Integrating Algebraic and SAT Solvers

Jan Horáček[1(✉)], Jan Burchard[2], Bernd Becker[2], and Martin Kreuzer[1]

[1] Faculty of Informatics and Mathematics, University of Passau, 94030 Passau, Germany
{Jan.Horacek,Martin.Kreuzer}@uni-passau.de

[2] Computer Architecture Group, Albert-Ludwigs-University Freiburg, 79110 Freiburg, Germany
{burchard,becker}@informatik.uni-freiburg.de

Abstract. For solving systems of Boolean polynomials whose zeros are known to be contained in $\mathbb{F}_2^n$, algebraic solvers such as the Boolean Border Basis Algorithm (BBBA) and SAT solvers use very different and possibly complementary methods to create new information. Based on suitable implementations of these solvers and conversion methods from Boolean polynomials to SAT clauses and back, we describe an automatic framework integrating the two solving techniques and exchanging newly found information between them. Using examples derived from cryptographic attacks, we present some initial experiments indicating the efficiency of this combination.

Keywords: Boolean polynomial · Border Basis Algorithm
SAT solving · Cryptographic attack

1 Introduction

Cryptographic attacks frequently require the solution of polynomial systems defined over the field $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$ for which it is known that the desired solution consists of one or more points in $\mathbb{F}_2^n$. In this case we may add the *field equations* $x_i^2 + x_i = 0$ to the given system, where $i = 1, \ldots, n$, to express that fact that we are looking for solutions $(a_1, \ldots, a_n)$ such that $a_i \in \mathbb{F}_2$. Equivalently, we may consider the system as a system of Boolean polynomials, i.e., a system defined by elements in

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

Several methods have been developed to deal with this task.

Algebraic solvers consider the ideal I in $\mathbb{F}_2[x_1, \ldots, x_n]$ generated by the given polynomials and the field equations and perform operations such as polynomial addition and multiplication in order to find simple polynomials in I which allow us to read off the solutions of the system. Examples for such methods are Boolean Gröbner basis computations (see [3]) and the Boolean Border Basis Algorithm (BBBA) (see [11,12]). After converting the polynomial system to a set of propositional logic clauses (see [1,10,13]), one can also use a *SAT solver* to determine a satisfying assignment for the logical variables, which in turn corresponds to a solution of the Boolean polynomial system (see for instance [5]). SAT solvers use logical reasoning (such as CDCL and DPLL) to eliminate large sets of assignments which do not satisfy the given set of clauses.

In this paper we combine both algebraic solvers and SAT solvers by running two processes in parallel and interchanging information between them. More specifically, we describe an implementation of an automatic framework which executes the Boolean Border Basis Algorithm (see [11]) and the SAT solver `antom` (see [16]) concurrently, transforms newly found "interesting" polynomials resp. SAT clauses using suitable conversion methods (see [10]), and introduces this new information into the other process. Up to now, Gröbner bases computations have been used predominantly to speed up certain stages of a SAT solver computation (see [6,7,17]). Hence this paper may be considered as a first step towards a systematic combination of algebraic and logical reasoning in order to solve Boolean polynomial systems.

Let us describe its contents in more detail. In Sect. 2 we recall some efficient algorithms for performing certain operations with order ideals of terms used afterwards, and in Sect. 3 we remind the reader of Boolean polynomials and spell out an explicit method for their linear reduction and interreduction. In Sect. 4 we present a version of the Boolean Border Basis Algorithm (originally presented in [11]) which is closer to the actual implementation and which allows us to introduce the integration with the SAT solver explicitly at suitable points of the calculation.

Section 5 contains the description of the integration of this version of the BBBA with a SAT solver. In particular, we have to select which polynomials and which clauses we send to the respective other solver, keeping the amount of transmitted data under control and providing that information which has the best chances to improve the overall solving speed. Then, in Sect. 6, we describe the design of the actual communication process between the two solvers. This entails finding suitable entry points for the new information as well as a queuing process for these data until a suitable point in time for the insertion is reached. Section 7 contains some observations about the necessary modifications to a standard SAT solver such as `antom` (cf. [16]).

Finally, in Sect. 8 we report some preliminary results about speed-ups of the overall solving time we could achieve. Both for a manual insertion of new information as well as for the automatic communication process described above, we found cases with substantial improvements of the total solving time. However, sometimes the combination of the two processes was slower, and the effects depend strongly on the chosen selection strategies for the transmitted information. Thus further experimentation using the new tools is needed to optimize the synergies which we can achieve.

Unless explicitly stated otherwise, we use the basic definitions and results in [11,15].

2 Algorithms for Basic Operations with Order Ideals

The set of **squarefree terms** in the indeterminates $x_1, \ldots, x_n$ is denoted by $\mathbb{S}^n$. We order terms in $\mathbb{S}^n$ by a *degree compatible term ordering* σ. An **order ideal** is a factor-closed set of terms. Let $\mathcal{O}$ be an order ideal in $\mathbb{S}^n$. A set of terms $C = \{t_1, \ldots, t_k\} \subseteq \mathcal{O}$ is called a set of **cogenerators** of $\mathcal{O}$ (or we say that C cogenerates $\mathcal{O}$) 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 for $j \neq i$. Thus order ideals are represented by their (unique) minimal set of cogenerators. For a set of terms $C \subseteq \mathbb{S}^n$, we denote by $\langle C \rangle_{\mathrm{OI}}$ the order ideal cogenerated by C.

In this section, three different non-trivial operations with order ideals used in the BBBA are discussed briefly. The order ideal membership problem is decided by Algorithm 1. Its proof of correctness follows immediately from the definition of cogenerators.

Algorithm 1. (Order Ideal Membership Test)
Input: Cogenerators C of an order ideal $\mathcal{O}$ in $\mathbb{S}^n$, $t \in \mathbb{S}^n$.
Output: `True` if $t \in \mathcal{O}$, `False` otherwise.

```
1: $a$ := False
2: foreach $c$ in $C$ do
3:    if $t$ divides $c$ then
4:       $a$ := True
5:    end if
6: end foreach
7: return $a$
```

The **squarefree border** of an order ideal $\mathcal{O}$ in $\mathbb{S}^n$ is defined as $\partial\mathcal{O}^{\,\mathrm{sf}} = \big((\bigcup_{i=1}^{n} x_i\mathcal{O}) \setminus \mathcal{O}\big) \cap \mathbb{S}^n$. Next we present Algorithm 3 (and its subroutine Algorithm 2) for computing the minimal set of cogenerators of an order ideal minus a monomial ideal.

Algorithm 4 decides if the squarefree border of one order ideal is contained in some other order ideal. These algorithms are variants of Propositions 7.4 and 7.5 in [11]. Step 11 of Algorithm 2 and Step 12 of Algorithm 4 can be computed by removing terms that are divisible by others.

3 Linear Interreduction for Boolean Polynomials

In the following we let $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$ be the binary field and $\mathbb{F}_2[x_1, \ldots, x_n]$ a polynomial ring over $\mathbb{F}_2$. The ideal $F = \langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$ is called the **field ideal**. The ring $\mathbb{B}_n = \mathbb{F}_2[x_1, \ldots, x_n]/\langle F \rangle$ is called the **ring of Boolean polynomials** in the indeterminates $x_1, \ldots, x_n$. We assume that its elements are represented by polynomials whose support consists only of squarefree terms, i.e. all operations with polynomials are performed modulo the field ideal.

A set of Boolean polynomials $G \subseteq \mathbb{B}_n$ is called linearly LT_σ**-interreduced** if $\mathrm{LT}_\sigma(g) \neq \mathrm{LT}_\sigma(g')$ for all $g, g' \in G$ with $g \neq g'$. Given an arbitrary set of Boolean polynomials $G \subseteq \mathbb{B}_n$, we can linearly LT_σ-interreduce G via (sparse) Gaußian elimination on the coefficient matrix of G. (Here the columns have to be ordered w.r.t. σ.) For a better understanding of linear LT_σ-interreduction,

Algorithm 2. (Order Ideal Minus a Monomial Ideal Generated by a Term)
Input: $t \in \mathbb{S}^n$, the minimal set of cogenerators C of an order ideal $\mathcal{O}$ in $\mathbb{S}^n$.
Output: The minimal set of cogenerators of the order ideal $\mathcal{O} \setminus \langle t \rangle$.

```
1: $D := \emptyset$
2: foreach $c$ in $C$ do
3:    if $t$ divides $c$ then
4:       for $i = 1$ to $n$ do
5:          if $x_i$ divides $t$ then
6:             $D := D \cup \{\frac{c}{x_i}\}$
7:          end if
8:       end for
9:    end if
10: end foreach
11: Let $A$ be the set of the minimal elements in $(C \cup D) \setminus \langle t \rangle$ w.r.t. division.
12: return $A$
```

Algorithm 3. (Order Ideal Minus a Monomial Ideal)
Input: The minimal set of cogenerators C' of an order ideal U in $\mathbb{S}^n$, a set of squarefree terms T.
Output: The minimal set of cogenerators C of the order ideal $U \setminus \langle T \rangle$.
Requires: Algorithm 2.

```
1: $C := C'$
2: foreach $t$ in $T$ do
3:    if $t \in \langle C' \rangle_{\mathrm{OI}}$ then
4:       $C :=$Algorithm 2$(t, C)$
5:    end if
6: end foreach
7: return $C$
```

we formulate the following definitions which are analogous to the rewriting rules in the Gröbner basis theory (see [15, Definition 2.2.1]).

Definition 1. *Let $V \subseteq \mathbb{B}_n$, and let $b, r, b' \in \mathbb{B}_n$.*

(a) We say that b **linearly** LT_σ**-reduces to** b' **in one step** *using r if $\mathrm{LT}_\sigma(b) = \mathrm{LT}_\sigma(r)$ and $b' = a + r$. We write $b \xrightarrow{r} b'$.*
(b) We say that b **linearly** LT_σ**-reduces to** b' *using V if there exist $v_i \in V$ for $i = 1, \ldots, k$ and $b_1, \ldots, b_{k-1} \in \mathbb{B}_n$ such that $b \xrightarrow{v_1} b_1 \xrightarrow{v_2} \ldots \xrightarrow{v_{k-1}} b_{k-1} \xrightarrow{v_k} b'$. We write $b \xrightarrow{V} b'$.*
(c) A polynomial b with the property that there is no $r \in V$ such that $b \xrightarrow{r} b'$ for some $b' \in \mathbb{B}_n$ is called **linearly** LT_σ**-irreducible** *with respect to V.*

Obviously, we have $b \xrightarrow{b} 0$ for any polynomial b. The following example shows us that the result of a sequence of linear LT_σ-reductions is not uniquely determined in general.

Algorithm 4. (Checking the Border)
Input: Cogenerators C' of an order ideal U in $\mathbb{S}^n$, cogenerators D of an order ideal $\mathcal{O}$ in $\mathbb{S}^n$.
Output: `True` if $\partial\mathcal{O}^{\,\mathrm{sf}} \subseteq U$, `False` otherwise; a set of squarefree terms C such that $\langle C\rangle_{\mathrm{OI}} = \langle C' \cup \partial\mathcal{O}^{\,\mathrm{sf}}\rangle_{\mathrm{OI}}$.

```
1: $a :=$ True
2: $B := \emptyset$
3: foreach $d$ in $D$ do
4:    for $i = 1$ to $n$ do
5:       $d' := x_i d$
6:       if $d' \in \mathbb{S}^n$ and $d' \notin \langle C'\rangle_{\mathrm{OI}}$ then
7:          $B := B \cup \{d'\}$
8:          $a :=$ False
9:       end if
10:   end for
11: end foreach
12: Let $C$ be the set of the minimal elements in $(C' \cup B)$ w.r.t. division.
13: return $(a, C)$
```

Example 1. Let $V = \{x_1x_2 + 1, x_1x_2, x_1 + 1\} \subseteq \mathbb{B}_2$. Then $x_1x_2 + x_1 \xrightarrow{x_1x_2+1} x_1 + 1 \xrightarrow{x_1+1} 0$, and thus $x_1x_2 + x_1 \xrightarrow{V} 0$. On the other hand, $x_1x_2 + x_1 \xrightarrow{x_1x_2} x_1 \xrightarrow{x_1+1} 1$, and thus $x_1x_2 + x_1 \xrightarrow{V} 1$.

If we would like to have unique linear LT_σ-reducers (and hence unique linear LT_σ-reductions), the set V has to be linearly LT_σ-interreduced.

Proposition 1. *Let V be a linearly LT_σ-interreduced set of Boolean polynomials. Let $b, b' \in \mathbb{B}_n$ such that $b \xrightarrow{V} b'$. Then the polynomial b' is uniquely determined.*

Proof. There exists exactly one element $v_1 \in V$ such that $\mathrm{LT}_\sigma(b) = \mathrm{LT}_\sigma(v_1)$, because the leading terms of the elements of V are pairwise distinct. Let $b_1 = b - v_1$. We have $b \xrightarrow{v_1} b_1$. There exists at most one element $v_2 \in V$ such that $\mathrm{LT}_\sigma(b_1) = \mathrm{LT}_\sigma(v_2)$. If there is no such v_2, the element b_1 is the unique linear LT_σ-reduction of b. Otherwise, we continue with $b_2 = b_1 - v_2$ in the same way, and the result follows by induction. □

The following proposition gives us another useful property of a linearly LT_σ-interreduced set of Boolean polynomials.

Proposition 2. *Let V be a linearly LT_σ-interreduced set of Boolean polynomials, and let $b, r \in \mathbb{B}_n$. Then we have $b \xrightarrow{V} 0$ if and only if $b \in \langle V\rangle_{\mathbb{F}_2}$.*

Proof. First we prove "⇒". By definition, there exist $v_1, \dots, v_k \in V$ and $b_i \in \mathbb{B}_n$ for $i = 1, \dots, k-1$ such that $b \xrightarrow{v_1} b_1 \xrightarrow{v_2} \dots \xrightarrow{v_{k-1}} b_{k-1} \xrightarrow{v_k} 0$. Hence we get $b = v_1 + \cdots + v_k$ in $\mathbb{B}_n$, and henceforth $b \in \langle V\rangle_{\mathbb{F}_2}$.

Conversely, let $b = v_1 + \cdots + v_k$ for some pairwise distinct elements $v_1, \ldots, v_k$ in V. Because V is linearly LT_σ-interreduced, there exists a unique index $i_1 \in \mathbb{N}$ with $1 \leq i_1 \leq k$ such that $\mathrm{LT}_\sigma(v_{i_1}) = \mathrm{LT}_\sigma(b)$. Hence $b \xrightarrow{v_{i_1}} (b - v_{i_1})$. There exists a unique $i_2 \in \mathbb{N}$ with $1 \leq i_2 \leq k$ such that $\mathrm{LT}_\sigma(v_{i_2}) = \mathrm{LT}_\sigma(b - v_{i_1})$. By induction we create a zero linear LT_σ-reduction chain starting from b and having linear LT_σ-reducers $v_{i_1}, \ldots, v_{i_k} \in V$. □

Example 2. Let $V = \{x_1x_2 + x_1, x_1x_2 + x_2\} \subseteq \mathbb{B}_2$. We can see that $x_1 + x_2 \in \langle V \rangle_{\mathbb{F}_2}$, but $x_1 + x_2$ is linearly LT_σ-irreducible with respect to V.

We are now ready to introduce and analyze Algorithm 5 for computing successive extensions of linearly LT_σ-interreduced sets. As a pivoting strategy for the reduction process, we first consider Boolean polynomials of smallest degree and among them the ones having smallest support. Algorithm 5 will be applied in Algorithm 8 in the next section.

Definition 2. *Let $f, g \in \mathbb{B}_n$. We write $f \prec g$ if and only if* $\deg(f) < \deg(g)$*, or* $\deg(f) = \deg(g)$ *and* $\#\mathrm{Supp}(f) < \#\mathrm{Supp}(g)$.

Algorithm 5. (Extensions of Linearly LT_σ-Interreduced Tuples)
Input: A non-zero Boolean polynomial b', a linearly LT_σ-interreduced set of Boolean polynomials V', and a degree compatible term ordering σ.
Output: A set $V \subseteq \mathbb{B}_n$ such that V is linearly LT_σ-interreduced and $\langle V \rangle_{\mathbb{F}_2} = \langle V' \cup \{b'\} \rangle_{\mathbb{F}_2}$.

```
1: $b := b', V := V'$
2: while there exists $r \in V$ with $\mathrm{LT}_\sigma(r) = \mathrm{LT}_\sigma(b)$ do
3:     $b := b + r$
4: end while
5: if $b \neq 0$ then
6:     $V := V \cup \{b\}$
7: end if
8: return $V$
```

Proposition 3. *Algorithm 5 returns a linearly* LT_σ*-interreduced list V such that $\langle V \rangle_{\mathbb{F}_2} = \langle V' \cup \{b'\} \rangle_{\mathbb{F}_2}$ holds.*

Proof. In Step 2 we search for a unique polynomial in V' which has the same leading term as b. If such a polynomial does not exist, the polynomial b is appended to V in Step 6.

The linear LT_σ-reduction chain is constructed in Steps 2–4. If $b' \xrightarrow{V'} 0$, then $b' \in \langle V' \rangle_{\mathbb{F}_2} \subseteq \langle V \rangle_{\mathbb{F}_2}$ by Proposition 2. If we have $b' \xrightarrow{V'} b \neq 0$, then we have $b' \in V$ by Step 6. □

4 The Boolean Border Basis Algorithm

To start with, we recall the definition of a Boolean $\mathcal{O}$-border basis (see [11]).

Definition 3. *Let $P = \mathbb{F}_2[x_1, \ldots, x_n]$, let $\mathcal{O} = \{t_1, \ldots, t_\mu\}$ be an order ideal in $\mathbb{S}^n$, and let $\partial\mathcal{O}^{\,\mathrm{sf}} = \{b_1, \ldots, b_\nu\}$ be its squarefree border. Let $I \subseteq \mathbb{F}_2[x_1, \ldots, x_n]$ be an ideal containing the field ideal $F = \langle x_1^2 + x_1, \ldots, x_n^2 + x_n\rangle$.*

(a) A set of polynomials $G = \{g_1, \ldots, g_\nu\}$ is called a **Boolean $\mathcal{O}$-border prebasis** *if $g_j = b_j + \sum_{i=1}^{\mu} c_{ij}\, t_i$ with $c_{1j}, \ldots, c_{\mu j} \in \mathbb{F}_2$ for $j = 1, \ldots, \nu$.*
(b) A Boolean $\mathcal{O}$-border prebasis $G \subset I$ is called a **Boolean $\mathcal{O}$-border basis** *of I if the residue classes $\overline{\mathcal{O}} = \{\bar{t}_1, \ldots, \bar{t}_\mu\}$ in P/I form an $\mathbb{F}_2$-basis of P/I.*

Let us motivate the idea of the BBBA using the problem of finding the $\mathbb{F}_2$-rational solutions of a Boolean system $f_1 = \cdots = f_s = 0$. Let $V = \{f_1, \ldots, f_s\}$. Define the ideal $I = \langle f_1, \ldots, f_s\rangle \subseteq \mathbb{B}_n$. Suppose that the system has a unique $\mathbb{F}_2$-rational solution. (For instance, this is common in the scenario of algebraic attacks.) We are looking for a set of linear polynomials $G \subseteq I$ such that G is a linearly LT_σ-interreduced basis of $\langle G\rangle_{\mathbb{F}_2}$ and $\#\mathrm{Supp}(G) = \#G$. Hence the goal is to create new linearly independent linear polynomials in I and to keep the support of polynomials in the system as small as possible at the same time.

Given a set of Boolean polynomials $V = \{f_1, \ldots, f_s\}$, the BBBA generates new polynomials by forming and linearly LT_σ-interreducing $V^{(+)} = V \cup x_1 V \cup \cdots \cup x_n V$. Note that the multiplications are done in $\mathbb{B}_n$. Every iteration of $V^{(+)}$ is then followed by linear LT_σ-interreduction. One could repeat these two operations in order to obtain the desired basis. On the other hand, this approach clearly leads to an exponentially large amount of work since all polynomials in V are multiplied by n indeterminates.

Thus the operation $V^{(+)}$ in the BBBA is restricted by the order ideal U. The order ideal U is called the **universe** and U is initially cogeneratored by $\bigcup_i \mathrm{Supp}(f_i)$. The $V^{(+)}$ operation is restricted to polynomials that have their support contained in the universe. In this way, the growth of V and the support of the polynomials in V is lower. The universe is extended by the support of polynomials that have leading terms contained in U. This extension of the universe is described in Algorithm 6 which is used in Step 12 of Algorithm 8. The proof of correctness of Algorithm 6 is easy and left to the reader.

The successive computation of $V^{(+)}$ tends to repeat the consideration of multiples of polynomials that have been already multiplied by all indeterminates. To avoid this overhead, we introduce the following notion.

Definition 4. *A Boolean polynomial $f \in \mathbb{B}_n$ is said to be* **covered** *in a linearly LT_σ-interreduced set $V \subseteq \mathbb{B}_n$ if $x_i f \xrightarrow{V} 0$ for all $i \in \{1, \ldots, n\}$.*

Covered polynomials should be avoided because they do not introduce any new leading terms. The definition is equivalent to the condition $x_i f \in \langle V\rangle_{\mathbb{F}_2}$ for $i = 1, \ldots, n$ by Proposition 2. Checking the latter condition is quite expensive for large sets V. When we repeat the $V^{(+)}$ operation and linear LT_σ-interreduction, we remember the polynomials that have been worked on as in the following example.

Algorithm 6. (Extension of the Universe)
Input: Cogenerators C' of an order ideal U in $\mathbb{S}^n$, a linearly LT_σ-interreduced set of Boolean polynomials V.
Output: A set of cogenerators $C \supseteq C'$ such that $\mathrm{LT}_\sigma(f) \in \langle C \rangle_{OI}$ for $f \in V$ implies that f is contained in $\langle C \rangle_{OI}$.

```
1: $C := C'$
2: repeat
3:    $D := C$
4:    foreach $f$ in $V$ do
5:       if $\mathrm{LT}_\sigma(f) \in \langle C \rangle_{\mathrm{OI}}$ and $f$ is not contained in $\langle C \rangle_{OI}$ then
6:          Let $A$ be the set of the minimal cogenerators of $\langle C \cup \mathrm{Supp}(f) \rangle_{OI}$.
7:          $C := A$
8:       end if
9:    end foreach
10: until $\#D = \#C$
11: return $C$
```

Example 3. Let $f = x_1x_2 + 1 \in \mathbb{B}_2$ and $V' = \{f\} \subseteq \mathbb{B}_2$. Let us compute $V'^{(+)}$ iteratively with successive linear LT_σ-interreduction. We compute $x_1 f = x_1x_2 + x_1 \xrightarrow{f} x_1 + 1$ and $x_2 f = x_1x_2 + x_2 \xrightarrow{f} x_2 + 1$. We get $V = \{x_1x_2 + 1, x_1 + 1, x_2 + 1\}$. Then f is covered in V, and therefore multiplication of $x_1x_2 + 1$ by indeterminates does not yield new linearly independent polynomials during the computation of $V^{(+)}$. Thus we remember that the polynomial f is covered in V.

Algorithm 7 computes $\{b\}^{(+)}$ for b a Boolean polynomial and immediately linearly LT_σ-reduces the result against the known polynomials. To keep the pseudo-code simple, the covered polynomials that are easily discoverable are stored in the set $M \subseteq V$. The proof of correctness of Algorithm 7 follows directly from Proposition 3.

Now we describe a restructured version of the BBBA in Algorithm 8. Its subroutine `FinalReduction` refers to the algorithm in [14, Proposition 17] whose purpose is to extract the desired border basis from $\langle V \rangle_{\mathbb{F}_2}$. Notice that this algorithm can be easily modified to output only the polynomials having squarefree border terms.

Proposition 4. *In the setting of Algorithm 8, Algorithm 8 outputs the Boolean $\mathcal{O}_\sigma(I)$-border basis of I.*

Proof. It is sufficient to prove that Algorithm 8 is equivalent to Algorithm 4.3 in [11]. The set V_a denotes the set of all polynomials in V which are contained in the current universe $U = \langle C \rangle_{\mathrm{OI}}$. Note that V may contain polynomials which are not in $\langle C \rangle_{\mathrm{OI}}$. Thus the set V in Algorithm 4.3 in [11], corresponds to V_a.

The only difference in the initialization (apart from defining the new set M) occur in Steps 3–5. They are equivalent to linear LT_σ-interreducing of the initial generators V.

Algorithm 7. (Plus and Reduce)
Input: A non-zero Boolean polynomial b, a linearly LT_σ-interreduced set of Boolean polynomials V', a degree compatible term ordering σ, cogenerators C of an order ideal U in $\mathbb{S}^n$, and a set $M' \subseteq V$ of covered polynomials in V.
Output: A linearly LT_σ-interreduced set V such that $\langle V' \cup \{x_1 b, \ldots, x_n b\}\rangle_{\mathbb{F}_2} = \langle V\rangle_{\mathbb{F}_2}$ if b is contained in $\langle C\rangle_{\mathrm{OI}}$, $V = V'$ otherwise, and a set of covered polynomials M.
Requires: Algorithm 5.

```
1: $V := V'$, $M := M'$
2: if $b$ is contained in $\langle C\rangle_{OI}$ and $b \notin M$ then
3:    for $i = 1$ to $n$ do
4:       $b' := x_i b$
5:       Update $V$ by calling Algorithm 5$(b', V, \sigma)$.
6:    end for
7:    $M := M \cup \{b\}$
8: end if
9: return $(V, M)$
```

Now we would like to show that Steps 7–13 computes the $\langle C\rangle_{\mathrm{OI}}$-stabilization of V_a, i.e. that $\langle V_a\rangle_{\mathbb{F}_2} = \langle V_a^{(+)}\rangle_{\mathbb{F}_2} \cap \langle U\rangle_{\mathbb{F}_2}$ holds in Step 14. The inclusion "$\subseteq$" is trivial. Let us look at the other inclusion. The set M contains polynomials in V such that $M^{(+)} \subseteq \langle V\rangle_{\mathbb{F}_2}$, so elements in M can be omitted in Algorithm 7.

Let $U = \langle C\rangle_{\mathrm{OI}}$ and $v \in \langle V_a^{(+)}\rangle_{\mathbb{F}_2} \cap \langle U\rangle_{\mathbb{F}_2}$ in Step 14. We know that $v \xrightarrow{V} 0$ because $\langle V_a^{(+)}\rangle_{\mathbb{F}_2} \subseteq \langle V\rangle_{\mathbb{F}_2}$ after Step 11. This means that $v \in \langle V\rangle_{\mathbb{F}_2}$ by Proposition 2 because V is linearly LT_σ-interreduced. We would like to show that $v \xrightarrow{V_a} 0$, which is equivalent to $v \in \langle V_a\rangle_{\mathbb{F}_2}$. Let $v = v_1 + \cdots + v_k$, where $\{v_1, \ldots, v_k\} \subseteq V$ is a linearly LT_σ-interreduced set. Then $\mathrm{LT}_\sigma(v) = \mathrm{LT}_\sigma(v_i)$ for some $1 \leq i \leq k$. Since $\mathrm{LT}_\sigma(v_i) = \mathrm{LT}_\sigma(v) \in U$, we get $v_i \in \langle U\rangle_{\mathbb{F}_2}$, i.e. $v_i \in V_a$ after Step 12. We continue with the polynomial $v - v_i$ and we get that $\{v_1, \ldots, v_k\} \subseteq V_a$ by induction.

The loop in Steps 9–11 enlarges V by elements in $\langle V_a^{(+)}\rangle_{\mathbb{F}_2}$ such that updated V is linearly LT_σ-interreduced. (This is equivalent to Step 5 of Algorithm 4.3 in [11].) Step 12 enlarges the universe in the same way as Steps 6–10 of Algorithm 4.3 in [11] do.

The rest (i.e., Steps 15–17) continues in the same way as Steps 13–16 of Algorithm 4.3 in [11]. □

5 The Integration of the BBBA with a SAT Solver

Many search problems can be encoded as systems of Boolean polynomials or SAT-instances. Inputs of SAT-solvers are usually in CNF (Conjunctive Normal Form), i.e. a conjunction of disjunctions of literals, where a literal is either a logical variable or its negation.

Algorithm 8. The BBBA (Restructured Version)
Input: A set of polynomials $V = \{f_1, \ldots, f_s\} \subseteq \mathbb{B}_n$ such that $V \cup F$ generates a 0-dimensional ideal I and a degree compatible term ordering σ.
Output: The polynomials of the Boolean $\mathcal{O}_\sigma(I)$-border basis of I where $\mathcal{O}_\sigma(I) = \mathbb{S}^n \setminus \mathrm{LT}_\sigma(I)$.
Requires: Algorithms 3, 4, 5, 6, 7, `FinalReduction`.

```
1: $V := \emptyset, M := \emptyset$
2: Let $C$ be a set of the minimal cogenerators of the order ideal $\langle \bigcup_{i=1}^{s} \mathrm{Supp}(f_i) \rangle_{\mathrm{OI}}$.
3: for $i = 1$ to $s$ do
4:     Update $V$ by calling Algorithm 5$(f_i, V, \sigma)$.
5: end for
6: repeat
7:     repeat
8:         $V' := V$
9:         foreach $f$ chosen in the increasing order according to "$\prec$" in $V'$ do
10:            Update $(V, M)$ by calling Algorithm 7$(f, V, \sigma, C, M)$.
11:        end foreach
12:        $C :=$Algorithm 6$(C, V)$.
13:    until $\#V = \#V'$
14:    $D :=$ Algorithm 3$(C, \mathrm{LT}_\sigma(V))$.
15:    Update $(a, C)$ by calling Algorithm 4$(C, D)$.
16: until $a =$ True
17: Apply FinalReduction$(V, \langle D \rangle_{\mathrm{OI}})$ and return the result.
```

One can solve the same problem with the BBBA or a SAT solver individually. There exist conversion methods that transform a Boolean system to a CNF formula (and vice versa) such that the $\mathbb{F}_2$-rational zeros of the system correspond to the satisfying assignments of the logical formula. Thus we may run both solvers in parallel and let them interchange the "new information", or one solver can dynamically help the another one with a certain subproblem, etc. We will focus on the scenario when an algebraic solver helps a SAT solver because it provides the best results according to our initial experiments. For more details on conversions, see [10].

Previously, we had handled the interaction of two solvers manually. During our experiments, several examples were observed where one solver is sped up by utilizing information derived by the other. Based on these observations, the communication was automated with a view towards optimizing the achievable gains.

The integration is tailored to be applicable for most SAT solvers. For our experiments, we used the SAT solver `antom` [16]. Modern SAT solvers are mainly based on CDCL. They produce many *conflict clauses* which contain new information that can be potentially used in the BBBA after a conversion. On the other hand, any new polynomial found in the ideal by the BBBA can be converted and sent to a SAT solver. To reduce the amount of information that needs to be transferred, we transmit only short clauses and short polynomials of a low degree. Moreover, an additional filtering technique has been developed to further reduce the number of clauses that are handled by the BBBA. This selection strategy makes the BBBA sufficiently fast to keep up with the SAT solver. In this way the BBBA is not stuck with computations which are potentially outdated and irrelevant for the SAT solver by the time they are finished.

Description of the integration. Assume that the SAT solver is running on a given CNF in the background. Our approach is divided into 7 steps (viewed from the BBBA side) which repeat until the SAT solver stops:

1. *Receiving clauses.* The SAT solver sends a set of new conflict clauses C that it has generated to the BBBA.
2. *Clause filtration.* We define a subset $C' \subseteq C$, where C' contains clauses $c \in C$ such that there exist $c' \in C$ with $c \neq c'$ that shares at least one variable with c. We buffer only the first 10 clauses on an as-they-come basis.
3. *Converting clauses to polynomials.* We use the standard conversion [10, Algorithm 1] to produce Boolean polynomials from the selected clauses.
4. *Computing a border basis.* We call Algorithm 8 on the output of the previous step. We restrict the sets of indeterminates of the Boolean ring to the indeterminates actually appearing in the input polynomials. We do not apply `FinalReduction`.
5. *Polynomial filtration.* We choose only linear, quadratic or cubic polynomials produced by Algorithm 8 that are different from the input of the BBBA. Among them, we select polynomials with the smallest support.
6. *Converting polynomials to clauses.* We convert these polynomials to clauses via the (sparse) truth-table method described in [10, Example 1]. We buffer only the first 100 clauses on an as-they-come basis.
7. *Sending the clauses.* We send these clauses to the SAT solver and go to Step 1.

6 Design of the Communication

To combine the power of the SAT solver with the advanced reasoning of the BBBA, a severe communication challenge has to be overcome. While it would be possible to create a fully integrated BBBA-SAT hybrid solver, the maintenance of such a solver would be difficult and the implementation of new features into either base solver challenging. Therefore, a communication framework that allows the exchange of data between the border basis and the SAT solver is developed instead. The design of this communication layer focused on two objectives: 1. The overhead for the data transfer must be low. 2. The base solvers should be modified as little as possible.

To achieve the first design goal, a shared memory communication approach is chosen. By defining a shared memory region that is accessible to both solvers, large amounts of data can be transmitted at extremely high speeds. To satisfy the second objective, the communication is restricted to consist only of clauses. Furthermore, the shared memory communication is implemented with the help of the `Boost Interprocess` Library [9]. This library allows different processes to access a common, shared memory region. Thus, the SAT solver and the BBBA can be executed independently and simply access the same shared memory region.

The communication itself is combined into a handler class that performs the generation of the shared memory region and the coordination and synchronization of the data access. It furthermore provides a simple interface for the sending and receiving of clauses. The handler class only needs to be instantiated by the solvers to gain access to the shared memory region.

When the BBBA and the SAT solver are combined, there are two instances of the shared memory manager that are communicating. Apart from the initialization of the shared memory region that is performed at the start of the application, these instances behave exactly in the same way. For simplicity, the two managers are referred to as m_1 and m_2 here. To allow for an efficient communication, each manager utilizes its own shared memory area for outgoing clauses, o_1 and o_2. This enables a fast full duplex communication, as each manager can send and receive at the same time.

When m_1 is asked to transmit a clause to m_2, it first stores the clause in a local queue. The next clause of the local queue is transferred to the shared region o_1 when m_2 indicates that it read the previously shared clause from that area. Once the clause has been written to o_1, m_1 informs m_2 that a new clause is available. The manager m_2 then copies the clause from o_1 to its own memory region and marks the clause as read. Thus, the next clause in the queue of m_1 can be transmitted.

To avoid any idle waiting in the background, the check for new clauses and the transmission of the next clause are only performed when the solvers update their shared memory handler.

7 Modifications of the SAT Solver

The SAT solver constantly generates new conflict clauses. The shear volume of conflict clauses makes it unfeasible to share all of them with the BBBA. Instead, only conflict clauses below a certain size threshold are transmitted. A new conflict clause is transmitted immediately after it has been generated. This modification adds only a single line of code to the solver.

Receiving new clauses is slightly more challenging because of the way clauses are stored and considered in the solver `antom` [16]. For efficiency reasons, the first literal of every clause that is not satisfied must be free (i.e., currently not assigned to a value). Therefore, each new clause that is received from the BBBA is first sorted and then added. Depending on the variable assignment that is currently under consideration by the SAT solver, a new clause might, furthermore, be unsatisfied at the moment. In this case, the solver backtracks until the clause is not unsatisfied anymore. Here the new clause acts similar to a conflict clause and guides the solver away from unsatisfied regions of the search space. The additional tasks that are required to handle a received clause are placed into a new function. Thus, the main SAT solver code only needs to be extended by a single line of code that checks for the arrival of new clauses.

The shared memory handler is updated once after every decision. This is sufficiently often to receive any new clauses, but does not add any undue overhead to the solver.

Overall, the SAT solver is modified only to a very small extent. Hence the solver can be freely developed without worrying about complex dependencies. Similarly, it should be comparatively easy to add the presented communication layer to a different SAT solver, should the need arise.

8 Experiments and Timings

To evaluate the combination of BBBA and `antom`, two different kinds of experiments have been performed. Timings in this paper were obtained on a computer under Linux having a 2.60 GHz Intel Core i7-5600U CPU and a total of 16 GB RAM. We note here that `antom` is deterministic, i.e. it gives the same result on the same input for each run.

8.1 Manual Combination of the Information

The first set of experiments is meant to showcase the general usefulness of the information that is derived by the BBBA for the SAT solver. Let C be a CNF input instance for `antom`. We convert C into a set of Boolean polynomials S via the standard conversion. Next we run the BBBA for 5 min and then stop the execution. We select one linear polynomial f in V manually and convert f back to CNF via the truth-table method. Let C' be its result. We run `antom` twice: once with the input C and then with the input $C \wedge C'$. The timings in Table 1 illustrates the speed-up obtained by manual section of extra information provided by the BBBA.

Table 1. Comparison of timings of `antom` on the Small Scale AES instances in [8] without vs with extra clauses corresponding to a linear polynomial.

`CNF instances`	`antom`	`antom` + lin. poly
`AES-2-1-2-8`	11.80	3.50
`AES-2-2-1-8`	111.59	88.15
`AES-2-2-4-4`	196.06	21.76
`AES-1-2-4-8`	666.93	209.11
`AES-2-4-2-4`	3997.91	1432.24

8.2 Automatic Combination of the Information

For the second set of experiments, the newly developed Algorithm 8 and the integration framework with `antom` in C++ as described in Sect. 5 were used. In Table 2 we present the timings of this automation on various benchmarks. Instances factoringx, y were generated by [2]. They encodes the factoring problem for $x \cdot y$. The other benchmarks encode algebraic attacks or algebraic fault attacks on the cryptosystems Small Scale AES and LED-64. For the full description of these benchmarks, we refer to [4,8]. The timeout limit was set to 1200 s.

Table 2. Timings of the integration of the BBBA with antom vs vanilla antom for various SAT instances.

CNF instances	antom	BBBA + antom
factoring81551,100057	0.23	0.22
AES-2-2-4faultInNibble1with1faultyBits	3.12	2.35
factoring3981643,3981641	7.83	6.91
factoring2190823,2190821	19.53	74.25
factoring7367627,7367621	29.18	146.13
factoring12619463,12619427	40.43	101.34
AES-4-4-4faultInNibble1with4faultyBits	41.15	55.87
LED64faultInNibble1with1faultyBits	45.70	55.38
AES-4-4-4faultInNibble1with1faultyBits	49.02	48.61
factoring5160011,5160007	63.98	55.47
factoring5621809,5621809	81.54	110.07
factoring4752977,4752949	207.18	189.58
factoring5308571,5308553	282.91	37.22
AES-2-2-4-4algebraicCNF	268.70	235.39
factoring49987277,49999553	337.58	45.29
factoring12598967,12598951	441.88	78.60
factoring4593761,4593737	527.22	10.11
factoring5287813,5287801	605.76	102.48
factoring5620907,5620907	653.63	5.04
factoring10000079,10000019	>1200	760.41

During our experiments we found examples where the integration was slower than the SAT solver by itself. In practice, we therefore suggest to run the SAT solver alone on one machine and the integration in parallel on another machine. In this way, we cannot be "unlucky" and we will always profit from the best timings. This is particularly relevant in cryptanalytic scenarios where the solution of an instance implies breaking a cryptosystem.

Notice that the timings of the integration of the two solvers are sometimes not stable, i.e. two timings for the same instance may differ substantially. These differences occur because new clauses are added at different points in time.

The filtration techniques described in Sect. 5, as well as the integration itself, are still preliminary. Our next goal is to develop deeper understanding of the synergy of both solvers. The main difficulty is that SAT solvers use various heuristics for literal assignment and for the choice on what clause to work on next. This makes it very hard to analyze which extra clauses from the BBBA affect the timings most. Nonetheless, our results show that the additional information from the BBBA already greatly increases the speed of the SAT solver.

Acknowledgments. We would like to express our gratitude to Tobias Schubert for providing us with the source code of the SAT solver `antom`. This work was financially supported by the DFG project "Algebraische Fehlerangriffe" [KR 1907/6-1].

References

1. Bard, G., Courtois, N., Jefferson, C.: Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over GF(2) via SAT-solvers. In: IACR Cryptology ePrint Archive (2007). https://eprint.iacr.org/2007/024.pdf
2. Bebel, J., Yuen, H.: Hard SAT instances based on factoring. In: SAT Competition 2013: Solver and Benchmark Descriptions, University of Helsinki, p. 102 (2013)
3. Brickenstein, M.: Boolean Gröbner Bases: Theory, Algorithms and Applications. Logos Verlag, Berlin (2010)
4. Burchard, J., Gay, M., Messeng Ekossono, A.S., Horáček, J., Becker, B., Schubert, T., Kreuzer, M., Polian, I.: AutoFault: towards automatic construction of algebraic fault attacks. In: Proceedings of Conference on Fault Diagnosis and Tolarence in Cryptography (FDTC 2017), Taipei (2017, to appear)
5. Burchard, J., Messeng Ekosonno, A.-S., Horáček, J., Gay, M., Becker, B., Schubert, T., Kreuzer, M., Polian, I.: Towards mixed structural-functional models for algebraic fault attacks on ciphers. In: Proceedings of International Verification and Security Workshop (IVSW 2017) (2017)
6. Condrat, C., Kalla, P.: A Gröbner basis approach to CNF-formulae preprocessing. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 618–631. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_48
7. Dreyer, A., Nguyen, T.H.: Improving Gröbner-based clause learning for SAT solving industrial sized Boolean problems. In: Young Researcher Symposium (YRS) (Kaiserslautern 2013), Fraunhofer ITWM, pp. 72–77 (2013)
8. Gay, M., Burchard, J., Horáček, J., Messeng Ekossono, A.S., Schubert, T., Becker, B., Kreuzer, M., Polian, I.: Small scale AES toolbox: algebraic and propositional formulas, circuit-implementations and fault equations. In: Conference on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2016), Barcelona (2016)
9. Gaztanaga, I.: The Boost Interprocess Library, version 1.63.0. www.boost.org/doc/libs/1_63_0/doc/html/interprocess.html
10. Horáček, J., Kreuzer, M.: On conversions from CNF to ANF. In: 2th International Workshop on Satisfiability Checking and Symbolic Computation, SC-square, Kaiserslautern (2017)
11. Horáček, J., Kreuzer, M., Messeng Ekossono, A.S.: Computing Boolean border bases. In: Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2016), Timisoara, pp. 465–472. IEEE (2016)
12. Horáček, J., Kreuzer, M., Messeng Ekossono, A.S.: A signature based border basis algorithm. In: Conference on Algebraic Informatics, CAI, Kalamata (2017)

13. Jovanovic, P., Kreuzer, M.: Algebraic attacks using SAT-solvers. Groups Complex. Cryptol. **2**, 247–259 (2010)
14. Kehrein, A., Kreuzer, M.: Computing border bases. J. Pure Appl. Algebra **205**(2), 279–295 (2006)
15. Kreuzer, M., Robbiano, L.: Computational Commutative Algebra 1. Springer, Heidelberg (2000)
16. Schubert, T., Reimer, S.: Antom (2016). https://projects.informatik.uni-freiburg.de/projects/antom
17. Zengler, C., Küchlin, W.: Extending clause learning of SAT solvers with boolean Gröbner bases. In: Gerdt, V.P., Koepf, W., Mayr, E.W., Vorozhtsov, E.V. (eds.) CASC 2010. LNCS, vol. 6244, pp. 293–302. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15274-0_26