the set of squarefree terms which is kept as small as possible. The linear algebra steps of the algorithm are optimized further by reusing some results of previous iterations. The necessary combinatorial operations on order ideals and monomial ideals are implemented in an improved way such that they do not contribute significantly to the running time anymore. A C++ implementation of the resulting algorithm is applied to a family of examples from cryptography, and it is used to compare the efficiency of dense and sparse representations of the coefficient matrices involved in the linear algebra steps. In particular, in combination with SAT solvers, the new algorithm offers good prospects for further improvements which allow tackling large polynomial systems of the described type.

## ACKNOWLEDGMENT

The authors would like to thank John Abbott and Anna Bigatti for helpful discussions about the implementation aspects of this paper and Jan Burchard for valuable input about SAT solvers. Further thanks go to the referees for their insightful and useful comments. This work was financially supported by the DFG Project “Algebraische Fehlerangriffe”.

## REFERENCES

[1] M. Brickenstein and A. Dreyer, POLYBORI: A framework for Gröbner-basis computations with Boolean polynomials, J. Symb. Comput. **44** (2009), 1326-1345.

[2] M. Brickenstein and A. Dreyer, Network-driven Boolean normal forms, in: B. Becker et al. (eds.), Verification over discrete-continuous boundaries, Dagstuhl Sem. Proc., Dagstuhl 2010.
[3] M. Brickenstein, A. Dreyer, G.-M. Greuel, M. Wedler, and O. Wienand, New developments in the theory of Gröbner bases and applications to formal verification, J. Prue Appl. Algebra **213** (2009), 1612-1635.
[4] G. Bard, N. Courtois, and C. Jefferson, Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers, Cryptology ePrint Archive, Report 2007/024, 2007.
[5] C. Cid, S, Murphy, and M.J.B. Robshaw, Small scale variants of the AES, in: H. Gilbert and H. Handschuh (eds.), Fast Software Encryption, LNCS **3557**, Springer Verlag, Heidelberg 2005, pp. 145–162.
[6] M. Clegg, J. Edmonds, and R. Impagliazzo, Using the Gröbner basis algorithm to find proofs of unsatisfiability, in: Proc. STOC'96, ACM, New York 1996, pp. 174-183.
[7] V. Gerdt and M. Zinin, A Pommaret division algorithm for computing Gröbner bases in Boolean rings, in: Proc. Conf. ISSAC 2008 (Linz/Hagenberg), ACM Press, New York 2008, pp. 95-102.
[8] V. Gerdt, M. Zinin, and Y. Blinkov, On computation of Boolean involutive bases, Prog. and Comp. Software **36** (2010), 117-123.
[9] H. Huang, W. Bao, Middle-solving F4 to compute Gröbner bases for cryptanalysis over GF(2), preprint 2014, available at `http://arxiv.org/abs/1310.2332`
[10] P. Jovanovic and M. Kreuzer, Algebraic attacks using SAT solvers, Groups - Complexity - Cryptology **2** (2010), 247-259.
[11] A. Kehrein and M. Kreuzer, Computing border bases, J. Pure Appl. Algebra **205** (2006), 279-295.
[12] M. Kreuzer, Tutorial 5: Variations on BBA, COCOA 2007: Int. School on Computer Alg., Hagenberg 2007, available at `http://cocoa.dima.unige.it/conference/cocoa2007/notes/KR_tutorial_5.pdf`
[13] M. Kreuzer and L. Robbiano, Computational Commutative Algebra 1, Springer Verlag, Heidelberg 2000.
[14] M. Kreuzer and L. Robbiano, Computational Commutative Algebra 2, Springer Verlag, Heidelberg 2005.
[15] B. Mourrain, A new criterion for normal form algorithms, in: M. Fossorier *et al.* (eds.), Proc. Conf. AAECC-13, Honolulu 1999, LNCS **1719**, Springer, Heidelberg 1999, pp. 440-443.
[16] T. H. Nguyen, Combinations of Boolean Gröbner bases and SAT solvers, dissertation, TU Kaiserslautern, 2014.
[17] M. Noro et al., Risa/Asir – A computer algebra system, available at `http://www.math.kobe-u.ac.jp/Asir/asir.html`
[18] Y. Sato, S. Inoue, A. Suzuki, K. Nabeshima, and K. Sakai, Boolean Gröbner bases, J. Symb. Comput. **46** (2011), 622-632.
[19] C. Zengler and W. Küchlin, Extending clause learning of SAT solvers with Boolean Gröbner bases, in: Computer Algebra in Scientific Computing, Proc. CASC'10, Springer Verlag, Heidelberg 2010, pp. 293-302.