Improving an algorithm to solve Multiple Simultaneous Conjugacy Problems in braid groups

Juan González-Meneses

November 2002.

Abstract

There are recent cryptographic protocols that are based on Multiple Simultaneous Conjugacy Problems in braid groups. We improve an algorithm, due to Sang Jin Lee and Eonkyung Lee, to solve these problems, by applying a method developed by the author and Nuno Franco, originally intended to solve the Conjugacy Search Problem in braid groups.

1 Introduction

In [14], Sang Jin Lee and Eonkyung Lee give an algorithm to solve the following problem, that they call Multiple Simultaneous Conjugacy Problem (MSCP), in the braid group B_n: given the r-tuples $(a_1, \dots, a_r)$ and $(x^{-1}a_1x, \dots, x^{-1}a_rx)$ in B_n, find the conjugator x.

This problem has been proposed for cryptographical applications: There is a Key Agreement Protocol proposed by Anshell, Anshell and Goldfeld in [2], improved by the same authors and Fisher in [1], which is based on the difficulty to solve a MSCP in some groups. Braid groups have been proposed as a good choice. There have been different attacks to this cryptosystem, namely length-based attacks ([13], [9]), linear algebraic ones ([14], [12]) and others ([11]). But the algorithm we describe in this paper can be thought of as a direct attack to the base problem of the protocol.

We will assume that the reader is familiar with the basic notions in braid theory, which can be found in [3] or [15]. It is also desirable to know the work in [10], [7] and [16].

Recall that, given a braid $a \in B_n$, the integer $\inf(a)$ is the biggest $k \in \mathbb{Z}$ such that $a = \Delta^k p$, where Δ is the usual Garside element (half twist of all the strands) and p is a positive braid (all its crossings are positive).

The algorithm in [14] works as follows: First they define, for every r-tuple of braids, $\alpha = (a_1, \dots, a_r) \in (B_n)^r$, the set $C^{\inf}(\alpha)$ consisting of all

$\beta = (b_1, \ldots, b_r) \in (B_n)^r$ such that $\inf(b_i) \geq \inf(a_i)$ for all i and there exists some $\omega \in B_n$ satisfying $b_i = \omega^{-1} a_i \omega$ for all i simultaneously (that is, $\beta = \omega^{-1}\alpha\omega$). Then they prove the following result:

Theorem 1.1. [14] *Let $\alpha = (a_1, \ldots, a_r)$ and $\beta = (b_1, \ldots, b_r)$ be an instance of a MSCP in B_n, and x a positive solution. Then one can compute a positive braid x_0 and a r-tuple $\beta' = (b'_1, \ldots, b'_r) \in C^{\inf}(\alpha)$ such that $b'_i = x_0 b_i x_0^{-1}$ for all i, in time proportional to*

$$n(\log n)|x| \left(|x| + \sum_{i=1}^{r} (|a_i| + |b_i|) \right),$$

where $|\cdot|$ denotes word length in generators. Moreover $x = x_1 x_0$ for some positive braid x_1.

Here $C^{\inf}(\alpha)$ plays the role of the *Summit Set* defined in [10] to solve the conjugacy problem in B_n, in the sense that it satisfies the following result:

Theorem 1.2. [14] *Given $\beta \in C^{\inf}(\alpha)$, there exists a chain of elements $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_{k+1} = \beta$ in $C^{\inf}(\alpha)$, where successive elements are simultaneously conjugated by a permutation braid. In other words, there exist permutation braids $s_1, \ldots, s_k$ such that $s_j^{-1}\alpha_j s_j = \alpha_{j+1}$ for every $j = 1, \ldots, k$.*

Therefore, by classical methods (see [10]), one can use these two results to solve any MSCP in finite time. Nevertheless, this classical approach gives a computational complexity which is exponential with respect to the braid index n, and involves the cardinality N of the set $C^{\inf}(\alpha)$.

S. J. Lee and E. Lee expect in [14] that one can apply the methods in [8] to this algorithm, so that the computational complexity becomes a polynomial in (n, r, l, N), where l is the maximal word-length of the a_i's and b_i's. Here we show that this is the case. More precisely, we show:

Theorem 1.3. *Let $\alpha = (a_1, \ldots, a_r) \in (B_n)^r$ and let $\beta = (b_1, \ldots, b_r) \in C^{\inf}(\alpha)$. Let l be the maximal word length of the a_i's and b_i's, and let N be the number of elements in $C^{\inf}(\alpha)$. Then one can compute a braid $x \in B_n$ such that $x^{-1}\alpha x = \beta$ in time $O(Nrl^2n^3)$.*

2 Minimal simple elements for MSCP

Let us consider the *Artin monoid of positive braids*, B_n^+. We can define a *prefix order* on its elements, $\prec$, as follows: for $a, b \in B_n^+$, $a \prec b$ if and only if there exists $c \in B_n^+$ such that $ac = b$. We will say that a is a *prefix* (or a *divisor*) of b, or that b is divisible by a. This is a partial order on B_n^+, with some nice properties: For every $u, v \in B_n^+$ there exists their *least common multiple*, $u \vee v$, and their *greatest common divisor*, $u \wedge v$. There also exists an element Δ (which is represented by a half twist of all the strands) which, together with the above partial order, endows B_n^+ with a structure of *Garside monoid*, so B_n is a *Garside group* (cf. [6] [5]).

The *permutation braids*, also called *simple elements*, are the prefixes (or divisors) of Δ. We denote by S the set of simple elements. In B_n^+ there are $n!$ simple elements.

The algorithm used in [14] to solve a MSCP goes as follows: given $\alpha, \beta \in (B_n)^r$ conjugated, one computes $\beta' \in C^{\inf}(\alpha)$ as in Theorem 1.1. Then one must construct the whole $C^{\inf}(\alpha)$ using the method by Garside: Conjugate α by all simple elements. If new elements in $C^{\inf}(\alpha)$ are obtained, conjugate each one of them by all simple elements. Continue until no new elements appear. At that point, by Theorem 1.2, we will have computed the whole $C^{\inf}(\alpha)$ and moreover, we will know a chain going from α to any other element in $C^{\inf}(\alpha)$, as in Theorem 1.2. Hence, the chain associated to β', together with the element x_0 in Theorem 1.1 will give us the solution to the MSCP.

One of the main problems of this algorithm is the size of S. For every element in $C^{\inf}(\alpha)$ one must compute $n!$ conjugations! The idea in [8] is to consider very small subsets of S, which can be fastly computed, satisfying some suitable properties that allow the classical algorithm to work with them, instead of the whole S. The general method to compute these small subsets is the following.

Let $\mathcal{P}$ be a property for simple elements, and let $S_{\mathcal{P}}$ be the set of simple elements satisfying $\mathcal{P}$. Then $\min(S_{\mathcal{P}})$ is defined as the set of minimal elements (with respect to $\prec$) in $S_{\mathcal{P}}$. We must then define some suitable properties.

Let $J = (j_1, \dots, j_r) \in \mathbb{Z}^r$ and let C_J be the set of r-tuples $\delta = (d_1, \dots, d_r) \in (B_n)^r$ such that $\inf(d_i) \geq j_i$ for all i.

Definition 2.1. Let $J = (j_1, \dots, j_r) \in \mathbb{Z}^r$ and let $\delta = (d_1, \dots, d_r) \in C_J$. We say that a simple element s satisfies the property $\mathcal{P}(\delta, J)$ if $s^{-1}\delta s \in C_J$. In other words, if $\inf(s^{-1}d_i s) \geq j_i$ for all i.

Now consider the subsets $S_{\delta,J} = \min(S_{\mathcal{P}(\delta,J)}) \subset S$, where $\delta \in C_J$. These are the small subsets of S we were talking about. We can use them to solve a MSCP by means of the following result:

Proposition 2.2. *Given $\alpha = (a_1, \dots, a_r) \in (B_n)^r$, let $J = (\inf(a_1), \dots, \inf(a_r)) \in \mathbb{Z}^r$. For every $\beta \in C^{\inf}(\alpha)$, there exists a chain $\alpha = \alpha_1, \alpha_2, \dots, \alpha_{k+1} = \beta$ in $C^{\inf}(\alpha)$, where for $j = 1, \dots, k$, α_j is conjugated to α_{j+1} by a simple element $s_j \in S_{\alpha_j,J}$. That is, $s_j^{-1}\alpha_j s_j = \alpha_{j+1}$ and s_j is minimal among the simple elements conjugating α_j to an element in C_J.*

Proof. This result is analogous to Proposition 4.10 in [8]. It suffices to take the chain given in Theorem 1.2 and decompose every simple element into minimal ones. We notice that we obtain a chain of elements in C_J, but since all these elements are conjugated to α, they all belong to $C^{\inf}(\alpha)$. $\square$

3 Size of $\mathbf{S}_{\delta,\mathbf{J}}$

In this section we will show that the cardinal of $S_{\delta,J}$, for every $J \in \mathbb{Z}^r$ and every $\delta \in C_J$, is always smaller that n. Hence, if we know how to compute it fastly, we will improve considerably the speed of the algorithm by Lee and Lee (recall that $\#(S) = n!$). We will need the following results:

Proposition 3.1. [8] *If a property $\mathcal{P}$ is closed under gcd (i.e., if $s_1, s_2 \in S_{\mathcal{P}}$ implies $s_1 \wedge s_2 \in S_{\mathcal{P}}$) then $\#(\min(S_{\mathcal{P}})) \leq n-1$.* $\square$

Proposition 3.2. *For every $J \in \mathbb{Z}^r$ and every $\delta \in C_J$, the property $\mathcal{P}(\delta, J)$ is closed under gcd.*

Proof. Suppose that $s_1, s_2 \in S_{\mathcal{P}(\delta,J)}$, that is, for every $i = 1, \ldots, r$, $\inf(s_1^{-1} d_i s_1) \geq j_i$ and $\inf(s_2^{-1} d_i s_2) \geq j_i$. Since $\delta \in C_J$ one has $d_i = \Delta^{j_i} p_i$ for some positive braid p_i. Then

$$s_1^{-1} d_i s_1 = s_1^{-1} \Delta^{j_i} p_i s_1 = \Delta^{j_i} \tau^{j_i}(s_1^{-1}) p_i s_1,$$

where τ is the inner automorphism of B_n which consists on conjugation by Δ. Hence, $\inf(s_1^{-1} d_i s_1) \geq j_i$ means that $\tau^{j_i}(s_1^{-1}) p_i s_1$ is positive, or in other words: $\tau^{j_i}(s_1) \prec p_i s_1$. In the same way one has $\tau^{j_i}(s_2) \prec p_i s_2$ for all i. We must therefore show that, for $i = 1, \ldots, r$, one has $\tau^{j_i}(s) \prec p_i s$, where $s = s_1 \wedge s_2$.

Since τ is a homomorphism that preserves the prefix order, then $\tau^{j_i}(s_1) \wedge \tau^{j_i}(s_2) = \tau^{j_i}(s_1 \wedge s_2) = \tau^{j_i}(s)$. This implies $\tau^{j_i}(s) \prec p_i s_1$ and $\tau^{j_i}(s) \prec p_i s_2$, hence $\tau^{j_i}(s) \prec (p_i s_1) \wedge (p_i s_2) = p_i(s_1 \wedge s_2) = p_i s$, as we wanted to show. $\square$

Corollary 3.3. *For every $J \in \mathbb{Z}^r$ and every $\delta \in C_J$, the set $S_{\beta,J} = \min(S_{\mathcal{P}(\beta,J)})$ has at most $n-1$ elements.* $\square$

4 How to compute $\mathbf{S_{\delta,J}}$

We will finally present an algorithm that computes $S_{\delta,J}$, given $J \in \mathbb{Z}^r$ and $\delta \in C_J$. This algorithm will have complexity $O(rl^2n^3)$. Hence, in the algorithm by Lee and Lee, we no longer need to conjugate every $\delta \in C^{\inf}(\alpha)$ by all simple elements (n! conjugations); we can compute $S_{\delta,J}$ and then we do no more than $n-1$ conjugations.

We first need to be more precise about the work in [8]. We saw in Proposition 3.1 that $\min(S_{\mathcal{P}})$ has at most $n-1$ elements; but be can actually say more: for every generator σ_i, there is exactly one element $r_i \in \min(S_{\mathcal{P}})$ such that $\sigma_i \prec r_i$. It can happen, however, that $r_i = r_j$ for some $i \neq j$. Anyway, in order to compute $\min(S_{\mathcal{P}})$ (in our particular case $S_{\delta,J}$), we just need to compute r_i for $i = 1, \ldots, n-1$.

It is also given in [8] a method to compute the least common multiple $s \vee p$ of a simple element s and a positive braid p. More precisely, the algorithm given in [8] computes a simple element s' such that $ps' = s \vee p$. This takes time $O(l^2 n \log n)$, where l is the word length of p, and n is the number of strands. Notice that, in terms of theoretical complexity, this algorithm is equivalent to the computation a normal form (cf. [16]). Furthermore, it is also shown in [8] that if p is given in left normal form, then the complexity becomes $O(ln \log n)$.

So let us suppose that we are given $J = (j_1, \ldots, j_r) \in \mathbb{Z}^r$ and $\delta = (d_1, \ldots, d_r) \in C_J$, and we want to compute $S_{\delta,J}$. As we said before, we just need to compute r_i for every $i = 1, \ldots, n-1$, where, in this case, r_i is the minimal simple element which is divisible by σ_i and conjugates δ to an element in C_J. We propose the following algorithm:

Algorithm to compute $\mathbf{r_i}$.

1. Let $D \subset \{1, \ldots, r\}$ consisting of those t such that $\inf(d_t) = j_t$.
2. For every $t \in D$, compute p_t such that $d_t = \Delta^{j_t} p_t$.
3. Let $s = \sigma_i$.
4. If $\tau^{j_t}(s) \prec p_t s$ for every $t \in D$, then return s. Stop.
5. Take $m \in D$ such that $\tau^{j_m}(s) \not\prec p_m s$.
6. Compute s' such that $(p_m s)s' = \tau^{j_m}(s) \vee p_m s$.
7. Let $s = ss'$ and go to step 4.

Proposition 4.1. *Given $J = (j_1, \ldots, j_r) \in \mathbb{Z}^r$, $\delta = (d_1, \ldots, d_r) \in C_J$ and $i \in \{1, \ldots, n-1\}$, the above algorithm computes r_i, the minimal simple element which is divisible by σ_i and conjugates δ to an element in C_J.*

Proof. The algorithm starts by considering just those d_t whose infimum is exactly j_t. This is due to the following fact: If we can write $d_t = \Delta^k p_t$ where $k > j_t$ and p_t is a positive braid, then for every simple element s we will have:

$$s^{-1} d_t s = s^{-1} \Delta^k p_t s = \Delta^k \tau^k(s^{-1}) p_t s = \Delta^{k-1}(\Delta \tau^k(s^{-1})) p_t s.$$

But $\tau^k(s)$ is a simple element, so $\Delta\tau^k(s^{-1})$ is a positive braid, hence the infimum of $s^{-1}d_t s$ is at least $k - 1 \geq j_t$. Therefore, we just need to care about those d_t where $t \in D$.

For every $t \in D$ one has $d_t = \Delta^{j_t} p_t$, where p_t is a positive braid. These elements p_t are computed in Step 2 just by computing the left normal form of d_t.

We want to find r_i, and we know that $\sigma_i \prec r_i$. In the algorithm, the simple element s will be the possible value of r_i. At every iteration of the loop in steps 4-7, we start with a simple element s such that $\sigma_i \prec s \prec r_i$, and we check if $s = r_i$. If it is not, we multiply s by some suitable simple element s', and we start again. We must show that this makes sense.

At Step 3 we set $s = \sigma_i$, so we are sure that $\sigma_i \prec s \prec r_i$. Then we start the loop. In order to decide if $s = r_i$, we must check if $\inf(s^{-1}d_t s) \geq j_t$ for all $t \in D$. But, in the same way as above, one has $s^{-1}d_t s = \Delta^{j_t}\tau^{j_t}(s^{-1})p_t s$, so $\inf(s^{-1}d_t s) \geq j_t$ if and only if $\tau^{j_t}(s^{-1})p_t s$ is a positive braid, or in other words, if $\tau^{j_t}(s) \prec p_t s$. This is what is checked at Step 4.

If Step 4 determined that $s \neq r_t$, we must have found some $m \in D$ such that $\tau^{j_m}(s) \not\prec p_m s$. Step 5 just takes one of these values.

Now it comes the main step: We know that $s \prec r_i$, so $r_i = s\widehat{s}$ for some simple element $\widehat{s}$. Moreover, $\inf(r_i^{-1}d_m r_i) \geq j_t$ so one has $\tau^{j_m}(r_i) \prec p_m r_i$. Hence, $\tau^{j_m}(s) \prec \tau^{j_m}(s)\tau^{j_m}(\widehat{s}) = \tau^{j_m}(r_i) \prec p_m r_i$ while on the other hand $p_m s \prec p_m s\widehat{s} = p_m r_i$. Therefore, the least common multiple $\tau^{j_m}(s) \vee p_m s$ must also divide $p_m r_i$. Step 6 computes this lcm. Actually, it computes s' such that $\tau^{j_m}(s) \vee p_m s = (p_m s)s'$. But since this divides $p_m r_i$, we finally obtain that $ss' \prec r_i$.

We must remark two facts: First, ss' is always a simple element, since it divides the simple element r_i. Second, s' cannot be trivial, since otherwise we would have $\tau^{j_m}(s) \vee p_m s = p_m s$, implying $\tau^{j_m}(s) \prec p_m s$, which gives a contradiction with the choice of m. Therefore, ss' is strictly greater than s, but still a divisor of r_i, so in Step 7 we set $s = ss'$, and start the loop again. This cannot run forever since the word length of s is increased at every iteration, so the maximal number of iterations is $\frac{n(n-1)}{2}$ (the word length of Δ).

Therefore, at a certain iteration, we will obtain $s = r_i$, and the algorithm stops at Step 4 giving the correct output. □

5 Theoretical complexity

The algorithm we presented in this paper is exactly as the one in [14] except for the computations of $S_{\delta,J}$, for every $\delta \in C^{\inf}(\alpha)$. The main step is the computation of r_i given by the algorithm in the previous section. So we start by studying the complexity of this computation:

Proposition 5.1. *Given $J = (j_1, \ldots, j_r) \in \mathbb{Z}^r$, $\delta = (d_1, \ldots, d_r) \in C_J$ and $i \in \{1, \ldots, n-1\}$, one can compute r_i (the minimal simple element which is divisible by σ_i and conjugates δ to an element in C_J) in time $O(rl^2n^2)$ where l is the maximal word-length of the d_i's.*

Proof. We need to study the complexity of the algorithm in the previous section. First, Step 1 can be performed by computing the left normal form of every d_t. Every normal form takes time $O(l^2 n \log n)$, so Step one can be done in time $O(rl^2 n \log n)$.

The requirements of Step 2 can be achieved while doing Step 1: if some d_t has infimum j_t, we keep the value of p_t. Hence Step 2 is negligible, as well as Step 3.

Now we start a loop in Steps 4-7, which has at most $\frac{n(n-1)}{2}$ iterations, as we saw above. The only non-negligible steps are Steps 4 and 6. In Step 4, for every $t \in D$ we must compute $\tau^{j_t}(s)$, which can be done in linear time on the word size of s (at most $\frac{n(n-1)}{2}$), and then we must compute the left normal form of $p_t s$ taking time $O(ln \log n)$ (notice that p_t is already in left normal form). After performing these computations, to check if $\tau^{j_t}(s) \prec p_t s$ is $O(n \log n)$ (cf [16]). Hence Step 4 takes time $O(rln^2)$. On the other hand, Step 6 can be done in time $O(ln \log n)$ by [8]. Therefore, each iteration of the loop takes time $O(rln^2)$.

Now we could say that, since there are at most $\frac{n(n-1)}{2}$ iterations, all of them can be computed in time $O(rln^4)$. But we can do better than that: The different values of s in the successive iterations form an ascending chain of simple elements. Hence, the total number of computations performed in all the iterations is the same as if it were just one iteration, with the maximum value of s (see [16]). Therefore, the whole loop can be done in time $O(rln^2)$, and the whole algorithm takes time $O(rl^2n^2)$. $\square$

We can now apply this result to measure our contribution to the algorithm in [14]:

Proof of Theorem 1.3. One just need to apply the classical algorithm by Garside, together with the results given in Proposition 2.2 and Corollary 3.3. To be more precise, let $J = (\inf(a_1), \ldots, \inf(a_r)) \in \mathbb{Z}^r$. For every element $\delta \in C^{\inf}(\alpha)$ (there are N elements) one must compute $S_{\delta,J}$. This takes time $O(rl^2n^2)$ for every element, by the above result. Since there are at most $n-1$ elements, it takes time $O(rl^2n^3)$. Then one must conjugate δ by all the elements in $S_{\delta,J}$ (at most $n-1$), so we do at most $n-1$ conjugations by simple elements, each one taking time $O(ln \log n)$ since δ is already in left normal form.

The algorithm stops when we find β. So, in the worst case, the complexity of the whole computation is $O(Nrl^2n^3)$, as we wanted to show. $\square$

6 Final remarks

In this paper we have improved the algorithm in [14] to solve a MSCP. More precisely, we have improved a particular case of a MSCP, when the conjugate elements α and β are such that $\beta \in C^{\inf}(\alpha)$.

It is shown in [14] how to transform the general situation into this particular case (see Theorem 1.1), but the complexity of this step depends on the size of the solution! Therefore, using this method we do not have an upper bound for the complexity of the general case, in terms of the input data. Nevertheless, if our interest is to attack the cryptosystem in [2], where the secret key is the solution to the MSCP, then the complexity given in Theorem 1.1, to transform the general case into this particular case, yields a very efficient running time.

Nevertheless, if one dislikes to measure the complexity in terms of the length of the solution, one can do the following: given two conjugate elements $\alpha = (a_1, \ldots, a_r)$ and $\beta = (b_1, \ldots, b_r)$ in $(B_n)^r$, let $J = (j_1, \ldots, j_r) \in \mathbb{Z}^r$ where $j_i = \min(\inf(a_i), \inf(b_i))$. Then one has $\alpha, \beta \in C_J$. Now define $C^{\inf}(\alpha, \beta)$ as the set of $\delta \in C_J$ conjugate to α (thus to β). Then all the above results can be applied to $C^{\inf}(\alpha, \beta)$, so we do not need to pass through Theorem 1.1. That is, we have:

Theorem 6.1. *Let $\alpha = (a_1, \ldots, a_r)$ and $\beta = (b_1, \ldots, b_r)$ in $(B_n)^r$. Let l be the maximal word length of the a_i's and b_i's, and let M be the number of elements in $C^{\inf}(\alpha, \beta)$. Then one can compute a braid $x \in B_n$ such that $x^{-1}\alpha x = \beta$ in time $O(Mrl^2n^3)$.*

Anyway, we do not think that this is the better way to proceed, since $C^{\inf}(\alpha, \beta)$ will be, in general, much bigger than $C^{\inf}(\alpha)$, so one should try first to raise the infimum of the entries of α and β, before starting to construct the whole $C^{\inf}(\alpha, \beta)$.

On the other hand, the complexity given in Theorems 1.3 and 6.1 may lead to confusion, since one may think that we solved the MSCP in polynomial time. This is not true, since the factors N and M (the size of $C^{\inf}(\alpha)$ and $C^{\inf}(\alpha, \beta)$) may not be a polynomial in (n, r, l) (there is no known bounds for N or M in terms of (n, l, r)). All we can say by now is that N and M get smaller as r grows, so it seems that MSCP's are simpler than usual conjugacy problems in braid groups (see the discussion in [14] about the size of N).

Finally, the algorithm in this paper works not only for braid groups, but for a larger class of groups, called *Garside* groups (see [6], [5] and [8]), that share with braid groups the existence of simple elements and their basic properties. It can also be applied to other Garside structures in braid groups, as the one obtained from the presentation by Birman, Ko and Lee in [4].

References

[1] I. Anshel, M. Anshel, B. Fisher and D. Goldfeld, *New Key Agreement Protocols in Braid Group Cryptography.* Topics in Cryptology–CT-RSA 2001 (San Francisco, CA), 13-27, Lecture Notes in Comput. Sci., **2020**, Springer, Berlin, 2001.

[2] I. Anshel, M. Anshel and D. Goldfeld, *An algebraic method for public-key cryptography.* Math. Res. Lett. **6**, No. 3-4 (1999), 287-291.

[3] J. Birman, "Braids, links and mapping class groups". Princeton University Press, Princeton, 1974.

[4] J. Birman, K. H. Ko and S. J. Lee, *A new approach to the word and conjugacy problems in the braid groups*, Adv. Math. **139**, No. 2 (1998), 322-353.

[5] P. Dehornoy, *Groupes de Garside*, Ann. Scient. Éc. Norm. Sup., 4^e série, t. 35, 2002, 267-306.

[6] P. Dehornoy and L. Paris, *Gaussian groups and Garside groups, two generalizations of Artin groups*, Proc. London Math. Soc. **79**, No. 3 (1999), 569-604.

[7] E. A. Elrifai, H. R. Morton, Algorithms for positive braids, *Quart. J. Math. Oxford* **45** (1994), 479-497.

[8] N. Franco and J. González-Meneses, *Conjugacy problem for braid groups and Garside groups.* To appear in Journal of Algebra. Available at `www.arxiv.org/math.GT/0112310`

[9] D. Garber, S. Kaplan, M. Teicher, B. Tsaban and U. Vishne, *Length-based conjugacy search in the Braid group.* Preprint. Available at `www.arxiv.org/math.GR/0209267.`

[10] F. A. Garside, *The braid group and other groups*. Quart. J. Math. Oxford **20** (1969), 235-154.

[11] D. Hofheinz and R. Steinwandt, *A Practical Attack on Some Braid Group Based Cryptographic Primitives*. Accepted for presentation at the International Workshop on Practice and Theory in Public Key Cryptography - PKC 2003.

[12] J. Hughes, *A Linear Algebraic Attack on the AAFG1 Braid Group Cryptosystem*. The 7th Australasian Conference on Information Security and Privacy ACISP 2002, Lecture Notes in Computer Science, **2384**, 176–189, Springer-Verlag, New York 2002.

[13] J. Hughes and A. Tannenbaum, *Length-Based Attacks for Certain Group Based Encryption Rewriting Systems*, Workshop SECI02 Securité de la Communication sur Intenet, September 2002, Tunis, Tunisia.

[14] S. J. Lee and E. Lee, *Potential weaknesses of the commutator key agreement protocol based on braid groups*. L.R. Knudsen (Ed.): EUROCRYPT 2002, LNCS 2332, pp. 14-28, 2002.

[15] K. Murasugi and B. Kurpita, “A Study of Braids”, Kluwer, Dordrecht 1999.

[16] W. P. Thurston, Braid Groups, Chapter 9 of “Word processing in groups”, D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson and W. P. Thurston, Jones and Bartlett Publishers, Boston, MA, 1992.

Juan González-Meneses:

Dep. Matemática Aplicada I, ETS Arquitectura, Univ. de Sevilla, Av. Reina Mercedes 2, 41012-Sevilla (SPAIN).

E-mail: *meneses@us.es*