A SIGNATURE BASED BORDER BASIS ALGORITHM

JAN HORÁČEK, MARTIN KREUZER, AND ANGE-SALOMÉ MESSENG E.

Abstract. The Border Basis Algorithm (BBA) still suffers from the lack of analogues of Buchberger's criteria for avoiding unnecessary reductions. In this paper we develop a signature based technique which provides a first remedial step: signature bounds allow us to recognize multiple reductions of the same ancestor polynomial. The new signature based algorithm is also combined with the Boolean BBA for ideals of Boolean polynomials. Experiments show that it is at least 5 times faster than the standard (Boolean) BBA.

1. Introduction

One of the central algorithms of computer algebra is the algorithm for computing Gröbner bases introduced by B. Buchberger in 1965 (cf. [2]). Significant efforts have been expended to improve its performance. The best current implementations use signature based versions of Buchberger's algorithm, the first of which was J.-C. Faugere's algorithm F5 (cf. [4]). Nowadays an entire zoo of such algorithms has been developed and their behavior has been studied thoroughly (see for instance [3]).

On the other hand, the Border Basis Algorithm (BBA), a framework for which was introduced in [10] and whose details were worked out in [7], is much less researched. In [6], the authors considered some optimizations of BBA for ideals of Boolean polynomials. However, for interesting ideals originating from cryptographic attacks, these optimizations still proved to be insufficient to produce running times comparable with optimized implementations of Buchberger's algorithm. The main reason for this is that the current implementations of the BBA still lack analogues of Buchberger's criteria for avoiding unnecessary reductions of critical pairs.

Our main goal in this paper is to go the first step in the direction to construct border basis analogues to these criteria. More specifically, let $I = \langle f_1, \ldots, f_s \rangle$ be the 0-dimensional ideal whose border basis we are calculating, and let V be the current tuple of polynomials generating a vector space which will contain the desired border basis eventually. To each polynomial g that we have to consider, we assign a signature bound which is a pair (t, i) with a term t and $1 \leq i \leq s$ that remembers the multiple tf_i of the input polynomial f_i whose (linear) reduction is g. Thus, if the same signature appears again, we can avoid the reduction of g against the polynomials in V because we know that it reduces to zero. As we shall see, in this way we avoid many repetitions and the algorithm becomes significantly faster.

The paper is structured as follows. In Section 2 we recall the definition of border bases and the standard version of the Border Basis Algorithm (BBA). In fact,

Key words and phrases. border basis, border basis algorithm, signature, Boolean polynomial, cryptographic attack, SAT solving.

we provide the slightly optimized version given in [7], Prop. 21. Then Section 3 contains the heart of the paper. For an element g of the ideal generated by a tuple of polynomials $\Phi = (f_1, \ldots, f_s)$, we introduce the notions of a *signature bound* and the *signature* of g with respect to Φ. More precisely, a signature bound is a tuple (t, i) with a term t and an index $1 \leq i \leq s$ such that tf_i occurs among the summands having maximal leading term in a representation $g = \sum_i c_i t_i f_{j_i}$ with scalars c_i and terms t_i. A signature bound allows us to recognize multiple occurrences of the same or equivalent polynomials during the reduction phase of the BBA. This idea is elaborated in the Signature Based BBA (SBBA), and in particular in the new stable span procedure `SStab` (see Algorithm 4). One nice feature of the SBBA is the possibility to choose a selection strategy for the next signature bound to work on, a freedom which deserves to be explored further.

In Section 4 we look at the Boolean BBA (BBBA) introduced in [6] and combine it with the new signature based technique. The result is the Signature Based Boolean BBA (SBBBA) of which we implemented an ApCoCoA prototype (see [1]) and an optimized C++ version. The details of the optimizations and implementation tricks for the C++ version are explained in Section 5. Finally, in Section 6 we provide some experiments and timings. We tested the new algorithms for some polynomial ideals representing algebraic attacks and algebraic fault attacks in cryptography. These experiments show that SBBBA is about 5 times faster than BBBA, that the performance gap between border basis and Gröbner basis techniques has been narrowed, and that SAT solving is several magnitudes faster for this kind of example.

Unless explicitly stated otherwise, we use the basic definitions and notation of [8] and [9]. The terminology surrounding the Border Basis Algorithm is explained in [7], and for the Boolean BBA we refer to [6]. The prototype implementation of our algorithms were done using the computer algebra system ApCoCoA (see [1]).

2. The Standard Border Basis Algorithm

In the following we let K be a field, $P = K[x_1, \ldots, x_n]$ a polynomial ring over K, and $f_1, \ldots, f_s \in P$ polynomials which generate a 0-dimensional polynomial ideal $I = \langle f_1, \ldots, f_s \rangle$. The goal of the Border Basis Algorithm (BBA) is to compute a special system of generators of I which corresponds to a K-basis of the residue class ring $R = P/I$ of the following type.

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) A (finite) subset $\mathcal{O}$ of $\mathbb{T}^n$ is called an **order ideal** if it is divisor closed, i.e., if $t \in \mathcal{O}$ and $t' \mid t$ implies $t' \in \mathcal{O}$.

(b) Let $t_1, \ldots, t_k \in \mathbb{T}^n$. Then the order ideal

$$\langle t_1, \ldots, t_k \rangle_{\mathrm{OI}} = \{t' \in \mathbb{T}^n \mid t' | t_i \text{ for some } i \in \{1, \ldots, k\}\}$$

is called the order ideal **spanned by** $t_1, \ldots, t_k$. The terms $t_1, \ldots, t_k$ are called the **cogenerators** of this order ideal.

(c) Given an order ideal $\mathcal{O}$ in $\mathbb{T}^n$, the set $\partial\mathcal{O} = (x_1\mathcal{O} \cup \cdots \cup x_n\mathcal{O}) \setminus \mathcal{O}$ is called the **border** of $\mathcal{O}$.

(d) Given an order ideal $\mathcal{O}$ in $\mathbb{T}^n$, the order ideal $\mathcal{O}^+ = \mathcal{O} \cup \partial\mathcal{O}$ is called the (first) **extension** of $\mathcal{O}$.

The following example illustrates these definitions.

Example 2.2. In the polynomial ring $P = K[x_1, x_2]$, the set $\mathcal{O} = \{1,\, x_1,\, x_2,\, x_1x_2\}$ is an order ideal. We have $\mathcal{O} = \langle x_1x_2\rangle_{\mathrm{OI}}$, i.e., the order ideal $\mathcal{O}$ is cogenerated by x_1x_2. Moreover, the border of $\mathcal{O}$ is $\partial\mathcal{O} = \{x_1^2, x_1^2x_2, x_1x_2^2, x_2^2\}$, and the extension of $\mathcal{O}$ is

$$\mathcal{O}^+ \;=\; \mathcal{O} \cup \partial\mathcal{O} \;=\; \{1,\, x_1,\, x_2,\, x_1x_2,\, x_1^2,\, x_1^2x_2,\, x_1x_2^2,\, x_2^2\}$$

One way to construct order ideals is to take complements of monomial ideals. E.g., for every term ordering σ, the set $\mathcal{O}_\sigma(I) = \mathbb{T}^n \setminus \mathrm{LT}_\sigma(I)$ is an order ideal. Given an order ideal $\mathcal{O}$, we are looking for the following kind of special generating system of I.

Definition 2.3. Let $\mathcal{O} = \{t_1, \ldots, t_\mu\}$ be an order ideal in $\mathbb{T}^n$, and let $\partial\mathcal{O} = \{b_1, \ldots, b_\nu\}$.

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

(b) An $\mathcal{O}$-border prebasis G is called an **$\mathcal{O}$-border basis** of the ideal $I = \langle G\rangle$, if the residue classes of the terms in $\mathcal{O}$ form a K-basis of P/I.

Let us examine this definition in the setting of the above example.

Example 2.4. Let P and $\mathcal{O}$ be defined as in Example 2.2.

(a) The set $G = \{g_1, g_2, g_3, g_4\}$, where $g_1 = x_1^2 - 1$, $g_2 = x_1^2x_2 - x_1x_2$, $g_3 = x_1x_2^2$, and $g_4 = x_2^2$, is an $\mathcal{O}$-border prebasis of $I = \langle G\rangle$. However, it is not an $\mathcal{O}$-border basis, since $x_2g_1 - g_2 = x_1x_2 - x_2 \in I$ shows that $\overline{x_2} = \overline{x_1x_2}$ in P/I.

(b) The set $H = \{h_1, h_2, h_3, h_4\}$, where $h_1 = x_1^2 - 1$, $h_2 = x_1^2x_2 - x_2$, $h_3 = x_1x_2^2$, and $h_4 = x_2^2$, is an $\mathcal{O}$-border basis of $J = \langle H\rangle = \langle x_1^2 - 1, x_2^2\rangle$, since $\overline{\mathcal{O}}$ is a K-basis of P/I by Macaulay's Basis Theorem (cf. [8], Thm. 1.5.7).

If an order ideal is of the form $\mathcal{O}_\sigma(I)$ for some term ordering σ, then I has an $\mathcal{O}_\sigma(I)$-border basis and this border basis contains the reduced σ-Gröbner basis of I (see [9], Prop. 6.4.18). The goal of the Border Basis Algorithm (BBA) is to compute such a border basis. To formulate it, we use the following terminology.

Definition 2.5. Let $p \in P\setminus\{0\}$, and let $G = (g_1, \ldots, g_k)$ be a tuple of polynomials $g_i \in P \setminus \{0\}$.

(a) The polynomial p is called **LT_σ-independent from** G if $\mathrm{LT}_\sigma(p)$ is not one of the leading terms $\mathrm{LT}_\sigma(g_1), \ldots, \mathrm{LT}_\sigma(g_k)$.

(b) The tuple G is called **LT_σ-independent** if the leading terms $\mathrm{LT}_\sigma(g_1)$, $\ldots$, $\mathrm{LT}_\sigma(g_k)$ are pairwise distinct.

In the following algorithm, the procedure `FinalReduction`$(V, \mathcal{O})$ refers to the one defined in [7], Prop. 17, and the procedure `Stab`(V, U) refers to the one defined below (see also [7], Prop. 13). For a tuple $V = (v_1, \ldots, v_k)$, we let $\mathrm{LT}_\sigma(V) = (\mathrm{LT}_\sigma(v_1), \ldots, \mathrm{LT}_\sigma(v_k))$ and denote by $\langle \mathrm{LT}_\sigma(V)\rangle$ the monomial ideal generated by these terms.

The main work in this algorithm is performed by the procedure `Stab`(V, U) defined below. Recall that for a tuple of polynomials V we let V^+ be the concatenation of V with $x_1V, \ldots, x_nV$.

Algorithm 1 Standard Border Basis Algorithm (BBA)
Input: Generators $\{f_1, \ldots, f_s\}$ of a 0-dimensional ideal I and a degree compatible term ordering σ.
Output: The order ideal $\mathcal{O} = \mathcal{O}_\sigma(I)$ and the $\mathcal{O}_\sigma(I)$-border basis G 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_σ-independent K-vector space basis of $\langle f_1, \ldots, f_s \rangle_K$.
3: **repeat**
4: Execute the procedure `Stab`(V, U) and get a new pair (V, U).
5: Let $\mathcal{O} := U \setminus \langle \mathrm{LT}_\sigma(V) \rangle$.
6: Let $U_{\mathrm{old}} := U$ and $U := U^+$.
7: **until** $\partial\mathcal{O} \subset U_{\mathrm{old}}$
8: Apply `FinalReduction`$(V, \mathcal{O})$ and return its output $(\mathcal{O}, G)$.

Algorithm 2 (Stab)
Input: An LT_σ-independent tuple of polynomials $V = (v_1, \ldots, v_k)$ and an order ideal U.
Output: A new pair (V, U).

1: **repeat**
2: Calculate a tuple W such that the concatenation $V \cup W$ is an LT_σ-independent K-basis of $\langle V^+ \rangle_K$.
3: **repeat**
4: $W' := \{w \in W \mid \mathrm{LT}_\sigma(w) \in U\}$
5: $U' := \langle \bigcup_{w \in W'} \mathrm{Supp}(w) \rangle_{\mathrm{OI}} \setminus U$
6: $U := U \cup U'$
7: **until** $U' = \emptyset$
8: Append the elements of W' to V.
9: **until** $W' = \emptyset$
10: Return (V, U).

For a proof of the correctness of the Standard BBA, we refer the reader to [7], Prop. 21. The order ideal U in this algorithm is commonly called the (computational) **universe**. The main step, which is Step 2 in `Stab`(V, U), consists of repeated Gaußian type reductions in the vector space $\langle U \rangle_K$. Hence its cost depends strongly on the size of U and it is imperative to keep that size small. Unfortunately, replacing U by U^+ in Step 6 increases U rapidly. Another major drawback of the Standard BBA is that the number of new polynomials which have to be reduced increases rapidly. One reason is that we do not recognize duplicities such as reducing both $x_1 g_2$ and $x_2 g_1$, although g_1 is the reduction of $x_1 f_i$ and g_2 is the reduction of $x_2 f_i$ for the same i. The central idea of the Signature BBA is to avoid this extra work and thus keep V and U as small as possible at all times.

3. The Signature Based BBA

In the setting of the preceding section, we choose a degree compatible term ordering σ. Let us define what we mean by the signature of a polynomial in I.

Definition 3.1. Let $I = \langle f_1, \ldots, f_s \rangle$ be a 0-dimensional polynomial ideal, let $\Phi = (f_1, \ldots, f_s)$, and let $g \in I \setminus \{0\}$.

(a) Given representation $g = \sum_{i=1}^{\ell} c_i t_i f_{j_i}$ with $c_i \in K \setminus \{0\}$, $t_i \in \mathbb{T}^n$, and $j_i \in \{1, \ldots, s\}$, let $k \in \{1, \ldots, \ell\}$ be such that $\mathrm{LT}_\sigma(t_k f_{j_k}) = \max_\sigma\{\mathrm{LT}_\sigma(t_i f_{j_i}) \mid i = 1, \ldots, \ell\}$. Then we call (t_k, j_k) a **signature bound** for g.

(b) Consider all representations $g = \sum_{i=1}^{\ell} c_i t_i f_{j_i}$ as above for which the term $\max\{\mathrm{LT}_\sigma(t_1 f_{j_1}), \ldots, \mathrm{LT}_\sigma(t_\ell f_{j_\ell})\}$ is minimal with respect to σ. Among these, let $j_m \in \{1, \ldots, s\}$ be the minimal value of j_i which shows up in a summand $c_m t_m f_{j_m}$ having the maximal leading term. Then the pair $\mathrm{Sig}_\Phi(g) := (t_m, j_m)$ is called the **signature** of g with respect to Φ. If the tuple of generators Φ is clear, we simply write $\mathrm{Sig}(g)$.

For a deeper understanding of the meaning of signature bounds and signatures, we refer the reader to the related notions of σ-degree and σ-leading form in [8], Section 2.3. Given a polynomial $g \in I \setminus \{0\}$ with $\mathrm{Sig}(g) = (t, i)$ and $j \in \{1, \ldots, n\}$, we also write $x_j \cdot \mathrm{Sig}(g)$ instead of $(x_j t, i)$. The notation can be generalized to tuples of polynomials as follows.

Definition 3.2. Let $G = (g_1, \ldots, g_k)$ be a tuple with $g_i \in I \setminus \{0\}$ for $i = 1, \ldots, k$, and let $S_G = (s_1, \ldots, s_k)$ be a tuple of signature bounds for G.

(a) The tuple $\mathrm{Sig}(G) = (\mathrm{Sig}(g_1), \ldots, \mathrm{Sig}(g_k))$ is called the **signature of** G.

(b) The tuple S_G^+ is now defined as follows:

$$S_G^+ \;=\; (s_1, \ldots, s_k,\; x_1 s_1, \ldots, x_1 s_k,\; x_2 s_1, \ldots, x_2 s_k,\; \ldots, x_n s_1, \ldots, x_n s_k)$$

Subsequently, we order signature bounds as follows.

Definition 3.3. Let $t_1, t_2 \in \mathbb{T}^n$ and $i, j \in \{1, \ldots, s\}$. Then we define

$$(t_1, i) \preceq (t_2, j) \quad \Leftrightarrow \quad \begin{cases} \mathrm{LT}_\sigma(t_1 f_i) <_\sigma \mathrm{LT}_\sigma(t_2 f_j), & \text{or} \\ \mathrm{LT}_\sigma(t_1 f_i) = \mathrm{LT}_\sigma(t_2 f_j) & \text{and } i \leq j \end{cases}$$

Notice that this implies $\mathrm{Sig}(x_i g) \preceq x_i \,\mathrm{Sig}(g)$ for $i \in \{1, \ldots, n\}$ and $g \in I \setminus \{0\}$, because multiplying the minimal representation of g by x_i yields one representation of $x_i g$, but not necessarily the minimal one.

The main idea of the Signature Based Border Basis Algorithm (SBBA) below is to avoid unnecessary reductions in Step 2 of Algorithm 2. For every component v_i of the tuple V we store a signature bound $\mathrm{Sig}(v_i) \preceq (t_i, j_i)$ such that v can be obtained by reducing the polynomial $t_i f_{j_i}$ with other elements of V. In other words, the signature remembers the "ancestor" $t_i f_{j_i}$ from which v_i was derived. Now, if we encounter another element w with signature (t_i, j_i), we can show that w reduces to zero using V. Hence w need not be considered in the calculation of the basis extension in Step 2 of Algorithm 2. A typical case of this occurs if v is the reduction of $x_2 f_i$, if $x_1 v$ has already been reduced, and if we now consider $x_2 w$, where w is a reduction of $x_1 f_i$. When we discover that the signature bounds of $x_1 v$ and $x_2 w$ are both $(x_1 x_2, i)$, we can skip the reduction of $x_2 w$. Thus we can avoid a lot of duplicate work. This idea leads to Algorithm 3 (SBBA).

Algorithm 3 Signature Based Border Basis Algorithm (SBBA)
Input: A tuple $\Phi = (f_1, \ldots, f_s)$ which generates a 0-dimensional ideal I and a degree compatible term ordering σ.
Output: The order ideal $\mathcal{O} = \mathcal{O}_\sigma(I)$ and the $\mathcal{O}_\sigma(I)$-border basis G of I.

1: Let $U = \langle \mathrm{Supp}(f_1) \cup \cdots \cup \mathrm{Supp}(f_s) \rangle_{\mathrm{OI}}$.
2: Calculate an LT_σ-independent K-vector space basis $V = (v_1, \ldots, v_k)$ of $\langle f_1, \ldots, f_s \rangle_K$.
3: Let $S_V := ((1,1), \ldots, (1,k))$ and $S_0 := \emptyset$.
4: **repeat**
5: Execute `SStab`(V, S_V, S_0, U) and get a new tuple (V, S_V, S_0, U).
6: Let $\mathcal{O} := U \setminus \langle \mathrm{LT}_\sigma(V) \rangle$.
7: Let $U_{\mathrm{old}} := U$ and $U := U^+$.
8: **until** $\partial\mathcal{O} \subset U_{\mathrm{old}}$
9: Apply `FinalReduction`$(V, \mathcal{O})$ and return the result.

This algorithm is clearly structured as the usual BBA, but it uses a different stabilization procedure given by Algorithm 4.

Proposition 3.4. *Algorithm 3 (SBBA) computes the order ideal $\mathcal{O}_\sigma(I)$ and the $O_\sigma(I)$-border basis of the 0-dimensional ideal $I = \langle f_1, \ldots, f_s \rangle_K$.*

Proof. If we compare Algorithm 3 to Algorithm 1, we see that the only non-trivial difference occurs in the stabilization procedure, where `SStab` performs the computation of the basis extension W of Step 2 of `Stab` in a specific way. Thus we have to prove that `SStab` returns a tuple (V, S_V, S_0, U) where U is the (possibly enlarged) universe containing the supports of all polynomials in V, and where V is an LT_σ-independent basis of the U-stable span of the input tuple V. The former condition is clearly enforced by Steps 20-26 of `SStab`. The latter condition means $\langle V^+ \rangle_K \cap \langle U \rangle_K = \langle V \rangle_K$ (see [7], Def. 10). To verify it, we need to look at the main differences between `Stab` and `SStab`.

One difference is that in the first inner **repeat** loop of `SStab`, we are not considering all candidates $x_i v_j$ and reducing them K-linearly against $V \cup W'$ to find new elements of the stable U-span. Instead, we only consider the candidates $x_i v_j$ for which the corresponding signature bound s_ℓ is not in $S_V \cup S_{W'} \cup S_0$. Therefore we need to show that the other candidates $x_i v_j$ do not yield new elements of $\langle V \cup W' \rangle_K \cap \langle U \rangle_K$. The second difference is that we use $V \cup W' \cup B$ to reduce a candidate $x_i v_j$, in contrast to using only $V \cup W'$.

To address these differences, we first look at the case $s_\ell \in S_0$. A signature bound is put into S_0 in Step 11 of `SStab` only if the corresponding element $x_i v_j$ reduces to zero using $V \cup W' \cup B$. If it reduces to zero using $V \cup W'$ only, we note that, after we have finished all iterations of the inner **repeat** loops of `SStab`, we have extended V to a basis of $\langle V^+ \rangle_K \cap \langle U \rangle_K$. Hence, if we multiply a zero reduction $v_j \longrightarrow 0$ by an indeterminate x_i, we get a zero reduction $x_i v_j \longrightarrow 0$ which still uses elements of the new tuple V. Thus we can replace the tuple of signature bounds S_0 representing elements that reduce to zero by the tuple $(S_0)^+$ in Step 28 of `SStab`.

On the other hand, suppose that the above reduction of $x_i v_j \longrightarrow 0$ involves an element of B. Then we know that $x_i v_j \longrightarrow x_i w \longrightarrow 0$ for some intermediate element $x_i w$ satisfying $\mathrm{LT}_\sigma(x_i w) \notin U$. However, this implies that we had $v_j \longrightarrow$

Algorithm 4 (SStab)
Input: An LT_σ-independent tuple $V = (v_1, \ldots, v_k)$ of polynomials, a tuple of signature bounds $S_V = (s_1, \ldots, s_k)$, another tuple of signature bounds S_0, and an order ideal U.
Output: A new tuple (V, S_V, S_0, U).

repeat
  $V^* := (x_1v_1, \ldots, x_1v_k, \ldots, x_nv_1, \ldots, x_nv_k)$
  $S_{V^*} := (x_1s_1, \ldots, x_1s_k, \ldots, x_ns_1, \ldots, x_ns_k)$
  Let $B := \emptyset$, let $W' := \emptyset$, and let $S_{W'} := \emptyset$.
  repeat
    Choose a signature bound s_ℓ in S_{V^*} and remove it from S_{V^*}.
    if $s_\ell \notin S_V \cup S_{W'} \cup S_0$ **then**
      Reduce the ℓ-th component x_iv_j of V^* K-linearly against $V \cup W' \cup B$ and get a polynomial v'.
      if $v' = 0$ **then**
        Append s_ℓ to S_0.
      end if
      if $v' \neq 0$ and $\mathrm{LT}_\sigma(v') \in U$ **then**
        Append v' to W' and s_ℓ to $S_{W'}$.
      end if
      if $v' \neq 0$ and $\mathrm{LT}_\sigma(v') \notin U$ **then**
        Append v' to B.
      end if
    end if
  until $S_{V^*} = \emptyset$
  $U := U \cup \langle \bigcup_{w \in W'} \mathrm{Supp}(w) \rangle_{\mathrm{OI}}$
  repeat
    $W'' := \{w \in B \mid \mathrm{LT}_\sigma(w) \in U\}$
    Move the elements of W'' from B to W' and the corresponding signature bounds from S_B to $S_{W'}$.
    $U' := \langle \mathrm{Supp}(w) \mid w \in W'' \rangle_{\mathrm{OI}} \setminus U$
    $U := U \cup U'$
  until $U' = \emptyset$
  Append W' to V and $S_{W'}$ to S_V.
  $S_0 := S_0^+$
until $W' = \emptyset$
return (V, S_V, S_0, U)

$w \longrightarrow 0$ and $\mathrm{LT}_\sigma(w) \notin U$ in a previous iteration, in contradiction to the fact that the signature bound for v_j was put into S_0.

Next we prove that the elements x_iv_j for which the corresponding signature bound s_ℓ is in $S_V \cup S_{W'}$ do not yield new elements of $\langle V \cup W' \rangle_K \cap \langle U \rangle_K$. Let $s_\ell = (\tilde{t}, k)$. We know that, using elements of $V \cup W'$, the element $\tilde{t}f_k$ reduces to an element $\tilde{v}$ of $V \cup W'$. Since $\tilde{v}$ has been put into V or W' before, the element $\tilde{t}f_k$ actually reduces to zero using $V \cup W'$. Moreover, s_ℓ is also of the form x_is_m where s_m is a signature bound for v_j. Writing $s_m = (\hat{t}, r)$, we get a reduction $x_i\hat{t}f_r \longrightarrow x_iv_j$ using elements of x_iV. If during this reduction we only use elements

of $V \cup W'$, we get from $x_i\hat{t}f_r = \tilde{t}f_k \in \langle V \cup W' \rangle_K$ that we have $x_i v_j \in \langle V \cup W' \rangle_K$, and thus $x_i v_j$ is not a new element of the U-stable span of V.

Now suppose that, during the reduction $x_i\hat{t}f_r \longrightarrow x_i v_j$, we arrive at an element $x_i w$ with $\mathrm{LT}_\sigma(x_i w) \notin U$. Again there are two cases. Starting from $x_i w$, if the remaining reduction steps use only elements of $V \cup W' \cup B$, then the element $x_i v_j$ is contained in $\overline{V} = \langle V \cup W' \rangle_K \oplus \langle B \rangle_K$, and the claim follows from $\overline{V} \cap \langle U \rangle_K = \langle V \cup W' \rangle_K \cap \langle U \rangle_K$.

Finally we need to consider the case of an intermediate element $x_i\tilde{w}$ which cannot be reduced further using $V \cup W' \cup B$. Then the reduction $\hat{t}f_r \longrightarrow \tilde{w}$ involves at least one step, because we have $x_i\hat{t}f_r \in \langle V \cup W' \rangle_K$ and $\mathrm{LT}_\sigma(x_i\tilde{w}) \notin U$. Hence $\mathrm{LT}_\sigma(\tilde{w})$ is smaller than $\mathrm{LT}_\sigma(v_j)$ and $\tilde{w}$ is a linear combination of elements of V which have smaller leading terms. As these elements will be considered in other iterations of the algorithm, the element $x_i v_j$ can be skipped. Notice that we are not getting into a loop of promises here, since the elements which we promise to treat in other iterations have a strictly smaller leading term. Furthermore, we remark that when we reach an element whose leading term is $\mathrm{LT}_\sigma(\tilde{w})$ during one of these iterations, that element will be appended to B and not constitute a new element of $\langle V \cup W' \rangle_K \cap \langle U \rangle_K$.

The second inner **repeat** loop in `SStab` merely moves some elements of B into W' because the enlargement of the universe in Step 20 has brought their leading term into U. Altogether, it follows that the inner **repeat** loops of `SStab` correctly calculate an LT_σ-independent basis $V \cup W'$ of $\langle V^+ \rangle_K \cap \langle U \rangle_K$. Therefore the outer **repeat** loop correctly calculates a (possibly enlarged) universe U and the stable U-span of the input tuple V. □

Let us point out that the elements s_ℓ in Algorithm 4 are not necessarily the exact signatures of the corresponding polynomials $x_i v_j$. Instead, they are upper bounds by the observation following Definition 3.3. For the algorithm, this does not matter, and in practice $s_\ell = \mathrm{Sig}(v')$ holds most of the time.

A particularly nice feature of the SBBA is that we are free to choose a selection strategy in Step 6 of `SStab` for the next signature to work on. Suitable choices could be the smallest untreated signature, the element $x_i v_j$ with the smallest leading term, the element $x_i v_j$ having the fewest terms in its support, a degree compatible strategy, or any combination of these. Future experiments have to indicate the best choice. A comparison to the analogous choice of a selection strategy in Buchberger's Algorithm indicates that this choice is likely to affect the running times of the algorithm substantially. It is quite conceivable that the best selection strategy depends on the example under consideration.

4. The Signature Based BBA for Boolean Polynomials

In this section we construct a signature based version of the BBA for Boolean polynomials. In particular, we show that we can combine the improvements offered by the signature based algorithm with the optimizations of the Boolean BBA introduced in [6].

Let us recall the setting used by the Boolean BBA. For the base field, we use the field of two elements $K = \mathbb{F}_2$. The ideal $F = \langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle$ is called the **field ideal** since it is the vanishing ideal of K^n. The ring

$$R = K[x_1, \ldots, x_n]/\langle x_1^2 + x_1, \ldots, x_n^2 + x_n \rangle.$$

is called the ring of **Boolean polynomials**. Its elements are represented by polynomials whose support is contained in $\mathbb{S}^n$, the set of squarefree terms. An arbitrary polynomial f represents the same Boolean polynomial as its normal form $\mathrm{NF}_F(f)$ with respect to the field ideal. Ideals in R are represented by ideals in $P = K[x_1, \ldots, x_n]$ containing F. A border basis G of a 0-dimensional ideal in P containing F has the shape $G = G^{\mathrm{sf}} \cup \{x_i t + t \mid i \in \{1, \ldots, n\}, t \in \mathcal{O}_i\}$ where $\mathcal{O}_i$ is the set of terms in $\mathcal{O}$ divisible by x_i. Thus it suffices to compute the part G^{sf} of G whose border terms are in $\mathcal{O}^{\mathrm{sf}} = \mathcal{O} \cap \mathbb{S}^n$.

For the convenience of the reader, we recall the following algorithm which combines the Boolean BBA (cf. [6], Alg. 4.3) with the improvements provided by the U-Extension Algorithm (cf. [6], Alg. 5.3 and 5.4).

Algorithm 5 Optimised Boolean BBA (OBBA)
Input: Generators $\{f_1, \ldots, f_s\}$ of a 0-dimensional ideal I containing F and a degree compatible term ordering σ.
Output: The squarefree subset $\mathcal{O}_\sigma(I)^{\mathrm{sf}}$ of $\mathcal{O}_\sigma(I)$ and the corresponding part of the $\mathcal{O}_\sigma(I)$-border basis of I.

1: Let $U = \langle \mathrm{NF}_F(\mathrm{Supp}(f_1)) \cup \cdots \cup \mathrm{NF}_F(\mathrm{Supp}(f_s)) \rangle_{\mathrm{OI}}$.
2: Calculate a tuple V which contains an LT_σ-independent K-vector space basis of $\langle \mathrm{NF}_F(f_1), \ldots, \mathrm{NF}_F(f_s) \rangle_K$.
3: **repeat**
4: Execute the procedure `OStab`(V, U) and get a new pair (V, U).
5: Let $\mathcal{O} := U \setminus \langle \mathrm{LT}_\sigma(V) \rangle_K$.
6: Let $U_{\mathrm{old}} := U$ and $U := U \cup \langle \partial \mathcal{O}^{\mathrm{sf}} \rangle_{OI}$.
7: **until** $\partial \mathcal{O}^{\mathrm{sf}} \subset U_{\mathrm{old}}$
8: Apply `FinalReduction`$(V, \mathcal{O})$ and return the result.

Here the procedure `OStab`(V, U) (see Algorithm 6) uses the **squarefree extension** $V^{(+)}$ of a tuple $V = (v_1, \ldots, v_k)$ which is defined by

$$V^{(+)} = (v_1, \ldots, v_k, \mathrm{NF}_F(x_1 v_1), \ldots, \mathrm{NF}_F(x_1 v_k), \ldots, \mathrm{NF}_F(x_n v_1), \ldots, \mathrm{NF}_F(x_n v_k))$$

Algorithm 6 (OStab)
Input: An LT_σ-independent tuple of polynomials $V = (v_1, \ldots, v_k)$ and an order ideal U.
Output: A new pair (V, U)

Let $i := 1$, let $V_1 := V$, let $W_0 := V$, and let $B_0 := \emptyset$.
repeat
  Increase i by one.
  Compute an LT_σ-independent basis extension W'_{i-1} for $\langle V_{i-1}\rangle_K \subseteq \langle V_{i-1} \cup B_{i-2} \cup W^{(+)}_{i-2}\rangle_K$.
  Let $W_{i-1} := \emptyset$ and $B_{i-1} := W'_{i-1}$.
  repeat
    $A = \{w \in B_{i-1} \mid \mathrm{LT}_\sigma(w) \in U\}$
    Append A to W_{i-1} and remove it from B_{i-1}.
    $U' := \langle \bigcup_{w\in A} \mathrm{Supp}(w)\rangle_{\mathrm{OI}} \setminus U$
    $U := U \cup U'$
  until $U' = \emptyset$
  Let $V_i := V_{i-1} \cup W_{i-1}$
until $W_{i-1} = \emptyset$
return (V_i, U)

Now we adapt Algorithm 3 (SBBA) to this Boolean setting.

Algorithm 7 Signature Based Boolean BBA (SBBBA)
Input: Generators $F = \{f_1, \ldots, f_s\}$ of a 0-dimensional ideal I containing F and a degree compatible term ordering σ.
Output: The squarefree subset $\mathcal{O}_\sigma(I)^{\mathrm{sf}}$ of $\mathcal{O}_\sigma(I)$ and the corresponding part of the $\mathcal{O}_\sigma(I)$-border basis of I.

Let $U := \langle \mathrm{NF}_F(\mathrm{Supp}(f_1)) \cup \cdots \cup \mathrm{NF}_F(\mathrm{Supp}(f_s))\rangle_{\mathrm{OI}}$.
Calculate a tuple V which contains an LT_σ-independent K-vector space basis of $\langle \mathrm{NF}_F(f_1), \ldots, \mathrm{NF}_F(f_s)\rangle_K$.
Let $S_V := ((1,1), \ldots, (1,k))$ and $S_0 := \emptyset$.
repeat
  Execute `SBStab`(V, S_V, S_0, U) and get a new tuple (V, S_V, S_0, U).
  Let $\mathcal{O} := U \setminus \langle \mathrm{LT}_\sigma(V)\rangle$.
  Let $U_{\mathrm{old}} := U$ and $U := U \cup \langle \partial\mathcal{O}^{\mathrm{sf}}\rangle_{\mathrm{OI}}$.
until $\partial\mathcal{O}^{\mathrm{sf}} \subset U_{\mathrm{old}}$
Apply `FinalReduction`$(V, \mathcal{O})$ and return the result.

Here the procedure `SBStab`(V, S_V, S_0, U) is defined in the following Algorithm 8. Notice that, for a signature bound (t, j) and an indeterminate x_i which divides t, the product $x_i \cdot (t, j) = (x_i t, j)$ corresponds the same Boolean polynomial as (t, j), since $\overline{x}_i\, \overline{t}\, \overline{f}_j = \overline{t}\, \overline{f}_j$ in R. Therefore, given a tuple of signature bounds $S = ((t_1, j_1), \ldots, (t_k, j_k))$, we let $S^{(+)}$ be the concatenation of S with all signature bounds $x_i \cdot (t_\ell, j_\ell)$ such that x_i does not divide t_ℓ.

Algorithm 8 (SBStab)
Input: An LT_σ-independent tuple $V = (v_1, \ldots, v_k)$ of polynomials, a tuple of signature bounds $S_V = (s_1, \ldots, s_k)$, another tuple of signature bounds S_0, and an order ideal U.
Output: A new tuple (V, S_V, S_0, U).

1: Let $i := 1$, let $V_1 := V$, let $W_0 := V$, and let $B_0 := \emptyset$.
2: Let $S_{V_1} := S_V$ and $S_{W_0} := S_V$, and let $S_{B_0} := \emptyset$.
3: **repeat**
4: Increase i by one.
5: Let $V^* := B_{i-1} \cup W_{i-2}^{(+)}$.
6: Let $S_{W_{i-2}} = (s'_1, \ldots, s'_m)$, and let S_{V^*} be the concatenation of $S_{B_{i-1}}$ and $(x_1 s'_1, \ldots, x_1 s'_m, \ldots, x_n s'_1, \ldots, x_n s'_m)$.
7: Remove from S_{V^*} all signatures $x_i s'_\ell = (x_i t, j)$ for which x_i divides t, and remove from V^* the corresponding elements $x_i w_\ell$.
8: Let $B_{i-1} := \emptyset$, let $W_{i-1} := \emptyset$, and let $S_{W_{i-1}} := \emptyset$.
9: **repeat**
10: Choose a signature s_ℓ in S_{V^*} and remove it from S_{V^*}.
11: **if** $s_\ell \notin S_{V_{i-1}} \cup S_{W_{i-1}} \cup S_0$ **then**
12: Reduce the ℓ-th component of V^* K-linearly against $V_{i-1} \cup W_{i-1} \cup B_{i-1}$ and get a polynomial v'.
13: **if** $v' = 0$ **then**
14: Append s_ℓ to S_0.
15: **end if**
16: **if** $v' \neq 0$ and $\mathrm{LT}_\sigma(v') \in U$ **then**
17: Append v' to W_{i-1} and s_ℓ to $S_{W_{i-1}}$.
18: **end if**
19: **if** $v' \neq 0$ and $\mathrm{LT}_\sigma(v') \notin U$ **then**
20: Append v' to B_{i-1} and s_ℓ to $S_{B_{i-1}}$
21: **end if**
22: **end if**
23: **until** $S_{V^*} = \emptyset$
24: $U := U \cup \langle \bigcup_{w \in W_{i-1}} \mathrm{Supp}(w) \rangle_{\mathrm{OI}}$.
25: **repeat**
26: $A := \{w \in B_{i-1} \mid \mathrm{LT}_\sigma(w) \in U\}$
27: Append A to W_{i-1} and remove it from B_{i-1}.
28: Move the corresponding signature bounds from $S_{B_{i-1}}$ to $S_{W_{i-1}}$.
29: $U' := \langle \bigcup_{w \in A} \mathrm{Supp}(w) \rangle_{\mathrm{OI}} \setminus U$
30: $U := U \cup U'$
31: **until** $U' = \emptyset$
32: Let $V_i := V_{i-1} \cup W_{i-1}$ and $S_{V_i} := S_{V_{i-1}} \cup S_{W_{i-1}}$.
33: Let $S_0 := S_0^{(+)}$.
34: **until** $W_{i-1} = \emptyset$
35: **return** (V_i, S_{V_i}, S_0, U)

The proof of the correctness of Algorithm 7 (SBBBA) follows by combining the proofs for SBBA and OBBA. Notice that we can drop a signature bound $(x_i t, j)$ with $x_i \mid t$ in Step 7 of `SBStab` since it corresponds to the same Boolen polynomial as (t, j) which is already in $S_{W_{i-2}}$, and hence in $S_{V_{i-1}}$.

5. Implementation of the Signature Based BBA

In this section we focus on the efficient implementation of the Signature Based Boolean BBA (SBBBA). Some observations on the implementation of the Boolean BBA (BBBA) can be found in [6]. They include efficient computations with order ideals and suitable data structures for Boolean polynomials that are stored as a coefficient matrix. (The polynomials correspond to rows in the matrix.) Similar implementation techniques and data structures can be applied to the general form of the BBA.

For the SBBBA implementation we may use a sparse representation of the matrix (i.e., the rows contain only indices of nonzero entries) and a sparse representation of the terms (i.e., only the indices of variables having exponent 1 are stored). This works well, since we are mainly working on sparse inputs, e.g., coming from cryptographic attacks. For larger base fields, we have to store the corresponding coefficients in the matrix, and for normal polynomials, we have to store the exponents of every indeterminate dividing a term.

A significant change of the design of the implementation is that we are storing the matrix corresponding to V only in REF. The rows corresponding to non-trivial elements of $V^{(+)}$ are created "on the fly", i.e., either a new row is reduced by already stored rows to zero and it is not appended to the matrix, or the reduction gives us a new row in the matrix. Thus this approach is more space efficient than the one in [6]. Pivots for reductions are carefully saved in the cache memory such that, given a row r, we have constant-time access to the row that can reduce r.

Signature bounds are nothing more than pairs (t, i), where t is a term and i is an integer. We associate initial signature bounds to the input polynomials and when producing a new polynomial during the calculation of $V^{(+)}$, we bind the corresponding signature bound to this new polynomial. The operations and orderings on signature bounds are very simple and need no special implementation tricks. We know that $1 \leq i \leq s$, where the number of generators s is a rather small number. Thus we can store a list of signature bounds S in a list of length s, where the i-th element of the list contains all terms t for which the signature bound (t, i) is in S.

For the implementation of SBBBA, we distinguish two types of signature bounds. First of all, there is the list of signature bounds S_V corresponding to the elements of V. For any signature bound (t, i), we need a fast procedure to decide if $(t, i) \in S_V$, i.e., we need a fast `find` method. In C++, there exist such structures and algorithms which have constant complexity in the average case, e.g., `std::unordered_map::find`.

The second type of signature bounds are stored in S_0. They correspond to polynomials which have been reduced to zero. Also for this list we need to have a fast decision procedure. Moreover, in Algorithm 4 we have to compute S_0^+ and in Algorithm 8 we need $S_0^{(+)}$. Using the same implementation as for S_V would fail here, because S_0 contains too many elements after several iterations of S_0^+. Instead, we use the same trick as for representing an order ideal via its cogenerators.

Definition 5.1. A set of signature bounds $\{(t_1, i), \ldots, (t_k, i)\} \subseteq S_0$ is called a set of **generators** of S_0 w.r.t. the input polynomial f_i if for every signature bound $(g, i) \in S_0$, the polynomial g is divisible by one of the terms $t_1, \ldots, t_k$. A set of generators $\{(t_1, i), \ldots, (t_k, i)\}$ of S_0 is called **minimal** if no term t_j divides t_k for $j \neq k$.

Using this representation, computing S_0^+ resp. $S_0^{(+)}$ does not fill up the memory, and its size is kept under control. On the other hand, the `find` function now has linear complexity in the length of generators. It is essential to note that the `find` functions for S_V and S_0 are critical to the performance of the implementation of SBBA and SBBBA, because they are called everytime before processing a new row. Choosing inappropriate data structures and algorithms would lead to significantly longer running times.

As mentioned previously, Step 6 of `SStab` (resp. Step 10 of `SBStab`) is very important. It affects the course of the algorithm in many ways. So far, choosing to work on the signature bound whose corresponding polynomial has the smallest degree, and among those the one having the fewest terms in its support, gave us the best results in our test runs. In fact, we influence the choice for the new pivot rows by preferring the shorter ones. As a result, a considerable speed-up is achieved merely by a clever selection of which polynomial in V^+ we work with next.

Then Step 7 of `SStab` (resp. Step 11 of `SBStab`) filters out many signature bounds that do not have to be treated. The corresponding polynomials do not even have to be created. Table 1 in the next section provides some profiling results about how many times this tends to happen.

6. Experiments and Timings

In this section we perform some experiments concerning the efficiency of the proposed improvements of the Boolean BBA. We use systems of quadratic Boolean polynomial equations coming from algebraic attacks and algebraic fault attacks at the Small Scale AES cryptosystem (see [5]), where we consider only its 4-bit version with state matrices of size 1×1, 1×2, and 2×2. We restrict our attention to ideals whose border basis can be computed within a time limit of 25 minutes. For each ideal, we list the number of variables and its number of generators.

All timings were obtained on a compute server having a 3.00 GHz Intel(R) Xeon(R) CPU E5-2623 v3 and a total of 48 GB RAM. The C++ programs implementing the different versions of the BBBA were compiled using the GCC compiler version 5.3.1 with the `-O2` optimization flag. Timings that exceed the timeout limit of 25 minutes are marked by "$*$". When measuring the time consumption, we take only the actual runtime in account. In particular, the initial memory allocation and the setup of Boolean rings are excluded. Since we are comparing algorithms called from other software systems to our native C++ implementation, that is usually faster in the initialization phase, this should be fair enough. The actual tests have been performed many times to assure that there is no disturbance during the computation which affects the running time.

Profiling of the BBA shows that the program spends more than 96% of the time in the reduction function, i.e., in Step 8 of `SStab` resp. Step12 of `SBStab`. Thus predicting and avoiding unnecessary reductions affects the actual runtime considerably. In Table 1 we measure the total number of calls to the reduction function. Here BBBA refers to the standard version described in [6]. The SBBBA is implemented on top of the standard BBBA. The main new addition is the signature mechanism. For the SBBBA, we provide the total number of times when we have $s_\ell \in S_V \cup S_{W'}$ and when we have $s_\ell \in S_0$ in Step 11 of `SBStab` (see Alg. 8). These numbers tell us how successful the signatures are for detecting unnecessary

reductions – instead of doing some reductions we just skip the polynomial. The relation between skipped signatures and the runtime of the algorithm is nonlinear, because the polynomials in the final iteration of the BBA are usually very dense, whence their reduction takes more time than the reduction of the polynomials in the very first iterations. Thus skipping such dense polynomials provides a significant speed-up.

Furthermore, note that the summation of the three rightmost columns in Table 1 does not give the value in the third column, because the choice of the next signature bound in Step 10 of `SBStab` guides the computation in a different way than in the standard version of the Boolean BBA.

Boolean system		Standard BBBA	Signature Based BBBA		
# vars	# gens	# red	# red	# in $S_V \cup S_{W'}$	# in S_0
20	36	3077	1647	183	530
36	67	17243	7748	775	2241
52	99	49809	23989	3996	10235
64	111	80564	41600	11233	20767
68	131	115252	50884	9806	24715
72	119	89325	42209	5172	13675
72	135	124834	57679	14051	28782
84	163	201419	85773	17158	44382
100	195	314388	140337	34696	77360
116	227	450060	235484	78668	151900
132	259	605857	313960	91796	186899

TABLE 1. The number of reductions in the Standard BBBA and the Signature Based BBBA

In Table 2 we compare our algorithm to PolyBoRi, a very good implementation of Buchberger's Algorithm for Boolean polynomials, and to the SAT solver CryptoMiniSat 5. For the Boolean BBA, we have in fact three versions: the standard version (see Algorithm 5), the signature based version (see Algorithm 7), and a version of SBBBA with substitutions. The later version shares the core of the SBBBA. However, when a polynomial of type x_i, x_i+1, x_i+x_j, or x_i+x_j+1 is discovered as a result of some row reduction, we substitute the value of x_i in all polynomials known so far, thereby reducing the complexity of the problem by one variable.

By PolyBoRi we refer to the PolyBoRi implementation of Buchberger's Algorithm called from within Sage 7.5.1 (see [11]). We remark here that PolyBoRi uses a very special implementation of Boolean polynomials (based on ZDDs) and other special techniques for computing Boolean Gröbner bases. Hence, it is a very good candidate for the comparison.

By SAT, we refer to CryptoMiniSat 5 (see [12]). We converted the Boolean polynomials of the input to the DIMACS format by calling the dense ANF to CNF conversion which is a built-in function in Sage.

The table shows that the timings of SBBBA are approaching the speed of PolyBoRi for smaller sized examples. The speedup provided by the signature technique seems to be around a factor of 5. For cryptographic examples, the substitution technique is apparently a significant further improvement. We expect that the next

Boolean system		BBBA	SBBBA	SBBBA+sub	PolyBoRi	SAT
# vars	# gens	in sec	in sec	in sec	in sec	in sec
20	36	0.04	0.03	0.01	0.22	0.01
36	67	0.24	0.30	0.09	0.62	0.02
52	99	4.97	1.70	0.79	0.97	0.03
64	111	27.31	16.97	12.45	1.79	0.06
68	131	27.35	6.79	1.68	1.25	0.06
72	119	73.24	12.20	4.59	1.8	0.06
72	135	129.58	10.96	2.8	2.22	0.06
84	163	87.83	18.19	6.05	1.69	0.07
100	195	282.23	54.06	11.61	2.10	0.07
104	199	$*$	1130.67	383.50	8.22	0.07
116	227	541.12	113.73	81.05	2.45	0.09
132	259	1087.58	252.95	152.87	3.10	0.10
148	291	$*$	473.78	263.10	3.41	0.10

TABLE 2. Comparison of the BBBA and the GBA timings

planned round of optimizations, including the prediction of zero reductions, i.e., the full analogues of Buchberger's criteria, will close the gap even more. Unsurprisingly, SAT solving is much faster than algebraic techniques for the Boolean polynomial setting. Thus, for actual cryptographic attacks, a hybrid method, such as the one suggested in [6], seems to be most promising.

ACKNOWLEDGMENTS

The authors thank Jan Burchard for valuable input about SAT solvers and Michael Brickenstein and Alexander Dreyer for providing us with a better insight into the structure of PolyBoRi. This work was financially supported by the DFG project "Algebraische Fehlerangriffe" [KR 1907/6-1].

REFERENCES

[1] The `ApCoCoA` Team, `ApCoCoA`: Applied Computations in Commutative Algebra, available at `http://apcocoa.uni-passau.de`.

[2] B. Buchberger, Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal, J. Symbolic Comput. **41** (2006), 475-511.

[3] C. Eder and J.-C. Faugère, A survey on signature-based algorithms for computing Gröbner bases, J. Symbolic Comput. **80** (2017), 719-784.

[4] J.-C. Faugère, A new efficient algorithm for computing Gröbner bases without reduction to zero (F5), in: Proc. Conf. ISSAC 2002, ACM Press, New York 2002, pp. 75-83.

[5] M. Gay, J. Burchard, J. Horacek, A.-S. Messeng Ekossono, T. Schubert, B. Becker, M. Kreuzer, and I. Polian, Small scale AES toolbox: Algebraic and propositional formulas, circuit-implementations and fault equations, in: Proc. Conf. Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE 2016), Barcelona, 2016, available at `http://upcommons.upc.edu/handle/2117/99210`

[6] J. Horacek, M. Kreuzer, and A.-S. Messeng Ekossono, Computing Boolean border bases, Proc. Conf. SYNASC'16, Timisoara 2016, IEEE (to appear), available at `www.sc-square.org/CSA/workshop1-papers/paper3.pdf`

[7] A. Kehrein and M. Kreuzer, Computing border bases, J. Pure Appl. Alg. **205** (2006), 279-295.

[8] M. Kreuzer and L. Robbiano, *Computational Commutative Algebra 1*, Springer, Heidelberg 2000.
[9] M. Kreuzer and L. Robbiano, *Computational Commutative Algebra 2*, Springer, Heidelberg 2005.
[10] B. Mourrain, A new criterion for normal form algorithms, in: M. Fossorier, H. Imai, S. Lin, A. Poli (eds.), Proc. Conf. AAECC-13, Honolulu 1999, LNCS **1719**, Springer Verlag, Heidelberg 1999, pp. 440-443.
[11] SageMath, the Sage Mathematics Software System (Version 7.5.1), The Sage Developers, 2017, available at `http://www.sagemath.org`.
[12] M. Soos, The CryptoMiniSat 5 set of solvers at SAT Competition 2016, in: T. Baljo, M.J.H. Heule, and M. Järvisalo (eds.), Proc. SAT COMPETITION 2016, University of Helsinki, Helsinki 2016, p. 28.

Fakultät für Informatik und Mathematik, Universität Passau, D-94030 Passau, Germany
E-mail address: `Jan.Horacek@uni-passau.de`

Fakultät für Informatik und Mathematik, Universität Passau, D-94030 Passau, Germany
E-mail address: `Martin.Kreuzer@uni-passau.de`

Fakultät für Informatik und Mathematik, Universität Passau, D-94030 Passau, Germany
E-mail address: `Ange-Salome.MessengEkossono@uni-passau.de`