Noncommutative Gröbner Basis Public Key Cryptosystems and Their Cryptanalysis

Raiza Cortés
Javier Hernández
Emil Morales
PREMUR 2007

August 6, 2007

Abstract

One example of a widely used modern day public-key cryptosystem is the RSA system, which relies on the difficulty of factoring large integers. In the 1990's Fellows and Koblitz and proposed a different type of public-key cryptosystem, known as"Polly Cracker" systems, based on the difficulty of computing Gröbner bases for certain polynomial ideals. However, it has been found that those systems are susceptible to certain types of attacks and their security is questionable. To counter this vulnerability, more recently, T. Rai (among others) has proposed similar cryptosystems based on Gröbner bases of certain two-sided ideals in the free associative algebra $R = K\langle x_1, \ldots, x_n\rangle$, whose elements are polynomials in noncommuting variables with coefficients in a finite field K. Unlike the commutative ring $K[x_1, \ldots, x_n]$, R has ideals all of whose Gröbner bases are infinite, so computing a full Gröbner basis definitely infeasible. Using the GAP computer algebra system, we implement a prototype of one of these cryptosystems. However, we also show that a certain basic form of Rai's noncommutative Polly Cracker system is susceptible to a different type of attack. If the polynomials constituting the public key are formed in a particular way, we show that it is actually very easy to solve for the coefficients in the polynomial constituting the private key in Rai's system, using a simple and fast commutative Gröbner basis computation.

1 Introduction

Cryptography can be defined as the science concerned with protecting the secrecy of information and the security of communications [1]. An example of a widely used modern day public-key cryptosystem is the RSA system, which relies on the difficulty of factoring large integers. Another type of public-key cryptosystem is a Gröbner basis cryptosystem, whose security depends on the difficulty of computing Gröbner bases. In [1], Rai studies the noncommutative version of these Gröbner basis cryptosystems, as well as the algebraic structures in which they rely. Motivated by the fact that some ideals of noncommutative polynomial rings do not have finite Gröbner basis under any admissible order, we paralleled Rai's work on a noncommutative Gröbner basis cryptosystem.

Essential for our research was the implementation of the cryptosystem using the GAP computational algebra software. As an extension to the GBNP noncommutative algebra package, we programmed a series of tools that the original package lacked, one of these being the noncommutative version of the division algorithm using the length-lex term order. Once we had this set of tools, we proceeded to study the weaknesses of the cryptosystem proposed by Rai. We found that the main weakness of the cryptsystem was that reduction of the encrypted message by the public key usually yielded the original message. We propose a technique that prevents this from happening. Finally, we show that Rai's cryptosystem is susceptible to an attack via the public key polynomials, and we describe the general procedure of the attack.

2 Noncommutative Gröbner Basis Theory

Let $\Sigma = \{x_1, \ldots, x_n\}$ be a finite alphabet, and let Σ^* be the set of finite words generated by Σ. The elements of Σ^* are of the form $u = x_{i_1} \ldots x_{i_s}$, where $i_1, \ldots, i_s \in \{1, \ldots, n\}$. Let λ be the empty word and let multiplication be given by concatenation. Note that this product is not commutative. For a finite field K, we define the free associative algebra in n non-commuting variables over a finite field K, $R = K\langle x_1, \ldots x_n\rangle$, to be the set of formal sums $\sum_{i=1}^{t} c_i u_i$, with $c_i \in K$ and $u_i \in \Sigma^*$, along with polynomial addition and multiplication.

An **admissible term order** $>$ on Σ^* is a total order such that:

1. If $u < v$, then $xuy < xvy$ $\forall u, v, x, y \in \Sigma^*$
2. $>$ is a well-ordering.

Examples of admissible term orders are the length-lexicographic and the total lexicographic order.

Given a term order $>$ and $f = \sum_{i=1}^{t} c_i u_i \in K\langle x_1, \ldots, x_n \rangle$, with $c_i \in K \setminus \{0\}$, we define the **leading term** of f, denoted $LT(f)$, to be u_i ocurring in f such that $u_i \geq u_j \; \forall u_j$ ocurring in f. We define the **leading coefficient** of f, denoted $LC(f)$, to be the coefficient of $LT(f)$.

For $X \subseteq K\langle x_1, \ldots, x_n \rangle$, we write

$$LT(X) = \{u \in \Sigma^* : u = LT(f) \text{ for some } f \in X\},$$

and $NonLT(X) = \Sigma^* - LT(X)$.

Let $R = K\langle x_1, \ldots, x_n \rangle$ be the free associative algebra in n non-commuting variables over a finite field K. A subset I of $K\langle x_1, \ldots, x_n \rangle$ that is closed under polynomial addition and satisfies $fgh \in I \; \forall g \in I$ and $\forall f, h \in R$ is said to be a **two-sided ideal** of R. If X is a subset of R, then $\langle X \rangle$ denotes the ideal generated by X, i.e. the smallest ideal of R that contains X.

We arrive to the main definition:

Definition 1 *Let $>$ be an admissible order on $R = K\langle x_1, \ldots, x_n \rangle$, and let I be a two-sided ideal of R. A subset G of I is a **Gröbner basis** for I with respect to $>$ if $\langle LT(G) \rangle = \langle LT(I) \rangle$.*

Theorem 1 (Division Algorithm, [1], 1.3.3) *Given $g \in K\langle x_1, \ldots, x_n \rangle$ and an ordered subset $F = \{f_1, \ldots, f_s\}$ of $K\langle x_1, \ldots, x_n \rangle$, we can find non-negative integers $t_1, \ldots, t_s$, and elements $u_{ij}, v_{ij}, r \in K\langle x_1, \ldots, x_n \rangle$, for $1 \leq i \leq s$, $1 \leq j \leq t_i$, such that:*

1. $g = \sum_{i=1}^{s} \sum_{j=1}^{t_i} u_{ij} f_i v_{ij} + r$,
2. $LT(g) \geq LT(u_{ij} f_i v_{ij}) \; \forall i, j$,
3. *$LT(f_i)$ does not divide any monomial that occurs in r for $1 \leq i \leq s$.*

This r is the remainder of the division; we say that g reduces to r modulo F, and we denote it by $g \xrightarrow{F} r$.

We now present the definition of overlap relations, which are analogous to the S-polynomials of the commutative setting.

Definition 2 *Let $f, g \in K\langle x_1, \ldots, x_n \rangle$, and suppose that $b, c \in \Sigma^*$ are such that:*

1. $LT(f) \cdot c = b \cdot LT(g)$
2. $LT(f) \nmid b, LT(g) \nmid c$

*Then the **overlap relation** of f and g by b, c is*

$$O(f,g,b,c) = \frac{1}{LC(f)} \cdot f \cdot c - \frac{1}{LC(g)} \cdot b \cdot g.$$

The following result is important for deciding whether a subset G of an ideal is a Gröbner basis or not.

Theorem 2 ([1], Theorem 1.3.4.5) *Let $K\langle x_1, \ldots, x_n\rangle$ be the free algebra over a finite field K in n non-commuting variables. Let Σ^* be the set of finite noncommutative words generated by $\Sigma = \{x_1, \ldots, x_n\}$, and let $>$ be an admissible order on Σ^*. Suppose G is a set of elements of $K\langle x_1, \ldots, x_n\rangle$ such that for distinct elements $f, g \in G$, $LT(f) \nmid LT(g)$, and every overlap relation $O(g_1, g_2, b, c)$ reduces to zero modulo G for every pair $g_1, g_2 \in G$. Then G is a Gröbner basis for $\langle G\rangle$.*

Finally, given an admissible term order and a set of generators of an ideal I, a noncommutative version of Buchberger's algorithm can be defined and used to construct Gröbner bases.

As can be seen, noncommutative Gröbner basis theory is analogous to commutative Gröbner basis theory and yet, we can find some differences between the two of them. The most interesting difference is, perhaps, the fact that whereas in the commutative case every Gröbner basis of an ideal is finite, in the noncommutative case we find that some ideals do not have finite Gröbner basis under some monomial orders. Moreover, there are ideals that do not have finite Gröbner basis under any admissible order. An example of such an ideal is given next.

Theorem 3 ([1], Proposition 3.4.1) *Let $K\langle x, y, z\rangle$ be the noncommutative free associative algebra over a finite field K. Let $g_1 = xzy + yz$, $g_2 = yzx + zy \in K\langle x, y, z\rangle$. Then $I = \langle g_1, g_2\rangle$ does not have a finite Gröbner basis under any admissible order.*

We end this section with the following corollary of Theorem 3:

Corollary 1 ([1], Corollary 3.4.2) *Let $K\langle x_1, \ldots, x_n\rangle$ be the noncommutative free associative algebra over a finite field K in n variables with $n \geq 5$. Let $A = \prod_{i=1}^{n} x_i$, $B = x_1(\prod_{i=2}^{n-1} \rho(x_i))x_n$ and $C = x_1(\prod_{i=2}^{n-1} \sigma(x_i))x_n$, where ρ and σ are nontrivial permutations of $\{x_2, \ldots x_{n-1}\}$. Let $g_1 = ACB + BC$ and $g_2 = BCA + CB$. Then $I = \langle g_1, g_2\rangle$ does not have a finite Gröbner basis under any admissible order.*

3 Noncommutative Gröbner basis Public Key Cryptosystems

A noncommutative Gröbner basis public-key cryptosystem consists of the following:

- *Private Key:* A Gröbner Basis, $G = \{g_1, \ldots, g_t\}$ for a two-sided ideal I of a noncommutative free associative algebra $K\langle x_1, \ldots, x_n\rangle$ over a finite field K, with respect to some monomial order $>$.
- *Public Key:* A set, $Q = \{q_1, \ldots, q_s\}$, where $q_r = \sum_{i=1}^{t} \sum_{j=1}^{d_{r_i}} f_{r_ij} g_i h_{r_ij}$, chosen so that computing a Gröbner basis of $\langle Q\rangle$ is infeasible.
- *Message Space:* $M = NonLT(I)$ or a subset of $NonLT(I)$.
- *Encryption:* $c = p + m$, where $p = \sum_{i=1}^{s} \sum_{j=1}^{k_{ir}} F_{rij} q_i H_{rij}$ and $m \in M$ is a message.
- *Decryption:* Reduction of c modulo G yields the message, m.

The advantage of the noncommutative Gröbner basis public-key cryptosystem over the commutative version is that Q can chosen so that $\langle Q\rangle$ does not have a finite Gröbner basis under any admissible order. In [1], Rai proposes a cryptosystem that has this property. This cryptosystem is based on Theorem 3, and we will devote the rest of the paper to analyzing it.

Construction 1 *To construct a public-key cryptosystem based on Theorem 3, pick a polynomial $g \in K\langle x, y, z\rangle$, whose leading term has no self-overlaps. This ensures that $G = \{g\}$ is a Gröbner basis for $\langle g\rangle$. Set $\{g\}$ as the private key. Next, suppose that $f, h \in K\langle x, y, z\rangle$ are such that $LT(f) \cdot LT(g) \cdot LT(h)$ and $LT(h) \cdot LT(g) \cdot LT(f)$ have no self-overlaps. Let $q_1 = fgh + hg$ and $q_2 = hgf + gh$. Set $\{q_1, q_2\}$ as the public key. The message space consists of elements of $NonLT(\langle G\rangle)$ and varies depending on the private key used. To encrypt a message m, choose random polynomials F_1, F_2, H_1 and H_2, and let $p = F_1q_1H_1 + F_2q_2H_2$. Then $c = p + m$ is the encrypted message. Decryption is achieved by reducing c modulo $G = \{g\}$.*

Next, we discuss some weaknesses of a cryptosystem based on Construction 1, as well as ways that help overcome them. The first of these weaknesses is the fact that, in practice, if F_1, F_2, H_1 and H_2 are arbitrary, $c = F_1q_1H_1 + F_2q_2H_2 + m$ can be reduced correctly by using the public key $Q = \{q_1, q_2\}$, thusly revealing the message m. In [1], Rai proposes some techniques that prevent this from happening. Using the GAP computational algebra software, we ran some examples and developed the following technique:

Technique 1 *Given a monomial order $>$, choose monomials A_1, A_2, B_1 and B_2 such that*

$$A_1 \cdot LT(q_1) \cdot B_1 = -A_2 \cdot LT(q_2) \cdot B_2$$

Set F_1, F_2, H_1 and H_2 as:

$$\begin{aligned} F_1 &= A_1 + \text{ random smaller monomials} \\ F_2 &= A_2 + \text{ random smaller monomials} \\ H_1 &= B_1 + \text{ random smaller monomials} \\ H_2 &= B_2 + \text{ random smaller monomials} \end{aligned}$$

This technique ensures that $c = F_1 q_1 H_1 + F_2 q_2 H_2 + m$ does not yield m when reduced by $\{q_1, q_2\}$.

Example 1. If we let $f = x - 5328, g = z - 6426$ and $h = y - 878$, then we have

$$\begin{aligned} q_1 = \quad & fgh + hg = xzy - 5328zy - 6426xy - 878xz + yz + 5642028x \\ & \qquad +4677106y + 34231302z - 30055083156 \\ q_2 = \quad & hgf + gh = yzx - 6426zx - 878yx - 5328yz + zy + 5642028x \\ & \qquad +34231302z + 4677106y - 30055083156. \end{aligned}$$

Let m be the message. Let $>$ be length-lex. Use the technique described above to find polynomials F_1, H_1, F_2 and H_2:

$$\begin{aligned} F_1 &= zyxzxy + xyz + zyz + xyx + yxxy + xy + yz \\ F_2 &= -zyxzxyxz + zzx + xyx + zx + xz \\ H_1 &= zxxyyzzx + zzy + yzy + yxy + zyx + yx + zy \\ H_2 &= xyyzzx + zyz + yzz + yxy + yx + zx \end{aligned}$$

The encrypted message is $c = F_1 q_1 H_1 + F_2 q_2 H_2 + m$, which does not yield m when reduced by $\{q_1, q_2\}$

A second weakness of the cryptosystem is the fact that the encrypted message can be reduced to m if the attacker is able to compute a partial

Gröbner basis for the public key. This problem can be overcome if the number of variables in f, g and h is increased, making the computation of a partial Gröbner basis for $\langle q_1, q_2 \rangle$ harder.

A third weakness of the cryptosystem is that it is susceptible to a trilinear algebra kind of attack. The following section will be dedicated to discussing this weakness.

4 An attack to the cryptosystem

We now turn our attention to the possibility of a trilinear algebra attack to a cryptosystem based on Theorem 3. Before describing the attack, we need the following definition. Let $\Sigma = \{x_1, \ldots, x_s\}$, and let $x_0 = 1$. For a word $x_{i_1} \ldots x_{i_r}$, with $x_{i_j} \in \{x_1, \ldots x_s\}$, we define a **subword** as $x_{\epsilon_1 i_1} \ldots x_{\epsilon_r i_r}$, where $\epsilon_j \in \{0, 1\}$.

Having this, the procedure of the attack is as follows. For a cryptosystem as in Construction 1, let f be a linear combination of all the subwords of $x_1 \ldots x_l$ with coefficients $a_1, \ldots, a_{2^l}$, where $a_1 \neq 0$, let g be a linear combination of all the subwords of $x_{l+1} \ldots x_{l+m}$ with coefficients $b_1, \ldots, b_{2^m}$, where $b_1 \neq 0$ and let h be a linear combination of all the subwords of $x_{l+m+1} \ldots x_{l+m+n}$ with coefficients $c_1, \ldots, c_{2^n}$, where $c_1 \neq 0$. Order the monomials of f so that a_1 is the coefficient of $x_1 \ldots x_l$ and a_l is the constant term. Proceed analogously with g and h.

We assume that by looking at the polynomials q_1 and q_2 of the public key, the attacker can know that these are of the form $q_1 = fgh + hg$ and $q_2 = hgf + gh$, and therefore can know what monomials occur in f, g and h. Let u_{ijk} denote the coefficient in q_1 of the word formed by multiplying the subwords corresponding to a_i, b_j and c_k. If $i = 0$, then u_{ijk} denotes the coefficient in q_1 of the word formed by multiplying the subwords corresponding to b_j and c_k. The attacker then uses the u_{ijk} to set up a system of polynomials in the variables $a_1, \ldots, a_{2^l}, b_1, \ldots, b_{2^m}, c_1, \ldots, c_{2^n}$. These polynomials are of the form:

$$
\begin{aligned}
&a_i b_j c_k - u_{ijk}, \forall\ i = 1, \ldots, 2^l - 1, j = 1, \ldots, 2^m, k = 1, \ldots, 2^n \\
&a_{2^l} b_j c_k - u_{2^l jk}, \forall\ j = 1, \ldots, 2^m - 1, k = 1, \ldots, 2^n - 1 \\
&a_{2^l} b_{2^m} c_k + b_{2^m} c_k - u_{2^l 2^m k}, \forall\ k = 1, \ldots, 2^n \\
&a_{2^l} b_j c_{2^n} + b_j c_{2^n} - u_{2^l j 2^n}, \forall\ j = 1, \ldots, 2^m - 1 \\
&b_j c_k - u_{0jk}, \forall\ j = 1, \ldots, 2^m - 1, k = 1, \ldots, 2^n - 1.
\end{aligned}
$$

Setting these polynomials equal to zero, we get a system of trilinear cubic equations in the a_i, b_j, c_k which, when solved, gives the attacker possession of the private key. Although there is no known general method for solving a system of cubic equations, we will show that this specific system can be easily solved by computing its Gröbner basis, which is given by the following theorem.

Theorem 4 *Let E denote the above set of polynomials, and let $>$ be a monomial order with $c_{2^n} > \cdots > c_1 > b_{2^m} > \cdots > b_1 > a_{2^l} > \cdots > a_1$. Let G denote the following set of polynomials:*

$$\begin{aligned}&u_{111}c_2 - u_{112}c_1, \ldots, u_{111}c_{2^n} - u_{112^n}c_1,\\ &u_{111}b_2 - u_{121}b_1, \ldots, u_{111}b_{2^m} - u_{12^m1}b_1,\\ &b_1c_1 - u_{011},\\ &u_{011}a_1 - u_{111}, \ldots, u_{011}a_{2^l} - u_{2^l11}.\end{aligned}$$

Then G is a Gröbner basis for $\langle E\rangle$ with respect to $>$.

Proof. First, we will show that $\langle G\rangle = \langle E\rangle$, by showing that $\langle G\rangle \subseteq \langle E\rangle$ and that $\langle E\rangle \subseteq \langle G\rangle$. To see that $\langle G\rangle \subseteq \langle E\rangle$, note that the elements of G are either elements of E or are obtained by computing the S-polynomials of certain pairs of elements of E:

$$\begin{aligned}S(a_1b_1c_1 - u_{111}, a_1b_1c_2 - u_{112}) &= u_{111}c_2 - u_{112}c_1\\ S(a_1b_1c_1 - u_{111}, a_1b_1c_3 - u_{113}) &= u_{111}c_3 - u_{113}c_1\\ &\vdots\\ S(a_1b_1c_1 - u_{111}, a_1b_1c_{2^n} - u_{112^n}) &= u_{111}c_{2^n} - u_{112^n}c_1\\ S(a_1b_1c_1 - u_{111}, a_1b_2c_1 - u_{121}) &= u_{111}b_2 - u_{121}b_1\\ S(a_1b_1c_1 - u_{111}, a_1b_3c_1 - u_{131}) &= u_{111}b_3 - u_{131}b_1\\ &\vdots\\ S(a_1b_1c_1 - u_{111}, a_1b_{2^m}c_1 - u_{12^m1}) &= u_{111}b_{2^m} - u_{12^m1}b_1\\ S(a_1b_1c_1 - u_{111}, b_1c_1 - u_{011}) &= u_{011}a_1 - u_{111}\\ &\vdots\\ S(a_{2^l}b_1c_1 - u_{2^l11}, b_1c_1 - u_{011}) &= u_{011}a_{2^l} - u_{2^l11}.\end{aligned}$$

To show that $\langle E\rangle \subseteq \langle G\rangle$, we will show that $\forall p \in E$, $p \xrightarrow{G} 0$. If p is of the form $p = a_ib_jc_k - u_{ijk}\ \forall\ i = 1, \ldots, 2^l - 1, j = 1, \ldots, 2^m, k = 1, \ldots, 2^n$, then

$$p = \quad (u_{111}c_k - u_{11k}c_1)\left(\frac{a_ib_j}{u_{111}}\right) + (u_{111}b_j - u_{1j1}b_1)\left(\frac{u_{11k}a_ic_1}{u_{111}^2}\right)$$

$$+(b_1c_1 - u_{011})\left(\frac{u_{1j1}u_{11k}a_i}{u_{111}^2}\right) + (u_{011}a_i - u_{i11})\left(\frac{u_{1j1}u_{11k}}{u_{111}^2}\right)$$
$$+\left(\frac{u_{i11}u_{1j1}u_{11k}}{u_{111}^2} - u_{ijk}\right),$$

which is equivalent to saying that $p \xrightarrow{G} \left(\frac{u_{i11}u_{1j1}u_{11k}}{u_{111}^2} - u_{ijk}\right)$. Taking a look at a subset of the S-polynomials of E, namely:

$$\begin{aligned}
&u_{011}a_1 - u_{111}, \ldots, u_{02^m2^n}a_1 - u_{12^m2^n}\\
&\vdots\\
&u_{011}a_{2^l-1} - u_{2^l-1,1,1}, \ldots, u_{02^m2^n}a_{2^l-1} - u_{2^l-1,2^m,2^n}\\
&u_{011}b_2 - u_{021}b_1, \ldots, u_{011}b_{2^m-1} - u_{0,2^m-1,1}b_1\\
&\vdots\\
&u_{0,1,2^n-1}b_2 - u_{0,2,2^n-1}b_1, \ldots, u_{0,1,2^n-1}b_{2^m-1} - u_{0,2^m-1,2^n-1}b_1,
\end{aligned}$$

we see that $\frac{u_{ijk}}{u_{0jk}} = \frac{u_{ij'k'}}{u_{0j'k'}}$ and $\frac{u_{0jk}}{u_{01k}} = \frac{u_{0jk'}}{u_{01k'}}$ $\forall\ i = 1, \ldots, 2^l, j = 1, \ldots, 2^m - 1$, and $k = 1, \ldots, 2^n - 1$. Hence, we have that

$$\begin{aligned}
u_{i11}u_{1j1}u_{11k} - u_{ijk}u_{111}^2 &= \left(u_{i11}u_{1j1}u_{11k} - \left(\frac{u_{i11}u_{0jk}}{u_{011}}\right)u_{111}^2\right)\\
&= u_{i11}\left(u_{1j1}u_{11k} - \left(\frac{u_{0jk}}{u_{011}}\right)u_{111}^2\right)\\
&= u_{i11}\left(u_{1j1}u_{11k} - \left(\frac{u_{0jk}}{u_{011}}\right)\left(\frac{u_{1j1}u_{011}}{u_{0j1}}\right)u_{111}\right)\\
&= u_{i11}u_{1j1}\left(u_{11k} - \left(\frac{u_{0jk}}{u_{0j1}}\right)u_{111}\right)\\
&= u_{i11}u_{1j1}\left(u_{11k} - \left(\frac{u_{0jk}}{u_{0j1}}\right)\left(\frac{u_{11k}u_{011}}{u_{01k}}\right)\right)\\
&= u_{i11}u_{1j1}u_{11k}\left(1 - \left(\frac{u_{0jk}}{u_{0j1}}\right)\left(\frac{u_{011}}{u_{01k}}\right)\right)\\
&= 0.
\end{aligned}$$

Here, we have used the fact that $\frac{u_{i11}}{u_{011}} = \frac{u_{ijk}}{u_{0jk}}$, $\frac{u_{1j1}}{u_{0j1}} = \frac{u_{111}}{u_{011}}$, $\frac{u_{11k}}{u_{01k}} = \frac{u_{111}}{u_{011}}$ and $\frac{u_{0j1}}{u_{011}} = \frac{u_{0jk}}{u_{01k}}$. From this it follows that $\left(\frac{u_{i11}u_{1j1}u_{11k}}{u_{111}^2} - u_{ijk}\right) = 0$. Thus, if $p = a_ib_jc_k - u_{i,j,k}$, then $p \xrightarrow{G} 0$. For the cases where $p = b_jc_k - u_{0jk}$, $p = a_{2^l}b_jc_{2^n} + b_jc_{2^n} - u_{2^lj2^n}$ and $p = a_{2^l}b_2^mc_k + b_2^mc_k - u_{2^l2^mk}$, we can show that $p \xrightarrow{G} 0$ in an analogous way. Therefore, $\langle E\rangle \subseteq \langle G\rangle$ and $\langle E\rangle = \langle G\rangle$.

Finally, consider the following lemma:

Lemma 1 ([2], Proposition 2.4) *Given a finite set $G \subset K[x_1, \dots, x_n]$, suppose that we have $f, g \in G$ such that $LT(f)$ and $LT(g)$ are relatively prime. Then $S(f, g) \xrightarrow{G} 0$.*

It is clear that for every pair $g_1, g_2 \in G$, $LT(g_1)$ and $LT(g_2)$ are relatively prime. By Lemma 1 and Buchberger's criterion, it follows that G is a Gröbner basis for $\langle G \rangle$, and thus, of $\langle E \rangle$. This concludes our proof. $\square$

Setting the polynomials of G equal to zero, we get a system of equations that yields values for unique values for the a_i's, and values for the b_j's and c_k's up to a constant factor. Thus, the attacker obtains g up to a constant factor. This is enough to retrieve the message m from c.

Example 2. Let q_1 in the public key be

$$
\begin{aligned}
q_1 = {} & 49333500510x_1x_2x_3x_4x_5x_6 + 34106767170x_2x_3x_4x_5x_6 \\
& +25879869120x_1x_3x_4x_5x_6 + 14446688865x + x_2x_4x_5x_6 \\
& +68707446990x_1x_2x_3x_5x_6 + 45414323466x_1x_2x_3x_4x_6 \\
& +10519896276x_1x_2x_3x_4x_5 + 9295930x_5x_6x_3x_4 \\
& +75324920790x_3x_4x_5x_6 + 9987733455x_2x_4x_5x_6 \\
& +47500965330x_2x_3x_5x_6 + 31397240022x_2x_3x_4x_6 \\
& +7272941292x_2x_3x_4x_5 + 7578590880x_1x_4x_5x_6 \\
& +36043250880x_1x_3x_5x_6 + 23823907392x_1x_3x_4x_6 \\
& +5518634112x_1x_3x_4x_5 + 4919642070x_1x_2x_5x_6 \\
& +13299007659x_1x_2x_4x_6 + 3080617974x_1x_2x_4x_5 \\
& +63249155034x_1x_2x_3x_6 + 14651204724x_1x_2x_3x_5 \\
& +212288756550x_1x_2x_3x_4 + 8557438x_6x_3x_4 + 2722195x_5x_6x_4 \\
& +12946570x_5x_6x_3 + 1982268x_5x_3x_4 + 22057946085x_4x_5x_6 \\
& +104906056710x_3x_5x_6 + 69340920114x_3x_4x_6 \\
& +16062317604x_3x_4x_5 + 3401199690x_2x_5x_6 + 9194282853x_2x_4x_6 \\
& +2129788458x_2x_4x_5 + 43727369478x_2x_3x_6 + 10129125708x_2x_3x_5 \\
& +146766053850x_2x_3x_4 + 2580795840x_1x_5x_6 + 6976528608x_1x_4x_6 \\
& +1616061888x_1x_4x_5 + 33179884608x_1x_3x_6 + 7685877888x_1x_3x_5 \\
& +111364593600x_1x_3x_4 + 4528813362x_1x_2x_6 + 1049066532x_1x_2x_5 \\
& +62166065325x_1x_2x_4 + 295657480950x_1x_2x_3 + 2505937x_6x_4
\end{aligned}
$$

$$
\begin{aligned}
&+11918062x_6x_3 + 7512489040x_5x_6 + 580482x_5x_4 + 2760732x_5x_3\\
&+20305607511x_4x_6 + 4703645646x_4x_5 + 96572056386x_3x_6\\
&+22370211396x_3x_5 + 324173371600x_3x_4 + 3130999854x_2x_6\\
&+725273244x_2x_5 + 42978574275x_2x_4 + 204403108650x_2x_3\\
&+2375770944x_1x_6 + 550329984x_1x_5 + 32611706400x_1x_4\\
&+155099006400x_1x_3 + 21169888350x_1x_2 + 6915678064x_6\\
&+1601966304x_5 + 94930053400x_4 + 451480728400x_3\\
&+14635824450x_2 + 11105515200x_1 + 32327261200.
\end{aligned}
$$

We assume that by looking at q_1, the attacker knows that q_1 is of the form $q_1 = fgh + hg$ and that f, g and h are of the form

$$
\begin{aligned}
f &= a_1x_1x_2 + a_2x_1 + a_3x_2 + a_4\\
g &= b_1x_3x_4 + b_2x_3 + b_3x_4 + b_4\\
h &= c_1x_5x_6 + c_2x_3 + c_3x_4 + c_4,
\end{aligned}
$$

where the a_i's, b_j's and c_k's are constants.

Then the attacker treats $a_1, \ldots, c_4$ as variables and uses the coefficients of q_1 to set up a system of polynomials as described above. For example, since $a_1x_1x_2 \cdot b_1x_3x_4 \cdot c_1x_5x_6 = a_1b_1c_1x_1x_2x_3x_4x_5x_6$, and $49333500510x_1x_2x_3x_4x_5x_6$ occurs in q_1, setting the coefficients equal to each other yields the equation $a_1b_1c_1 - 49333500510 = 0$. The polynomials on the left-hand side of the resulting equations conform the following list:

$$
\begin{array}{ll}
a_1b_1c_1 - 49333500510 & a_3b_1c_1 - 34106767170 \\
a_2b_1c_1 - 25879869120 & a_1b_3c_1 - 14446688865 \\
a_1b_2c_1 - 68707446990 & a_1b_1c_3 - 45414323466 \\
a_1b_1c_2 - 10519896276 & a_4b_1c_1 - 75324920790 \\
a_3b_3c_1 - 9987733455 & a_3b_2c_1 - 47500965330 \\
a_3b_1c_3 - 31397240022 & a_3b_1c_2 - 7272941292 \\
a_2b_3c_1 - 7578590880 & a_2b_2c_1 - 36043250880 \\
a_2b_1c_3 - 23823907392 & a_2b_1c_2 - 5518634112 \\
a_1b_4c_1 - 4919642070 & a_1b_3c_3 - 13299007659 \\
a_1b_3c_2 - 3080617974 & a_1b_2c_3 - 63249155034 \\
a_1b_2c_2 - 14651204724 & a_1b_1c_4 - 212288756550 \\
a_4b_3c_1 - 22057946085 & a_4b_2c_1 - 104906056710 \\
a_4b_1c_3 - 69340920114 & a_4b_1c_2 - 16062317604 \\
a_3b_4c_1 - 3401199690 & a_3b_3c_3 - 9194282853 \\
a_3b_3c_2 - 2129788458 & a_3b_2c_3 - 43727369478 \\
a_3b_2c_2 - 10129125708 & a_3b_1c_4 - 146766053850 \\
a_2b_4c_1 - 2580795840 & a_2b_3c_3 - 6976528608 \\
a_2b_3c_2 - 1616061888 & a_2b_2c_3 - 33179884608 \\
a_2b_2c_2 - 7685877888 & a_2b_1c_4 - 111364593600 \\
a_1b_4c_3 - 4528813362 & a_1b_4c_2 - 1049066532 \\
a_1b_3c_4 - 62166065325 & a_1b_2c_4 - 295657480950 \\
c_3b_1 - 8557438 & c_2b_1 - 1982268 \\
c_1b_3 - 2722195 & c_1b_2 - 12946570 \\
c_1b_1 - 9295930 & c_3b_3 - 2505937 \\
c_3b_2 - 11918062 & c_2b_3 - 580482 \\
c_2b_2 - 2760732 & a_4b_3c_3 - 20305607511 \\
a_4b_3c_2 - 4703645646 & a_4b_2c_3 - 96572056386 \\
a_4b_2c_2 - 22370211396 & a_3b_4c_3 - 3130999854 \\
a_3b_4c_2 - 725273244 & a_3b_3c_4 - 42978574275 \\
a_3b_2c_4 - 204403108650 & a_2b_4c_3 - 2375770944 \\
a_2b_4c_2 - 550329984 & a_2b_3c_4 - 32611706400 \\
a_2b_2c_4 - 155099006400 & a_1b_4c_4 - 21169888350 \\
a_3b_4c_4 - 14635824450 & a_2b_4c_4 - 11105515200 \\
a_4b_4c_1 + c_1b_4 - 7512489040 & a_4b_4c_3 + c_3b_4 - 6915678064 \\
a_4b_1c_4 + c_4b_1 - 324173371600 & a_4b_4c_4 + c_4b_4 - 32327261200 \\
a_4b_4c_2 + c_2b_4 - 1601966304 & a_4b_2c_4 + c_4b_2 - 451480728400 \\
a_4b_3c_4 + c_4b_3 - 94930053400. &
\end{array}
$$

Let $>$ be a monomial order with

$$c_1 > \ldots > c_4 > b_1 > \ldots > b_4 > a_1 > \ldots > a_4.$$

The attacker then computes a Gröbner Basis with respect to $>$ for the above set of polynomials and gets:

$$1235c_1 - 287c_4 \tag{1}$$
$$6175c_2 - 306c_4 \tag{2}$$
$$6175c_3 - 1321c_4 \tag{3}$$
$$b_4c_4 - 3989050 \tag{4}$$
$$323b_1 - 3239b_4 \tag{5}$$
$$323b_2 - 4511b_4 \tag{6}$$
$$646b_3 - 1897b_4 \tag{7}$$
$$a_1 - 5307 \tag{8}$$
$$a_2 - 2784 \tag{9}$$
$$a_3 - 3669 \tag{10}$$
$$a_4 - 8103. \tag{11}$$

Setting these polynomials equal to zero gives a system of equations. Solving (8) through (11) yields unique values for the a_i's. Letting $b_4 = d$, d some constant, and then solving equations (1) through (7) gives values for the b_j's and c_k's up to a constant factor. Thus, the attacker knows g up to a constant factor, and can reduce c and obtain m.

If f, g and h are allowed to have share variables, for example, if f, g and h are of the form described in Corollary 1, then the attack, after a few variations, still holds. Similarly, increasing the number of variables in f, g and h does not give any additional security.

A better attempt at countering this attack is, perhaps, to use the following result, which follows from Theorem 3 and provides us with a way to generate polynomials q_1 and q_2 that can be used for the cryptosystem:

Corollary 2 *Let $K\langle x_1, \ldots, x_n\rangle$ be the free associative algebra in n non-commuting indeterminates over a finite field K. Let $q_1 = x_1A + B$ and $q_2 = Bx_1 + A$, where $A = x_2 \ldots x_n$ and $B = \prod_{i=2}^{n} \rho(x_i)$, and ρ is a permutation of $\{x_2, \ldots, x_n\}$. Suppose that A and B can be expressed as $A = CD$ and $B = D'C'$, where $C = x_2 \ldots x_r$, $D = x_{r+1} \ldots x_n$, $C' = \prod_{i=2}^{r} \sigma_1(x_i)$, and $D' = \prod_{i=r+1}^{n} \sigma_2(x_i)$, and σ_1, σ_2 are permutations of $\{x_2, \ldots, x_r\}$ and $\{x_{r+1}, \ldots, x_n\}$ respectively. Then the ideal generated by q_1 and q_2 does not have a finite Gröbner basis under any admissible order.*

Given polynomials q_1, q_2 obtained by using Corollary 2, the attacker is slowed down, for he does not know the specific way in which q_1, q_2 were constructed. However, the attacker can still perform the attack in a reasonably short amount of time.

5 Conclusion

A cryptosystem based on Theorem 3 was the base for our work, which consisted of three parts: the implementation of the cryptosystem using the GAP computational algebra software, the search for weaknesses of the cryptosystem, and the formulation of techniques to overcome such weaknesses. As a result for the implementation part of the project, we programmed an extension for the GBNP noncommutative algebra package. The extension consists, mainly, of an encryption and decryption algorythm.

We then checked the cryptosystem for possible weaknesses, and found that reduction of the encrypted message by the public key usually yielded the original message. We provided a technique that prevents this from happening. A second weakness of the cryptosystem, reduction of the encrypted message by a partial Gröbner basis, was countered by increasing the number of variables used in f, g and h, which makes the computation of a partial Gröbner basis harder.

Finally, we found that the cryptosystem is susceptible to a trilinear algebra attack. This is so because the public key is constructed in a very specific way. We were not able to find a way to counter this attack. Thus, we conclude that a cryptosystem based on Theorem 3 is insecure.

This does not mean that noncommutative Gröbner basis cryptosystems in general are insecure. As more classes of ideals that do not have finite Gröbner basis under any admissible order are discovered, the possibilities of an attack along the lines discussed here may be exhausted. Yet, unless a general way to generate ideals that do not have finite Gröbner basis under any admissible order is discovered, a noncommutative Gröbner basis cryptosystem remains highly unrecommended.

6 Appendix

Complete code for division algorithm, functions that compute the leading monomial, leading term and leading coefficient, and other functions needed to encrypt and decrypt a message for a cryptosystem based on Theorem 3.

```
# This method (ifPolys)is used to test whether or not the parameters
```

```
# that go in the functions are actual polynomials. A polynomial
# is represented as a list of two lists, the first list
# (list[1])contains the monomials (lists of the form
# [var1,...,Varn] where the different 'vars' are the variables
# concatenated as a monomial.
# Input: poly
#    * poly: is a non-commutative polynomial.
# Output: return false if the function is a polynomial and
returns true if is not.

ifPolys:= function(poly)
  if Length(poly) = 2 and Length(poly[1]) = Length(poly[2]) then
      return false;
  else
      return true;
  fi;
end;;

#This function get the leading monomial of the noncommutative
# polynomial. Takes as paramater in the form [[monomials],
# [coefficients]].
# Makes a copy of [monomials] and compares the length of the
# monomials and takes out of the list of the lesser one. To
# break ties, compares the i'th variable simultaniously until
# it finds one pair with different preference and takes out.
# At the end the list will only contain the leading monomial
# wich is the one biger in length or with more preference.
# Input: poly
# * poly : Polynomial in NP form (look in GBNP documentation
# for NP form). Output: the leading term of poly, a monomial.                                     # How it
using length-lex order.

LM := function(poly)
  local monlist,i,finish,mon,polym;

  polym := ShallowCopy(poly);
  if polym = [[],[]] then #case of zero polynomial
     Error("Monomio indefinido!!!LM");

  else
     monlist := ShallowCopy(poly[1]);

  while Length(monlist)> 1 do
     if Length(monlist[1]) > Length(monlist[2]) then
     Remove(monlist,2);
   elif Length(monlist[2]) > Length(monlist[1]) then
     Remove(monlist,1);
   elif Length(monlist[1]) = Length(monlist[2]) then
     finish:=false;
```

```
     i:=1;
            while finish=false do
             if monlist[1][i] > monlist[2][i] then
              finish:=true;
              Remove(monlist,1);
             elif monlist[2][i] > monlist[1][i] then n
              finish:=true;
              Remove(monlist,2);
             else
                i := i+1;
               fi;
             od;
   fi;
  od;
   return [monlist,[1]];
  fi;
end;;

# This function is use to get the leading coefficient of a
# noncommutative polynomial. Uses LM() that looks for the
# leading term using Length-lex order.

# Input: poly
#   *poly: the noncommutative polynomial in the NP form.
# Output: the leading coefficient in the form of an integer.

LC:=function(poly)
  local mon1, indice,polyc;
    if ifPolys(poly) then
       Error("A ifDiv estan entrando cosas raras!!!!!!\n");
    else
       polyc := ShallowCopy(poly);
       mon1:= ShallowCopy(LM(polyc)[1][1]);
       indice:=Position(polyc[1], mon1);
       return polyc[2][indice];
    fi;
end;;

# This function return the leading term of the
# noncommutative polynomial. Looks for the leading
# monomial (using length-lex ordering) and the LC(),
# and returns a NP polynomial with one
# term(the leading term).

# Input: poly
#   *poly: the noncommutative polynomial in the form
# of a list.
# Output: the leading term in the form of a list.
```

```
LTer:=function(poly)
  local ret;
    ret := ShallowCopy(LM(poly));
    ret[2][1] := LC(poly);
  return ret;
end;;

# This function checks if U = UI*LTerm*UR. If it does then it will
# return a list with UI and UR ([UI,UR]).

# Input: U and LTerm
# Output: A list with Ui and Ur
# How its done: Compares the list LTerm with posible sublists
# of U and if it finds then it will create UI as all the
# variables until the variable before the first that matches
# the first variable of Lterm in U. Similarly for UR, but from
# the next variable in U that matches with the one of LTerm.

ifDiv:= function(MonU,MonLTerm)
local founds,continues,primerito,lastui,primeritour,newlists
,UI,UR,k,l,j,U,LTerm;
       if Length(MonU) <> 2 or Length(MonLTerm) <> 2 then
          Error("A ifDiv estan entrando cosas raras!!!!!!\n");
       else
          U := ShallowCopy(MonU[1][1]);
          LTerm := ShallowCopy(MonLTerm[1][1]);
          founds := false;
          continues := true;
          primerito := 1;
          newlists := [];
          UI := [];
          UR := [];
          lastui:=1;

    while continues = true do
          if Length(U) - primerito + 1 < Length(LTerm) then
               continues := false;
        elif LTerm[1] = U[primerito] then
              newlists:=[];
     for j in [primerito..(primerito + Length(LTerm) - 1)] do
           newlists := Concatenation(newlists,[U[j]]);
              od;
          if newlists = LTerm then
             founds := true;
             continues := false;
                   if primerito = 1 then
                       UI := [];
                   else
                   lastui := primerito - 1;
```

```
                    for k in [1..lastui] do
                  UI := Concatenation(UI,[U[k]]);
                  od;
               fi;
                  if primerito = 1 then
               primeritour := lastui + Length(LTerm);
                  else
                   primeritour := lastui + Length(LTerm)+1;
                   fi;
                     if primeritour > Length(U) then
                   UR := [];
                    else
                   for l in [primeritour..Length(U)] do
                   UR := Concatenation(UR,[U[l]]);
                         od;
                       fi;
    else
              primerito := primerito + 1;
     fi;
      else
              primerito := primerito + 1;
      fi;
   od;

     if founds = false then
        return 0;
     else
        return [[[UI],[1]],[[UR],[1]]];
     fi;
    fi;
end;;

# This function return the reminder of the division algorithm,
#in the
# form of a list, for a noncommutative polynomial.
# Input: gg anf FF.
# gg: the divisor in the form of a noncommutative polinomial
# list.
# FF: the list of the two "dividendo" in the form of a list.
# Ex.
# f1: is the first polynomial.
# f2: the second polynomial.
# FF: would be in the form FF:=[f1,f2].
# Output: the remainder of the division algorithm in the
# form of a list.

algoDiv:= function(gg,FF)
  local r,u,c,j,found,ui,ur,damcoef,g1,g2,g3,g,F,LMFj;
```

```
  g := ShallowCopy(gg);
  F := ShallowCopy(FF);
  r := [[],[]];
  damcoef :=0;
  g1:=0;
  g2:=0;
  g3:=0;
  ui:=0;
  ur:=0;

while g <> [[],[]] do
  u:= LM(g);
  c:= LC(g);
  j:= 1;
  found:=false;

    while j<=Length(F) and found = false do
        LMFj:= LM(F[j]);
          if ifDiv(u,LMFj) <> 0 then
             found:=true;
             ui:= ifDiv(u,LMFj)[1];
             ur:= ifDiv(u,LMFj)[2];
             u:= MulNP(ui,MulNP(LMFj,ur));
             damcoef:= [[[]],[c/LC(F[j])]];
             g1:=MulNP(damcoef,ui);
             g2:=MulNP(g1,F[j]);
             g3:=MulNP(g2,ur);
             g:= AddNP(g,g3,1,-1);
          else
             j:=j+1;
          fi;
    od;

  if found = false then
     r:= AddNP(r,LTer(g),1,1);
     g:= AddNP(g,LTer(g),1,-1);

  fi;
od;
  return r;
end;;

# Returns a random monomial of lenght 'monlen' and  'vars'
# different variables
# and with max coefficient 'coeffmax.

# Input: monlen, vars and coeffmax
#     * monlen: the maximun lenght of the monomial.
#     * vars: the number of variables
```

```
#     * coeffmax: the maximun value of the coefficients.
# Output: A Random monomial y the form of a list.

RandMon := function (monlen, vars, coeffmax)
   local poly, mon, rs1, ml,i;
   poly := [[],[]];
   mon := [];
   rs1 := GlobalRandomSource;
   ml := Random(rs1,[1..monlen]);
      for i in [1..ml] do
         Add(mon,Random(rs1,[1..vars]));
      od;
   Add(poly[1],mon);
   Add(poly[2],Random(rs1,[1..coeffmax]));
     return poly;
end;;

# Returns a random polynomial with number of terms 'Terms'.

# Input: Terms, monlens, varss and coeffmaxs
#    * Terms: the maximun number of terms in the polynomial.
#    * monlens: the maximun lenght of the monomials in the
#polynomial.    * varss: the maximun number of variables.
#    * coeffmaxs: the maximun value of the coefficients.
# Output: A random polynomial in the form of a list.

RandPol := function(Terms, monlens, varss, coeffmaxs)
    local poly, monom, j;
    j := 1;
    poly := RandMon(monlens,varss,coeffmaxs);

        while j <= Terms do
           monom := RandMon(monlens, varss, coeffmaxs);
           j := j + 1;
           poly:=AddNP(poly,monom,1,1);
        od;
    return poly;
end;;

# This function creates F1,F2,H1,H2 such that when you do the
# multiplication F1*q1*H1 and F2*q2*H2 the leading terms of q1
# and q2 go away. this is because if you divide by the public
# key (q1 and q2) you can get the message.

FQH := function()
local mF1, mF2, mH1, mH2, F1, H1, F2, H2,temp, LengthmF1,
LengthmH2,rs1,i;

   mF1:=[];
```

```
   mH2:=[];
   mF2:= [1,3]
   #this line depends on how you define your f,g and h.
   mH1:= [3,1];
   H1:=[[[]],[]];
   F1:=[[[]],[]];
   F2:=[[[]],[]];
   H2:=[[[]],[]];
   LengthmF1 := 0;
   LengthmH2 := 0;
   rs1 := GlobalRandomSource;

   LengthmF1 := Random(rs1,[3..25]);
   LengthmH2 := Random(rs1,[3..25]);

       for i in [1..LengthmF1] do
         Add(mF1,Random(rs1,[1..3]));
       od;
   temp := List(mF1);
       for i in [1..Length(mF2)] do
         Add(temp,mF2[i]);
       od;
   mF2 := List(temp);

       for i in [1..LengthmH2] do
         Add(mH2,Random(rs1,[1..3]));
       od;

       for i in [1..Length(mH2)] do
         Add(mH1,mH2[i]);
       od;

F1 := AddNP([[[mF1]],[-1]],RandPol(10,Length(mF1)-1,3,34567),1,1);
F2 := AddNP([[[mF2]],[1]],RandPol(10,Length(mF2)-1,3,34567),1,1);
H1 := AddNP([[[mH1]],[1]],RandPol(10,Length(mH1)-1,3,34567),1,1);
H2 := AddNP([[[mH2]],[1]],RandPol(10,Length(mH2)-1,3,34567),1,1);

   return [F1,F2,H1,H2];
   Reset(rs1);
end;;

# This function encrypt the message, that is in the form of a
# polynomial in this case in the form of a list. The values of
# a, b and c are constant, if you whant to change them it has
# to be only in the code.

# Input: mensaje (In English message)
#   * mensaje: the message in the form of a polynomial (list).
# Outpu: A polynomial, the encrypted message.
```

```
Encrypt := function(mensaje)
local F, A, G, Q, x, y, z,encrypt, div1, g, g1, h1, h;
local f1, f, F1, F2, H1, H2, q1, q2,a ,b, c;

   F:=ZmodnZ(9973);
   A:=FreeAssociativeAlgebraWithOne(F,"x","y","z");
   G:=GeneratorsOfAlgebraWithOne(A);
   GBNP.ConfigPrint("x","y","z");

   a:=5328;
   b:=6426;
   c:=878;

   x:=G[1];
   y:=G[2];
   z:=G[3];

   f1:=x-a*x^0;
   f:=GP2NP(f1);

   g1:=z-c*x^0;
   g:=GP2NP(g1);

   h1:=y-b*x^0;
   h:=GP2NP(h1);

   Q:=FQH();

   F1:=Q[1];
   F2:=Q[2];
   H1:=Q[3];
   H2:=Q[4];

   q1:=AddNP(MulNP(MulNP(f,g),h),MulNP(h,g),1,1);
   q2:=AddNP(MulNP(MulNP(h,g),f),MulNP(g,h),1,1);

   CleanNP(encrypt);
encrypt:=AddNP(AddNP(MulNP(MulNP(F1,q1),H1),MulNP
(MulNP(F2,q2),H2),1,1),mensaje,1,1);

   return encrypt;
end;;

# This function reduces the polynomial of the encrypted
# message by the private key g. Returns the remainder,
# that is the decrypted message.

# Input: mensaje (In English message).
```

```
# * mensaje: In this case the message is the return
# polynomial oin the encryption.
# Output: The decryption, the output would be the message.

Decrypt := function(mensaje)
  local div1,F,A,G,c,x,y,z,g1,g;

  F:=ZmodnZ(9973);
  A:=FreeAssociativeAlgebraWithOne(F,"x","y","z");
  G:=GeneratorsOfAlgebraWithOne(A);
  GBNP.ConfigPrint("x","y","z");

  c:=878;

  x:=G[1];
  y:=G[2];
  z:=G[3];

  g1:=z-c*x^0;
  g:=GP2NP(g1);

  div1:=algoDiv(mensaje,[g]);

  return div1;
end;;
```

References

[1] Rai, Tapan: "Infinite Gröbner Bases and Noncommutative Polly Cracker Cryptosystems," Ph.D. Thesis, Virginia Polytechnic Institute and State University, 2004.

[2] D. Cox, J. Little and D. O'Shea: *Ideals, varieties and algorithms: an introduction to computational algebraic geometry and commutative algebra*, 3rd ed. Springer, 2007.

[3] P. Ackermann, M. Kreuzer: "Gröbner Basis Cryptosystems," *Appl. Algebra Eng. Commun. Comput.*, **17** (2006), 173-194.