Des Codes Crypt (2006) 39:347–373
DOI 10.1007/s10623-005-5382-9

A weak key test for braid based cryptography

Samuel Maffre

Received: 4 May 2004 / Revised: 13 September 2005 /
Accepted: 5 October 2005
© Springer Science + Business Media, LLC 2006

Abstract This work emphasizes an important problem of braid based cryptography: the random generation of good keys. We present a deterministic, polynomial algorithm that reduces the conjugacy search problem in braid group. The algorithm is based on the decomposition of braids into products of canonical factors and gives a partial factorization of the secret: a divisor and a multiple. The tests we performed on different keys of existing protocols showed that many protocols in their current form are broken and that the efficiency of our attack depends on the random generator used to create the key. Therefore, this method gives new critera for testing weak keys. We also propose a new random generator of key which is secure against our attack and the one of Hofheinz and Steinwandt.

Keywords Braid groups · Conjugacy problem · Random generator · Weak key test · Cryptanalysis

AMS Classification 20F36 · 94A60

1. Introduction

Braid based cryptography appeared in 1999, (see [1]); and was developed in 2000, (see [19]). Braid groups are non-abelian groups, where the conjugacy morphism is not trivial. In fact, for some instances, conjugation is considered as a one-way function. The first cryptographic schemes have been subject to several cryptanalyses and developments that have contributed to define more and more normalized and efficient protocols. Nevertheless the current required key size is not competitive for general use with other cryptosystems like ECC or RSA. Thus, an important request is to find a random generator of shorter keys ensuring the same security.

Communicated by P. Wild.

S. Maffre (✉)
Laboratory XLIM, University of Limoges, 123 av. A. Thomas, 87060 Limoges Cedex, France
e-mail: samuel.maffre@unilim.fr

The main arguments in favor of braid groups are:

- the word problem is solved and we have a normal form for the elements of the group that can be efficiently computed (and that allows to digitalize elements).
- the conjugacy problem and its variants are a source of hard problems.
- the group operations can be easily implemented.

Until now, there are principally three methods to solve the conjugacy problem in braid group. The first one consists in working in some sub-set of the *conjugacy class*, see [8, 10, 12, 13, 17]. The second one is some attacks based on length or complexity, see [11, 14, 16, 20]. The third one uses linear representations, see [5, 15, 20, 21]. The choice of the random generator play a part in the security of braid based schemes. In practice, a conjugacy instance can be weak or strong. Our first aim is to give some critera to determine that and to prevent any length attack. The second goal is to propose a new random generator that keeps the same security with shorter key.

Our work is related to the one of Hofheinz and Steinwandt (see [14]). They propose an algorithm which computes a prefix of the secret ($d \prec a$); afterwards, if the canonical length of $d^{-1}a$ is 1, then they map the problem to the symmetric group and conclude. Our method is different: we give some results on the reduction of key by other means, then we use this reduction to evaluate the security of the key. We propose an algorithm which computes a multiple of the secret: knowing (x, axa^{-1}) and the canonical length of a, it computes m such that a is a prefix of m ($a \prec m$) and $axa^{-1} = m\tilde{x}m^{-1}$. This algorithm exploits the decomposition of braids in products of canonical factors. Used otherwise, it allows also to obtain a prefix of the secret: $d \prec a$. The knowledge of the canonical length of the secret is required but it is reasonable assumption: on the one hand the security is not based on this data because there are not many possible values; on the other hand we give some results to compute it.

The algorithm is deterministic and polynomial. Let $l{=}l(a)$ the size of the secret, and n the size of the group. Its complexity is $\mathcal{O}(l^3 n \log n)$, whereas the one of the efficient normal form *left-greedy decomposition* is $\mathcal{O}(l^2 n \log n)$. Thus it is very efficient too. To measure its efficiency, we consider $d^{-1}m$, $ak(a)$ and $rk(a)$: see the Fig. 1. This gives some critera, on the reduction of the secret, allowing to test weak keys. We have not theoretical result on these data. We can only compute them in practice; then we give some means.

For instance, in B_{60} (group with 60 strands) using a classical random generator which produces braids with a canonical length of 9, we get on an average: $l(a) \approx 8000$ and $ak(a) \approx 20$. The reduction of the size of the remaining secret is about $1/400$. Nevertheless, we note that even if only 20 generators remain unknown, the complexity of the rough completion is $59^{20} \approx 2^{117}$.

In fact, the efficiency of this "attack" depends on the random generator: some standard random generator produce a ratio of weak keys. Applying this attack to existing protocols, we notice that many are dangerously affected or broken. More precisely, for the standard

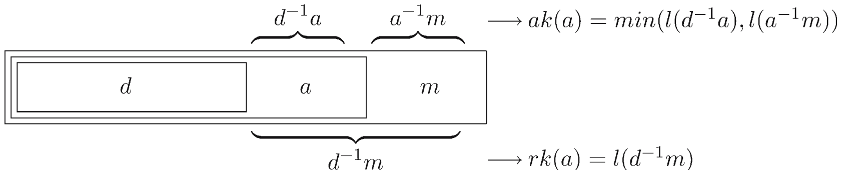
$d^{-1}a$
$a^{-1}m$
$\longrightarrow ak(a) = min(l(d^{-1}a), l(a^{-1}m))$
d
a
m
$d^{-1}m$
$\longrightarrow rk(a) = l(d^{-1}m)$

Fig. 1 Illustration of the reduction

random generator and for the proposed size of key, a large rate of conjugacy instances can be considered to be broken. Moreover, having a divisor and a multiple, we propose a way to reduce the rough completion.

For instance, we consider the schemes of [19] and a classical (canonical) random generator for the following conjugacy instance: B_{50}, the canonical length of the secret key is 5 and the one of the public key is 3. The percent of broken keys on 1000 is 99.99%.

This study allows us to determine reliable critera to generate randomly braids without increasing the size. Then we propose a new (canonical) random generator of braid. We hope that it answers to the question of [24]. Here are the contributions of this article:

– a prefix of the secret as efficient as [14]
– a multiple of the secret, knowing the canonical length of the secret
– some results to compute the canonical length of the secret and to manage the case of the Generalized Conjugacy Problem
– a test for weak keys
– a (canonical) random generator secure against our attack and the one of [14]
– some theoretical results to compute with braids

In any case, this study will hopefully lead to a more useful (reducing the size of key) and secure (testing weak keys) way to use braid based cryptography.

We begin, in Sect. 1, by recalling some background on braid groups and braid based cryptography. Next, in Sect. 2, we develop some theoretical results to handle braids. In Sect. 3, we describe the algorithm, its directions for use and its proof. Afterwards, in Sect. 4, we analyze its efficiency with some simulations and its applications to existing problems. In Sect. 5, we give some developments: the computation of the canonical length, a tester of weak key, a new random generator.

2. Background on braid and its use in cryptography

2.1. Introduction on the n-braid group

The braid group with n strands, denoted B_n, is defined as the fraction group of the following monoid presentation, named *Artin's presentation* (see [3]).

$$B_n^+ = \left\langle \sigma_1, \sigma_2, \ldots, \sigma_{n-1} \middle| \begin{array}{ll} \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{if } |i-j| = 1 \\ \sigma_i\sigma_j = \sigma_j\sigma_i & \text{if } |i-j| \geq 2 \end{array} \right\rangle \tag{1}$$

There are $n-1$ generators and the integer n, called *braid index*, identifies the group; it is a significant parameter of security. An element of B_n is called a *n-braid*. The terminology "*word*", commonly used in presented groups, refers to a writing of a group element. One defines the *Artin length* of a word by the number of generators belonging to its decomposition. A *positive word* is a word written as a product of some generators having a positive power. The *Artin length* takes more meaning in B_n^+ than in B_n, because the associated relations keep this length stable.

The great majority of handling (about normal forms, for instance) do not happen in the group but rather in the monoid. Wrongly, one says that the monoid is the set of positive braids.

B_n^+ has a *series* ***partial order***: set $a, b \in B_n^+$, $\quad a \prec b$ if $\exists c \in B_n^+$ such that $ac = b$

This order is called *prefix order*. One says *a divides b on the left*, or a is a *left divisor* of b, or again b is a *right multiple* of a. One defines, in the same way, a *right divisibility*, denoted $\prec^r$.

Remark 1 Notion of Greatest Common Divisor [9. chp. 9] W.P. Thurston has established the existence of a lattice structure in the monoid respecting defined partial order. Therefore, he shows the existence of a left great common divisor, $\wedge$, and a right least common multiple, $\vee$, of two elements. The unicity comes from the left-cancellativity of braid group.

$$\begin{aligned} \text{Set } a, b \in B_n^+, \quad d = a \wedge b \quad &\Leftrightarrow \quad \forall c: \quad c \prec a \text{ and } c \prec b \text{ iff } c \prec d \\ m = a \vee b \quad &\Leftrightarrow \quad \forall c: \quad a \prec c \text{ and } b \prec c \text{ iff } m \prec c \end{aligned}$$

2.1.1. Towards a normal form: the left-greedy decomposition

There is a particular element of $B_n^+ : \Delta = (\sigma_1\sigma_2\cdots\sigma_{n-1})\cdots(\sigma_1\sigma_2\sigma_3)(\sigma_1\sigma_2)\sigma_1$, called the *fundamental braid*. It is characterized by the fact that the set of its left divisors is equal to the set of its right divisors and this set generates B_n^+. These divisors are called *canonical factors* and their set is denoted by S_n. Here are some fundamental results on these objects:

Property 1 [12, Theorem 7, Lemma 2] [8, Theorem 2.6]

(i) Δ^2 *generates the center of* B_n*, that is the set of braids commuting with each other one.*
(ii) *For* $i \in [1, n-1]$, $\sigma_i\Delta = \Delta\sigma_{n-i}$.
(iii) *A bijection exists between* S_n *and the n-symmetric group,* Σ_n.

As in all non-abelian groups, the ***wordproblem*** appears in braid groups:

Given x and y, two words, the problem is to determine if $x \equiv y$ in B_n

This problem can be effectively solved in braid groups; two types of responses exist:

reduced forms: they lead to decide if a word (xy^{-1}) represents the trivial braid. See for example *Dehornoy's reduction*, ([6]).
normal forms: they define a canonical decomposition for each element of the group, see [8, 9, 12].

Remark 2 In practice, reduced forms are faster to compute; therefore, they keep a non-negligible role in cryptographic applications.

To define the normal form, we begin by giving a definition:

Definition 1 One defines for $a \in B_n^+$ the two following sets: the ***starting set*** $S(a) = \{\sigma_i;\ \sigma_i \prec a\}$ the ***finishing set*** $F(a) = \{\sigma_i;\ \sigma_i \prec^r a\}$

Theorem 1 [8] Let $a \in B_n$. There exists a unique decomposition: $a = \Delta^p A_1 A_2 \cdots A_k$ such that $p = max\{j \in \mathbb{Z};\ \Delta^j \prec a\}$ and for all i, $A_i \in S_n \backslash \{e, \Delta\}$, $\quad S(A_{i+1}) \subset F(A_i)$.

Definition 2 The "left-greedy decomposition" is the decomposition of a braid as in Theorem 1. With the same notations, one defines also: the "infimum": $inf(a) = p$, the "supremum": $sup(a) = p + k = min\{j \in \mathbb{Z};\ a \prec \Delta^j\}$ and the "canonical length": $cl(a) = sup(a) - inf(a) = k$.

2.1.2. Braids random generator

To handle words of a n-braid, we have two possible representations: Artin's one, which is the decomposition of a word as a product of group generators and their inverses; and the representation using normal forms, like the *left-greedy decomposition*, as a product of canonical factors (or "permutations").

Now, we present three random generators of n-braid:

ARG ARTIN RANDOM GENERATORS: the *Artin length*, l, is fixed; we make l random draws on $\{\sigma_i, \sigma_i^{-1}; 1 \leq i \leq n-1\}$ avoiding consecutive draws of the type $\sigma_i^{\varepsilon}\sigma_i^{-\varepsilon}$.
PARG POSITIVE ARTIN RANDOM GENERATORS: the *Artin length*, l, is fixed; we make l random draws on $\{\sigma_i; 1 \leq i \leq n-1\}$ and we can add on the left a factor of the type Δ^k where k is a random integer.
CRG CANONICAL RANDOM GENERATORS: the *canonical length*, cl, is fixed; considering Property 1 (iii), we make cl random draws on S_n. Afterwards, we reduce the n-braid to its *left-greedy decomposition*. While the *canonical length* is smaller than cl, we complete with other random draws. Finally, we can add a random factor Δ^k on the left.

Example in B_6 of representation of a braid:

$$\begin{aligned} a &= \sigma_4\sigma_2^{-1}\sigma_3\sigma_5^{-1}\sigma_2 \\ &= \Delta^{-1}\sigma_2^2\sigma_1\sigma_3\sigma_2\sigma_1\sigma_4\sigma_3\sigma_2^2\sigma_1\sigma_5\sigma_4\sigma_3\sigma_2\sigma_1 \text{ (Garside's normal form)} \\ &= \Delta^{-1}\underbrace{\sigma_2\sigma_3\sigma_2\sigma_1\sigma_4\sigma_3\sigma_2\sigma_5\sigma_4\sigma_3\sigma_2\sigma_1}_{\pi_1=(1624)(35)}\underbrace{\sigma_5\sigma_4\sigma_3\sigma_2}_{\pi_2=(26543)} \text{ left-greedy decomposition} \rightarrow (-1,\pi_1,\pi_2) \end{aligned}$$

Note that the permutation associated to a braid is the inverse of the permutation that the braid effects on the n strands. The Fig. 2 gives faithful representations to the current literature (the negative cross are in grey).

2.2. Cryptography in braid group

The cryptographic interest in braid groups comes mainly from the conjugacy morphism, which is considered as very hard to inverse on certain instances. Several derived problems are defined, which serve as support to protocols, and depend on a one-way function such as the following one:

$$\begin{array}{ccc} G \times B_n & \longrightarrow & B_n \times B_n \\ (a, x) & \longmapsto & (axa^{-1}, x) \end{array} \qquad \text{where } G \text{ is a sub-group of } B_n \tag{2}$$

In order to obtain a property like Diffie–Hellman's for discrete logarithm, we need of an argument replacing commutativity. We can obtain this property using two sub-groups of B_n:

$$LB_l = \langle \sigma_1, \sigma_2, \ldots, \sigma_{l-1} \rangle \quad \text{and} \quad RB_r = \langle \sigma_{n-r+1}, \sigma_{n-r+2}, \ldots, \sigma_{n-1} \rangle, \quad \text{where } l + r \leq n$$

In practice, we can set $l = \lfloor n/2 \rfloor = n - r$. Both sub-groups commute together, indeed: $\forall (i, j) \in [1, l-1] \times [n-r+1, n-1], \quad |i-j| > 1 \Rightarrow \sigma_i\sigma_j = \sigma_j\sigma_i$, thus

$$l + r \leq n \quad \Rightarrow \quad \forall (x, y) \in LB_l \times RB_r, \quad xy = yx$$

Here are the main problems, they come nearly all from [19].

Conjugacy Search Problem: (**CSP**)
Instance: let $(x, y) \in B_n \times B_n$ such that x and y are conjugate.
Objective: find $z \in B_n$ such that $y = zxz^{-1}$.

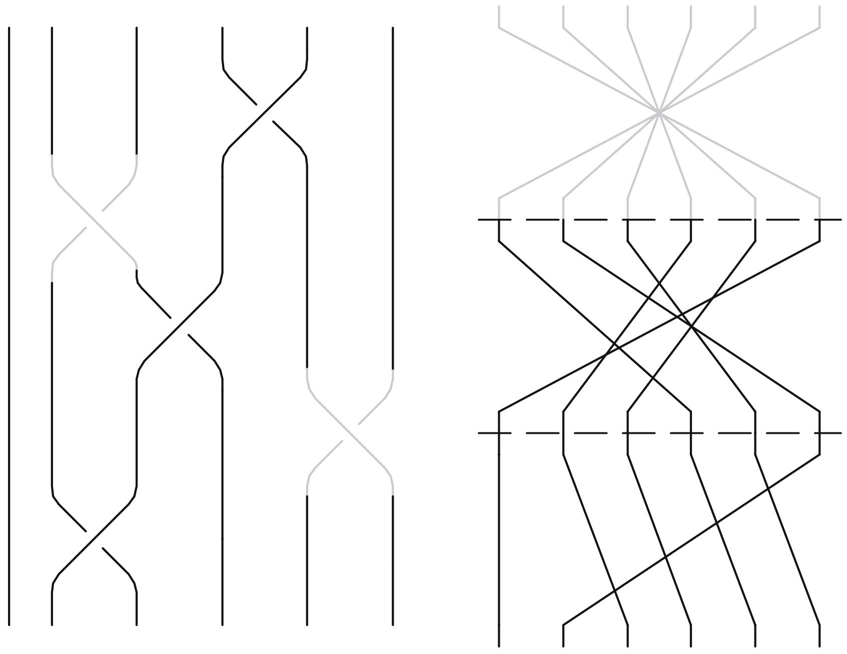
Fig. 2 Example in B_6 of representation of a braid

Generalized Conjugacy Problem: (**GCP**)
Instance: let $(x, y) \in B_n \times B_n$ which are conjugate in LB_l, [resp. RB_r].
Objective: find $z \in LB_l$ [resp. RB_r] such that $y = zxz^{-1}$.

Simultaneous Conjugacy Problem: (**SCP**)
Instance: let $(x_i, y_i)_{i\in[1,k]} \in (B_n^2)^k$ such that $\exists a \in B_n$ with $\forall i \in [1, k],\ y_i = ax_ia^{-1}$.
Objective: find $b \in B_n$ such that $\forall i \in [1, k],\ y_i = bx_ib^{-1}$.

Generalized Conjugacy Problem of the Diffie–Hellman type: (**GCP-DH**)

Instance: let $(x, y_a, y_b) \in B_n^3$ such that $\exists(a, b) \in LB_l \times RB_r$ with
$(y_a, y_b) = (axa^{-1}, bxb^{-1})$, $(l + r \leq n)$.
Objective: find $z \in B_n$ such that $z = by_ab^{-1}$ $(= ay_ba^{-1} = abxb^{-1}a^{-1})$.

Decomposition Problem: (**DP**)

Instance: let $(x, y) \in B_n^2$ such that $\exists(a_1, a_2) \in LB_l^2$ [resp. RB_r^2], such that $y = a_1xa_2$.
Objective: find $(b_1, b_2) \in {LB_l}^2$ [resp. ${RB_r}^2$] such that $y = b_1xb_2$.

Remark 3 From Property 1 (i), there is not unicity of the secret for a conjugacy instance. In practice, we can consider that the infimum of the secret is 0 or 1.

2.2.1. Cryptographic protocol

Braid based cryptography belongs to the field of asymmetric cryptography; the proposed services are various: generation of shared secrets [2, 19], authentification [25], signature [18], and of course, public key cryptosystem [19], ...

Here, we present only a **Public Key Cryptosystem**: it uses a cryptographic hash function $H : B_{l+r} \longmapsto \{0, 1\}^k$ that is modelled as a random oracle.

1: Key generation:
Choose an n-braid sufficiently complicated $x \in B_{n=l+r}$.
Choose an l-braid $a \in LB_l$, which will be the private key.
The public key is $(x, axa^{-1}) = (x, y)$.
2: Encryption of the message m:
Choose randomly an r-braid $b \in RB_r$.
The cipher-text is $(c, d) = (bxb^{-1}, H(byb^{-1}) \oplus m)$.
3: Decryption of (c, d) with the key a:
The message is $m = H(aca^{-1}) \oplus d$.

The presence of hash functions or the generation of shared secrets requires the use of a normal form. But normal forms are not indispensable to create protocols: in [25], authentification protocols are defined, which only use reduced forms.

Besides, the used protocol affects the choice of random generators, see [24]. Consider influence of the normal form on a random generated element:

- The CRG seems the most adapted: it preserves the canonical length and the Artin length remains acceptable.
- The PARG keeps stable the Artin length in the monoid. We do not control the canonical length. However, there is another normal form based on the *Artin length* : the *Garside's form*. It is less efficient than the *left-greedy decomposition*, but it is polynomial and preserves the *Artin length*, see [17]. Thus, this random generator could be used with an adapted normal form.
- The ARG seems adapted to reduced forms, where it is very efficient. A normalized element from it, has often a large (random) size.

3. Notations and theoretical results

The presented attack uses several tools; this section introduces them. The important results of this section are Properties 4, 5 and 6.

The monoid plays an important part in computation. Canonical factors seem to be adequate objects to work in the monoid. Then, we begin by introducing a new notation and two applications:

Notation 1 A product of canonical factors, $a_1a_2\cdots a_k$, is called "left greedy" if

$$\forall i \in [1, k],\quad a_i \in S_n\backslash\{e\} \quad \forall i \in [1, k-1],\quad S(a_{i+1}) \subset F(a_i)$$

The link with the *left-greedy decomposition* is evident : it is enough to replace the factors "Δ" on the top by the adequate power of Δ. We notice that the *left greedy* product of a is unique.

$$\begin{array}{rcl} \tau : B_n & \longrightarrow & B_n \\ a & \longmapsto & \Delta^{-1}a\Delta\ (= \Delta a\Delta^{-1}) \end{array} \qquad \begin{array}{rcl} \partial : S_n & \longrightarrow & S_n \\ q & \longmapsto & q^{-1}\Delta \text{ (such that } q\partial(p) = \Delta) \end{array}$$

The function τ is an automorphism of B_n; τ leaves stable B_n^+ and S_n and respect $\prec$. Moreover, from Property 1 (i), τ is an involution ($\tau = \tau^{-1}$). For instance, we have:

$$\text{if} \quad a = \sigma_1\sigma_{n-2}\sigma_3\sigma_{n-1}\sigma_4 \quad \text{then} \quad \tau(a) = \sigma_{n-1}\sigma_2\sigma_{n-3}\sigma_1\sigma_{n-4}$$

Here is a first result:

Property 2 *In S_n* $\quad \tau \circ \partial = \partial \circ \tau \quad$ *and* $\quad \partial^2 = \tau.$

Proof Let q be a canonical factor. From definition of ∂:

$$\partial(\partial(q)) = \partial(q^{-1}\Delta) = (\Delta^{-1}q)\Delta = \tau(q)$$

Thus τ is a power of ∂, then τ and ∂ commute. □

The *left greedy decomposition* of the inverse of an n-braid is given by

Property 3 [8, p 494] *If $\Delta^p A_1 A_2 \cdots A_k$ is the left-greedy decomposition of $a \in B_n$, then the one of a^{-1} is given by*

$$a^{-1} = \Delta^{-(p+k)}\tau^{-p-k}\left(\partial(A_k)\right)\tau^{-p-k+1}\left(\partial(A_{k-1})\right)\cdots\tau^{-p-1}\left(\partial(A_1)\right) \tag{3}$$

Notation 2 Given $a \in B_n^+$, we note: $\quad a^\star = a^{-1}\Delta^{sup(a)} \quad$ so that $\quad aa^\star = \Delta^{sup(a)}$.

Corollary 1 *If $a \in B_n^+$, then we have: $cl(a) = cl(a^{-1}) = cl(a^\star)$ and $inf(a^\star) = 0$. Moreover, if $a_1 a_2 \cdots a_k$ is a decomposition of a as a product of canonical factors, then the have again:*

$$a^\star = \partial(a_k)\tau\left(\partial(a_{k-1})\right)\cdots\tau^{k-1}\left(\partial(a_1)\right)$$

The following property is essential, it binds the left divisibility for a to the right divisibility for $a^\star$: the knowledge of a right multiple of a gives a right divisor of $a^\star$ and reciprocally, the knowledge of a left multiple of $a^\star$ gives a left divisor of a.

Lemma 1 *Let $a, b \in B_n^+$ such that $a = a_1 a_2 \cdots a_l$ and $b = b_1 b_2 \cdots b_l$ are products of some canonical factors (but the products are not necessarily "left greedy", and some factors can be in $\{1, \Delta\}$). Then we have:*

$$a \prec b \quad \Leftrightarrow \quad \partial(b_l)\tau(\partial(b_{l-1}))\cdots\tau^{l-1}(\partial(b_1)) \prec^r \partial(a_l)\tau(\partial(a_{l-1}))\cdots\tau^{l-1}(\partial(a_1))$$

The lemma follows directly from the definitions of $\prec$, $\prec^r$, τ and ∂.

Property 4 *If $a, b \in B_n^+$, then*

$$a \prec b \quad \Rightarrow \quad sup(a) \le sup(b), \tag{4}$$

$$a \prec b \quad \Leftrightarrow \quad b^\star \prec^r a^\star\Delta^{sup(b)-sup(a)}. \tag{5}$$

Proof To prove (4), there are two cases:

- $inf(a) = 0$: we proceed by contradiction and suppose that $cl(a) = sup(a) > sup(b)$. Let $a_1 a_2 \cdots a_{sup(a)}$ and $b_1 b_2 \cdots b_{sup(b)}$ be the *left greedy* product of a and b.

$$\begin{array}{lll} a \prec b & \Rightarrow & a \prec 1^{sup(a)-sup(b)} * b \\ & \Rightarrow_{\text{Lemma 1 and Corollary 1}} & b^\star\Delta^{sup(a)-sup(b)} \prec^r a^\star \\ & \Rightarrow & \Delta^{sup(a)-sup(b)} \prec^r a^\star \\ & \Rightarrow_{\text{Property 1 (ii)}} & inf(a^\star) \ge sup(a) - sup(b) > 0 \end{array}$$

From Corollary 1 this inequality is wrong; thus $sup(a) \le sup(b)$.
- $inf(a) > 0$: as $a \prec b \Rightarrow inf(a) \le inf(b)$; we can apply the first point after we have simplified beforehand by $\Delta^{inf(a)}$ both a and b.

(5): Considering a and b written in the *left greedy* product, it is enough to apply Lemma 1, knowing (4), to: $1^{sup(b)-sup(a)} * a \prec b$. □

The following properties are at the heart of the algorithm: the first one allows to find a multiple having the same *canonical length*. The second one will be used for a refinement of the algorithm.

Lemma 2 *Let $b, c \in B_n^+$ and let $b_1 \cdots b_{sup(b)}$ be the left greedy product of b. Then, we have: $\Delta \prec cb \quad \Leftrightarrow \quad \Delta \prec cb_1$*

Proof This lemma comes from [8, Property 2.10], even if the partial order is not the same. From Property 1(ii), we notice that "$\Delta \leq c$" $\Leftrightarrow \Delta \prec c$. □

Lemma 3 *Let $a \in S_n$, $b, c \in B_n^+$ and let $b_1 \cdots b_{sup(b)}$ be the left greedy product of b. Then, we have: $a \prec cb \quad \Leftrightarrow \quad a \prec cb_1$*

Proof As $a \in S_n$ and $\partial^4 = \tau^2 = Id$, we get:

$$a \prec cb \quad \Leftrightarrow \quad \Delta = \partial^3(a)a \prec \underbrace{\partial^3(a)c}_{\in B_n^+} b \quad \Leftrightarrow_{\text{Lemma 2}} \quad \Delta \prec \partial^3(a)cb_1 \quad \Leftrightarrow \quad a \prec cb_1 \quad \square$$

Property 5 *Let $a, b, c \in B_n^+$ and let $b_1 b_2 \cdots b_{sup(b)}$ be the left greedy product of b. Then we have:*

$$a \prec cb \quad \Rightarrow \quad a \prec cb_1 \cdots b_{sup(a)}$$

In particular $\quad a \prec b \quad \Rightarrow \quad a \prec b_1 \cdots b_{sup(a)}$

Proof We process by induction on $sup(a)$. If $a = a_1$ then Lemma 3 gives the result. Suppose that it is correct for $a = a_1 \cdots a_k$, then:

$$\begin{aligned} a_1 \cdots a_{k+1} \prec cb \quad &\Rightarrow \quad a_{k+1} \prec \underbrace{(a_1 \cdots a_k)^{-1} cb_1 \cdots b_k}_{\in B_n^+} b_{k+1} \cdots b_{sup(b)} \\ &\Rightarrow_{\text{Lemma 3}} \quad a_{k+1} \prec (a_1 \cdots a_k)^{-1} cb_1 \cdots b_{k+1} \end{aligned}$$

The case $k + 1$ is correct, then the property is true for any $sup(a)$. □

Corollary 2 *Let $a, b, c \in B_n^+$ and let $a_1 a_2 \cdots a_{sup(a)}$ and $b_1 b_2 \cdots b_{sup(b)}$ be the left greedy product of a and b. Then we have for $k \in [1, sup(a)]$:*

$$\begin{cases} a \prec cb \\ a_1 \cdots a_k \prec c \end{cases} \quad \Rightarrow \quad a \prec cb_1 \cdots b_{sup(a)-k}$$

Corollary 3 *Let $C = c_1 \cdots c_l \in B_n^+$ be a product of l canonical factors and $C_1 \cdots C_{sup(C)}$ be the left greedy product of C. Then we have:*

$$\forall k \in [1, sup(C)] \qquad C_k \cdots C_{sup(C)} \prec^r c_k \cdots c_l$$

In particular, if $a = cb$ with $a, b, c \in B_n^+$ and $a_1 \cdots a_{sup(a)}$ is the left greedy product of a, then we have:

$$a_{sup(c)+1} a_{sup(c)+2} \cdots a_{sup(a)} \prec^r b$$

Proof The case $k = 1$ is right. Suppose $k \in [2, sup(C)]$, we get:

$$c_1 \cdots c_{k-1} \prec c_1 \cdots c_l \prec C_1 \cdots C_{sup(C)} \quad \Rightarrow_{\text{Property 5}} \quad c_1 \cdots c_{k-1} \prec C_1 \cdots C_{k-1}$$

Then $c_k \cdots c_l = \underbrace{(c_1 \cdots c_{k-1})^{-1} C_1 \cdots C_{k-1}}_{\in B_n^+} C_k \cdots C_{sup(C)}$ □

Lemma 4 *Let $a_1 \cdots a_{sup(a)}$ and $b_1 \cdots b_{sup(b)}$ be the left greedy product of a and b. Then we have:*

$$\forall k \in [1, sup(a)], \qquad a \prec^r b \quad \Rightarrow \quad a_k \cdots a_{sup(a)} \prec^r b_k \cdots b_{sup(b)}$$

Proof We proceed by induction on $k \in [1, sup(a)]$. $k = 1$ is trivial. Suppose $a_k \cdots a_{sup(a)} \prec^r b_k \cdots b_{sup(b)}$ then there exists $m \in B_n^+$ such that

$$ma_k \cdots a_{sup(a)} = b_k \cdots b_{sup(b)} \quad \Rightarrow_{\text{Lemma 3}} \quad b_k \prec ma_k$$

This gives $a_{k+1} \cdots a_{sup(a)} \prec^r b_{k+1} \cdots b_{sup(b)}$ and ends the induction. □

Property 6 *Let $a, b, c \in B_n^+$. If $sup(a) > sup(b)$ then $a \prec^r bc \Rightarrow a_{sup(b)+1} \cdots a_{sup(a)} \prec^r c$*

Proof Let $d = bc$. Let $d_1 d_2 \cdots d_{sup(d)}$ be its decomposition in left greedy product:

$$d = bc \quad \Rightarrow_{\text{Corollary 3}} \quad d_{sup(b)+1} \cdots d_{sup(d)} \prec^r c$$

and if $sup(a) > sup(b)$ we get

$$a \prec^r d \quad \Rightarrow_{\text{Lemma 4}} \quad a_{sup(b)+1} \cdots a_{sup(a)} \prec^r d_{sup(b)+1} \cdots d_{sup(d)} \prec^r c$$ □

Remark 4 Corollary 3 is also a corollary of Property 6 and generalizes Lemma 2.9 from [10]. Moreover, Property 5 is a generalization of Lemma 4.10 from [23], which gives also (4).

4. A weakness in conjugacy problems

An advantage of non-abelian groups, for cryptography, is the real lack of identifiable structure. This property in braid groups is a cryptographic asset; however, with the apparition of normal forms like the *left-greedy decomposition*, a certain structure appears. In particular, canonical factors can be used.

We consider the *Conjugacy Search Problem*: set $a, x \in B_n$.
The aim is to determine a knowing only $(x, x' = axa^{-1})$. Our solution consists in building a right multiple and a left divisor of the secret a:

$$\text{find} \quad (d, m) \in B_n \quad \text{such that} \quad d \prec a \prec m \tag{6}$$

That gives a partial factorization of the secret. The knowledge of the *canonical length* of a is needed by our algorithm. In Sect. 6, we will give some techniques to find it. In this part, we begin by considering a simple framework and we propose an algorithm computing a multiple of the secret. Afterwards, we prove that it is deterministic and polynomial. To finish, we show that any case can be solved by the same algorithm and that it allows to determine a divisor too: it suffices to take the appropriate assumptions.

Remark 5 What is the foundation of this attack ?
The multiplication of two n-braids reverts to concatenate their decompositions. Then, when we make a product of two normalized words in a normal form, the whole decomposition is more or less altered. In fact, the more the first factors are complex, the less its decomposition is altered and the more information we can recover.

4.1. Elementary algorithm

| The studied situation is: let $a, x \in B_n$ with $inf(a) = inf(x) = 0$ and $x' = axa^{-1}$
| The supposed known data are: x, x' and $cl(a)$

We get from Notation 2 and the fact that τ is an involution:

$$x' = axa^{\star}\Delta^{-cl(a)} = \Delta^{-cl(a)}\tau^{cl(a) \mod 2}\left(axa^{\star}\right) = \Delta^{-cl(a)}X$$

Here, we want to present an algorithm usable with some other parameters, therefore we generalize its input format:

Input: $(X, l_1, l_2, \alpha, \beta) \in B_n^+ \times \mathbb{N}^2 \times \mathbb{Z}^2$ | Output: m

such as $\exists\, T, y \in B_n^+$; $\begin{cases} X = \tau^{\alpha}(T)y\tau^{\alpha+\beta}(T^{\star}) \\ l_1 = cl(y),\ l_2 = cl(T) \\ inf(T) = inf(y) = 0 \end{cases}$ $\quad T \prec m$

Algorithm 1 ***RightM*** $(X, l_1, l_2, \alpha, \beta)$

```
Input:  Set $X \in B_n^+$,  $l_1, l_2 \in \mathbb{N}$  et $\alpha, \beta \in \mathbb{Z}$.

Init:   $A := e$,  $Y := \tau^{-\alpha}(X)$
        Compute the left greedy product $Y_1Y_2\cdots Y_{\sup(Y)}$ of $Y$

Loop:   For $l$ from 1 to $l_2$ do                                          (a)
            $l := l + 1$,  $A := AY_1$,
            $Z := Y_1^{-1}Y\tau^{\beta+l_2-l}\left(\partial(Y_1)^{-1}\right)$                   (b)
            $Y := Y_1^{-1}Y$
            If $Z \notin B_n^+$ then break_For                               (c)
            Else Compute the left greedy product $Z_1Z_2\cdots Z_{\sup(Z)}$ of $Z$
                If $\sup(Z) = l_1 + 2(l_2 - l) + 1$ then                     (d)
                    $A := A\tau^{-\beta-l_2+l}(Z_{\sup(Z)})^{-1}$
                    $Z := \tau^{-\beta-l_2+l}(Z_{\sup(Z)})ZZ_{\sup(Z)}^{-1}$
                EndIf
                $Y :=$ left greedy product of $Z$
            EndIf
        EndFor
        If $l < l_2$ then $A := AY_2Y_3\cdots Y_{l_2-l+1}$ EndIf              (e)

Output: $A$
```

Remark 6 In fact, this algorithm rewrites X under the form:

$$X = \tau^{\alpha}(a)y\tau^{\alpha+\beta}(a^{\star}) \quad\longrightarrow\quad \tau^{\alpha}(m)\tilde{y}\tau^{\alpha+\beta}(m^{\star}) \text{ with } a \prec m$$

Remark 7 The test of line (d) can be improved, considering Property 6. But it complicates the proof of the algorithm without giving a clearly better result.

4.2. Proof and complexity

The halting problem of this algorithm is easy; it is linear except a simple loop *For ... do ... EndFor*. Then, it stops. Here, we prove it and study its complexity.

Proof We use the notations of the algorithm. This algorithm is undertaken to compute a right multiple of the *secret* T; thus, it is exactly what we prove now. The efficiency of the algorithm is its capacity to produce the smallest multiple possible; it is studied in Sect. 5. Here, we prove that A is a right multiple of T. We proceed by induction on l and denote the number of the iteration in "power", thus we have for begin:

$$l = 0 \quad \longrightarrow \quad A^0 = e \qquad Y^0 = \tau^{-\alpha}(X) = T\tau^{-\alpha}(y)\tau^{\beta}(T^{\star})$$

Let $t_1 \cdots t_{l_2}$ be the *left greedy* product of T. The property of induction is:

$$\text{for } l;\ \ 1 \le l \le l_2 \ \ \mathcal{P}(l) : \begin{cases} (1): \quad t_1 t_2 \cdots t_l \prec A^l \prec Y_1^0 Y_2^0 \cdots Y_l^0 \\ \\ (2): T \prec A^l Y_1^l \cdots Y_{l_2 - l}^l \prec Y_1^0 Y_2^0 \cdots Y_{l_2}^0 \end{cases} \tag{7}$$

The case $l = 0$ is not taken into account even if it is correct:

$$(2) \qquad T \prec Y^0 \quad \Rightarrow_{\text{Property 5}} \quad T \prec Y_1^0 Y_2^0 \cdots Y_{l_2}^0$$

Before beginning the induction, we note some points which will help the understanding of the proof:

- $\mathcal{P}(l)[1] \Rightarrow \sup(A^l) = l$
- By building, the reader can check at each stage that the following relation is true:

$$l;\ 1 \le l \le cl(T), \qquad Y^0 = A^l Y^l \tau^{\beta + l_2 - l}((A^l)^{\star})$$

- The condition $\mathcal{P}(l)[2]$ is here to manage the cases where the loop stops: line (c). It guarantees that the output element of the algorithm is a right multiple of T with the same supremum.

Initialization $l = 1$: Decomposing T, $T^{\star}$ and Y^0 in their *left greedy* product, we have:

$$Y^0 = t_1 t_2 \cdots t_{l_2} \tau^{-\alpha}(y)\tau^{\beta}(\partial(t_{l_2})) \cdots \tau^{\beta+l_2-2}(\partial(t_2))\tau^{\beta+l_2-1}(\partial(t_1)) = Y_1^0 \cdots Y_{\sup(Y^0)}^0$$

Then $t_1 \prec Y^0$ et $\tau^{\beta+l_2-1}(\partial(t_1)) \prec^r Y^0$

$$t_1 \prec Y^0 \quad \Rightarrow_{\text{Property 5}} \quad t_1 \prec Y_1^0 \quad \Rightarrow_{\text{Property 4}} \quad \partial(Y_1^0) \prec^r \partial(t_1)$$
$$\left(\Rightarrow \quad \tau^{\beta+l_2-1}(\partial(Y_1^0)) \prec^r \tau^{\beta+l_2-1}(\partial(t_1)) \prec^r Y^0 \right)$$

The veracity of assertion $\tau^{\beta+l_2-1}(\partial(Y_1^0)) \not\prec^r Y_2^0 \cdots Y_{\sup(Y^0)}^0$ is equivalent to the boolean (c): "$Z \notin B_n^+$" with $Z := {Y_1^0}^{-1} Y^0 \tau^{\beta+l_2-1}(\partial(Y_1^0))^{-1}$. There are two cases.

– True: The condition (c) is true, then $A^1 = Y_1^0$ and $Y^1 = Y_2^0 \cdots Y_{\sup(Y^0)}^0$; the process exits the loop and $\mathcal{P}(1)$ is true.
– False: We notice that:

$$Z \prec^r \underbrace{t_2 \cdots t_{l_2} \tau^{-\alpha}(y)\tau^{\beta}(\partial(t_{l_2})) \cdots \tau^{\beta+l_2-2}(\partial(t_2))}_{\sup(\text{''}) \le l_1 + 2l_2 - 2} \underbrace{\tau^{\beta+l_2-1}(\partial(t_1))\tau^{\beta+l_2-1}(Y_1^0)^{-1}}_{\in S_n}$$

Let $Z_1 \cdots Z_{\sup(Z)}$ be the *left greedy* product of Z; we note that:

$$\sup(Z) = l_1 + 2l_2 - 1 \quad \Rightarrow_{\text{Property 6}} \quad Z_{\sup(Z)} \prec^r \tau^{\beta+l_2-1}(\partial(t_1))\tau^{\beta+l_2-1}(Y_1^0)^{-1}$$

Now, we set:

$$\partial(A^1) \prec^r \partial(t_1) \quad \text{with} \quad \partial(A^1) := \begin{cases} \tau^{1-\beta-l_2}(Z_{\sup(Z)})\partial(Y_1^0) & \text{if } \sup(Z) = l_1 + 2l_2 - 1 \\ \partial(Y_1^0) & \text{else} \end{cases}$$

Then we have: $\partial(Y_1^0) \prec^r \partial(A^1) \prec^r \partial(t_1)$; the Lemma 1 gives:

$$\mathcal{P}(1)[1] \ : \ t_1 \prec A^1 \prec Y_1^0$$

Now, let $Y_1^1 \cdots Y_{\sup(Y_1)}^1$ be *left greedy* product of $Y^1 := {A^1}^{-1}Y^0\tau^{\beta+l_2-1}(\partial(A^1))^{-1}$ From Corollary 2 we get that $\mathcal{P}(1)[2]$ is true:

$$T \prec A^1 Y_1^1 \cdots Y_{l_2-1}^1 \prec Y_1^0 \cdots Y_{l_2}^0$$

<u>Heredity $l = k+1$:</u> Let k; $1 \le k < l_2$, we suppose that $\mathcal{P}(k)$ is true.

$$\begin{aligned}
\mathcal{P}(k) &\Rightarrow_{\text{Property 5}} t_1 \cdots t_{k+1} \prec A^k Y_1^k \\
&\Rightarrow_{\text{Lemma 1}} \partial(Y_1^k)\tau(\partial(t_k^k)) \cdots \tau^{k-1}(\partial(t_1^k)) \prec^r \partial(t_{k+1}) \cdots \tau^{k-1}(\partial(t_1)) \\
&\Rightarrow \quad \tau^{\beta+l_2-k-1}(\partial(Y_1^k))\tau^{\beta+l_2-k}(\partial(t_k^k)) \cdots \tau^{\beta+l_2-1}(\partial(t_1^k)) \prec^r Y^0 \\
&\qquad\qquad \text{with } Y^0 = A^k Y^k \tau^{\beta+l_2-k}((A^k)^\star) \\
&\Rightarrow \quad \tau^{\beta+l_2-k-1}\left(\partial(Y_1^k)\right) \prec^r A^k Y^k
\end{aligned}$$

The veracity of assertion $\tau^{\beta+l_2-k-1}(\partial(Y_1^k)) \not\prec^r Y_2^k \cdots Y_{\sup(Y^k)}^k$ is equivalent to the boolean test (c): "$Z \notin B_n^+$" with $Z := {Y_1^k}^{-1} Y^k \tau^{\beta+l_2-k-1}(\partial(Y_1^k))^{-1}$. There are two cases:

– True: The condition (c) is true; then $A^{k+1} = A^k Y_1^k$ and $Y^{k+1} = (Y_1^k)^{-1}Y^k$. The process goes out the loop. $\mathcal{P}(k)$ and the Property 5 give:

$$t_1 \cdots t_{k+1} \prec A^{k+1} \prec Y_1^0 \cdots Y_{k+1}^0 \quad \Rightarrow \quad \mathcal{P}(k+1)[1] \text{ is true}$$

Moreover $A^k Y_1^k \cdots Y_{l_2-k}^k = A^{k+1} Y_1^{k+1} \cdots Y_{l_2-k-1}^{k+1} \quad \Rightarrow \quad \mathcal{P}(k+1)[2]$ is true.

– False: We notice $Z = (A^k Y_1^k)^{-1} Y^0 \tau^{\beta+l_2-k-1}((A^k Y_1^k)^\star)^{-1}$. Since $\sup(A^k Y_1^k) = k+1$, the Corollary 1 gives the following relation:

$$Z \prec^r t_{k+2} \cdots t_{l_2}\tau^\beta(y)\tau^\beta(\partial(t_{l_2})) \cdots \tau^{\beta+l_2-k-2}(\partial(t_{k+2})c$$

with $c = \tau^{\beta+l_2-k-1}\left((t_1 \cdots t_{k+1})^\star((A^k Y_1^k)^\star)^{-1}\right) \in B_n^+$.
The Property 6 gives:

$$A^{k+1\star} \prec^r (t_1 \cdots t_{k+1})^\star$$

$$\text{with } A^{k+1\star} := \begin{cases} \tau^{k+1-l_2-\beta}(Z_{\sup(Z)})(A^k Y_1^k)^\star & \text{if } \sup(Z) = l_1 + 2(l_2 - k) - 1 \\ (A^k Y_1^k)^\star & \text{else} \end{cases}$$

Therefore we have $(Y_1^0 \cdots Y_{k+1}^0)^\star \prec^r (A^k Y_1^k)^\star \prec^r (A^{k+1})^\star \prec^r (t_1 \cdots t_{k+1})^\star$. From the Property 4, $\mathcal{P}(k+1)[1]$ is true. Let

$$Y^{k+1} = (A^{k+1})^{-1} Y^0 ((A^{k+1})^\star)^{-1}$$

The Corollary 3 gives $\mathcal{P}(k+1)[2]$.

This ends the induction.

Remark 8 The supremum of the multiple, given by this algorithm, is the same that the supremum of the secret. This is guaranteed by the second argument of $\mathcal{P}$.

4.2.1. Complexity

The following table gives the complexity of efficient algorithms working in braid groups. Let n is the *braid index* and l the maximum of both lengths, (see [4]). The complexity of *RightM* is clearly $\mathcal{O}(l^3 n \log n)$: the most expensive operation in the algorithm is the computation of the normal form and we notice that $\forall a \in B_n^+, \quad cl(a) \leq l$. The size of the variable x is considered as a multiplicative constant Table 1.

4.3. Directions for use

We have built an algorithm that computes a right multiple of the secret. Now, we are going to show that the same algorithm allows to compute a left divisor of the secret. Afterwards, we will show how this algorithm can solve any conjugacy instance and we finish by a remark on the Simultaneous Conjugacy Problem.

4.3.1. Obtaining of left divisor

The work made in previous sections, is done on the left. It produces a right multiple of a. If we do a similar work on the right, for instance using the *right-greedy decomposition*, then we obtain a left multiple of $a^\star$. From Property 4, that gives a left divisor of a. That is exactly what we propose to do with the same algorithm, introducing a new application:

$$\begin{array}{rcll}\textbf{reverse}: B_n & \longrightarrow & B_n & \\ a & \longmapsto & r(a) & \text{anti-automorphism such that } \forall i \in [1, n-1],\ r(\sigma_i) = \sigma_i \\ & & & \forall b, c \in B_n \ \ r(bc) = r(c)r(b)\end{array}$$

We notice that *reverse* leaves B_n^+ invariant and Lemma 3 (ii) from [12] gives $r(\Delta) = \Delta$.

Table 1 Table of complexity

Operation	Complexity
Canonical factor	
Product	$\mathcal{O}(n)$
Inverse	$\mathcal{O}(n)$
τ^k	$\mathcal{O}(n)$
gcd	$\mathcal{O}(n \log n)$
Random	$\mathcal{O}(n)$
n-Braid	
Product	$\mathcal{O}(ln)$
Inverse	$\mathcal{O}(ln)$
Left-greedy decomposition	$\mathcal{O}(l^2 n \log n)$
Comparison	$\mathcal{O}(l^2 n \log n)$
Random	$\mathcal{O}(ln)$
$\Rightarrow$ our algorithm	
Right M	$\mathcal{O}(l^3 n \log n)$

Property 7

(i) $\forall a, b \in B_n^+, \quad a \prec b \quad \Leftrightarrow \quad r(a) \prec^r r(b)$
(ii) *r and τ commute and r is an involution.*
(iii) $\forall a \in B_n, \quad inf(r(a)) = inf(a),\ sup(r(a)) = sup(a)$ *and* $cl(r(a)) = cl(a)$
(iv) $\forall a \in B_n^+, \quad (r(a^\star))^\star = r(a)\Delta^{-inf(a)}$

Proof (i): Let a, b be in B_n^+, then:

$$a \prec b \quad \Leftrightarrow \quad \exists c \in B_n^+;\ b = ac \quad \Leftrightarrow \quad \exists c \in B_n^+;\ r(b) = r(c)r(a) \quad \Leftrightarrow \quad r(a) \prec^r r(b)$$

(ii): In fact, we have from the definitions of both applications: $r \circ \tau = \tau^{-1} \circ r$. As $r(\Delta) = \Delta$ and τ is an involution, the result is established. Moreover, r is clearly an involution.

(iii): We have from Property 1 (ii): $\forall a \in B_n,\ \forall j \in \mathbb{Z},\ \Delta^j \prec a \quad \Leftrightarrow \quad \Delta^j \prec^r a$

$$inf(a) = max\{j \in \mathbb{Z};\ \Delta^j \prec a\} = max\{j \in \mathbb{Z};\ \Delta^j \prec^r a\}$$

From (i) and the relation $r(\Delta) = \Delta$ we have $inf(a) = inf(r(a))$.
It is the same way for $sup(a) = min\{j \in \mathbb{Z};\ a \prec \Delta^j\} = sup(r(a))$, then we get the relation on the canonical length by definition.

(iv): From the Notation 2 and the relation $r(\Delta) = \Delta$, for $a \in B_n^+$ we have:

$$aa^\star = \Delta^{sup(a)} \quad \Rightarrow \quad r(a^\star)r(a) = \Delta^{sup(a)}$$

Now $sup(r(a^\star)) = sup(a^\star) = cl(a^\star) = cl(a)$ and $inf(r(a)) = inf(a)$, thus:

$$r(a^\star)r(a)\Delta^{-inf(a)} = \Delta^{cl(a)} \quad \Rightarrow \quad r(a^\star)^\star = r(a)\Delta^{-inf(a)}$$

□

Using notations of the algorithm, in Sect. 4.1, we want to determine a left divisor of the *secret* T. The inputs are:

$$X = \tau^\alpha(T)y\tau^{\alpha+\beta}(T^\star),\ l_1 = cl(y),\ l_2 = cl(T), \alpha \text{ and } \beta, \text{ with } inf(T) = inf(y) = 0$$

A the left divisor of the secret is obtained by

$$d := r\big(RightM(r(X), l_1, l_2, \alpha + \beta, -\beta)^\star\big) \text{ with } d \prec T \tag{8}$$

Proof of (8) Considering Property 7 (ii) and (iv):

$$r(X) = \tau^{\alpha+\beta}(r(T^\star))r(y)\tau^\alpha(r(T)) = \tau^{\alpha+\beta}(r(T^\star))r(y)\tau^\alpha(r(T^\star)^\star)$$

The hypothesis for the computation of a right multiple of $r(T^\star)$ are correct: $inf(r(T^\star)) = inf(T^\star) = 0$, $inf(r(y)) = inf(y) = 0$, $cl(r(T^\star)) = cl(T^\star) = cl(T)$ and $cl(r(y)) = cl(y) = cl(x)$, then we get a right multiple of $r(a^\star)$ by

$$M := RightM(r(X), l_1, l_2, \alpha + \beta, -\beta) \text{ with } \begin{cases} r(T^\star) \prec M \\ sup(r(T^\star)) = sup(M) \end{cases}$$

Properties 4 and 7 (i) give successively: $M^\star \prec^r r(T^\star)^\star = r(T)$ and $r(M^\star) \prec T$.

□

Treat the case of the following elementary instance: Let $a, x \in B_n$ such that $inf(a) = inf(x) = 0$ and $x' = axa^{-1}$ and let $X = \Delta^{cl(a)}x' = \tau^{cl(a)}(axa^{\star})$. The attack on (x, x') consists in building $(d, m) \in B_n^{+2}$ such that $d \prec a \prec m$:

$$\left\|\begin{array}{l} m := RightM(X, cl(x), cl(a), cl(a) \bmod 2, 0) \\ d := r\,(RightM(r(X), cl(x), cl(a), cl(a) \bmod 2, 0)^{\star}) \end{array}\right.$$

We conclude that we can, with the same algorithm, obtain a left divisor and a right multiple of the secret.

4.3.2. Generalization of instance

Up to here, we have considered an elementary conjugacy instance. In fact, the previously described algorithm can attack any instance. We propose to generalize parameter x and a successively. All previous restrictions are analysed, only the choice of input variables changes.
Recall that the studied situation is: $a, x \in B_n$ and $x' = axa^{-1}$; the supposed known data are: x, x' and $cl(a)$. The aim is to find $(d, m) \in B_n$ such that $d \prec a \prec m$.

• $x \in B_n$: Treat the parameter "x", considering $inf(a) = 0$. We reduce to $inf(x) = 0$. Let $\Delta^{inf(x)}x_1x_2\cdots x_{cl(x)}$ be the *left-greedy decomposition* of $x = \Delta^{inf(x)}x^+ \in B_n$. The conjugacy problem becomes:

$$x' = axa^{-1} = a\Delta^{inf(x)}x^+a^{\star}\Delta^{-cl(a)} \quad \underline{X = \Delta^{(cl(a)-inf(x))}x'} = \tau^{cl(a)+inf(x)}(a)\tau^{cl(a)}(x^+a^{\star})$$

Thus the right thing to do:

$$\begin{cases} m := RightM(X, cl(x), cl(a), cl(a) + inf(x) \mod 2, inf(x) \mod 2) \\ d := r\,(RightM(r(X), cl(x), cl(a), cl(a) \mod 2, inf(x) \mod 2)^{\star}) \end{cases}$$

As x is known, $inf(x)$ is known too. In the choice of $x \in B_n$, it is not useful to take $inf(x) \neq 0$; indeed, the attack would work as well. Then, we consider from now on:

$$inf(x) = 0$$

• $a \in B_n$: Treat the parameter "a", considering $inf(x) = 0$. We reduce to $inf(a) \in \{0, 1\}$. Let $\Delta^{inf(a)}a_1a_2\cdots a_{cl(a)}$ be the *left-greedy decomposition* of $a = \Delta^{inf(a)}b \in B_n$ where $inf(a)$ is supposed unknown. The element a is a secret corresponding to the conjugacy problem instance $(x, x' = axa^{-1})$. From Property 1 (i), Δ^2 commutes with any n-braid, thus $\Delta^{\varepsilon}b$ where $\varepsilon = inf(a) \mod 2$ is also a secret for the same instance.

$$x' = axa^{-1} = \Delta^{\varepsilon}bxb^{\star}\Delta^{-\varepsilon-cl(a)} \qquad X = \Delta^{cl(a)}x' = \tau^{cl(a)+\varepsilon}(bxb^{\star})$$

We notice that $\forall c \in B_n^+, \quad \tau(c)^{\star} = \tau(c^{\star})$, then we begin by computing a multiple and a divisor of $\tau^{\varepsilon}(b)$:

$$\begin{cases} m(\tau^{\varepsilon}(b)) := RightM(cl(x), X, cl(a), cl(a) \mod 2, 0) \\ d(\tau^{\varepsilon}(b)) := r\,(RightM(cl(x), r(X), cl(a), cl(a) \mod 2, 0)^{\star}) \end{cases}$$

In practice, ε is unknown, but this method gives valuable information:

$$(\tau^{\varepsilon}(b))^{-1}x'\tau^{\varepsilon}(b) = \begin{cases} x & \Leftrightarrow \varepsilon = 0 \ \ (inf(a) \text{ is even}) \\ \tau(x) & \Leftrightarrow \varepsilon = 1 \ \ (inf(a) \text{ is odd}) \end{cases}$$

So, the only drawback is that the work to complete this attack and to find $\tau^{\varepsilon}(b)$, should be made “twice” (for each above case). Then we also obtain ε: *e*e is not a parameter of security.

Remark 9 We note that other attacks can come in concurrence with this one. Applying this attack on the conjugacy instance (x, axa^{-1}), we have:

$$(d, m) \in B_n \quad \text{such that} \quad d \prec a \prec m$$

Thus, the initial conjugacy instance (x, x') on the secret a can be reduced to the following instances:

$$\begin{array}{ll} \text{either } (x, d^{-1}x'd) & \text{on the secret } d^{-1}a \\ \text{either } (x, m^{-1}x'm) & \text{on the secret } m^{-1}a \end{array}$$

In both cases, we transform a conjugacy instance into a new conjugacy instance where the secret is a shorter positive word. In practice, $cl(x) = cl(\tau(x)) \approx cl(d^{-1}x'd)$, therefore it is not useful to iterate this attack, but we can use other existing cryptanalysis.

4.3.3. Variant with the simultaneous conjugacy problem

The conjugacy application is a morphism. Thus, the data of the conjugacy problem instance (x, x'), allows to define a set of instances with the same secret. If we apply our attack to each instances, we get:

$$\left\{(x^i, x'^i) = (x^i, ax^ia^{-1}), \quad i \in \mathbb{Z}\right\} \quad \Rightarrow \quad \{(d_i, m_i)_{i\in\mathbb{Z}} \in \left(B_n^{+2}\right)^{\mathbb{Z}}; \ \forall i \in \mathbb{Z}, \ d_i \prec a \prec m_i\}$$

$$d := \bigvee_{i\in\mathbb{Z}} d_i \quad \text{and} \quad m := \bigwedge_{i\in\mathbb{Z}} m_i \tag{9}$$

Then, (9) gives a multiple and a divisor of a, with a better precision. In practice, it is sufficient to compute (d_i, m_i) only for few positive and negative values of index.

This remark can be adapted to the *Simultaneous Conjugacy Problem*:

$$\text{Let } \left(x_i, x_i' = ax_ia^{-1}\right)_{i\in I} \quad \text{known}$$

Then, we can define the new following instances:

$$\left\{\left(y, y' = aya^{-1}\right), \quad \forall y \in \langle x_i, \ i \in I\rangle\right\} \tag{10}$$

We have used these remarks in our implementation to obtain better results!

5. Analysis of the attack and exploitation

The aim of this section is to evaluate the efficiency of the presented attack and, in particular, to introduce the remaining search necessary to find the secret. We can measure the weakness of one key and determine an heuristic in the choice of the secret.

This attack allows to determine one right multiple and one left divisor of the secret of a conjugacy instance, $(x, x' = axa^{-1})$: we get (d, m) such that $d \prec a \prec m$ (recall: $l()$ refers to the Artin length and $inf(a) = inf(x) = 0$, see previous section !)

The knowledge of d and m allows to give some information about the remaining secret, see the Fig. 1:

$$\begin{array}{rl} \textit{absolute knowledge}: & ak(a) = min\left(l(d^{-1}a), l(a^{-1}m)\right) \\ \textit{relative knowledge}: & rk(a) = l(d^{-1}m) \end{array} \tag{11}$$

The *absolute knowledge* is the number of generators to complete the secret, lacking in d or in excess in m. Thus, the *absolute knowledge* is a practical upper bound for the supplementary exhaustive attack.

Remark 10 The *"absolute knowledge"* is not always known. If the Positive Artin Random Generator is used then $l(a)$ and $ak(a)$ are known. However, if it is the Canonical Random Generator which is used, then $cl(a)$ is known but $l(a)$ is not supposed to be known and we can not compute $ak(a)$. We can always compute $rk(a)$, therefore $ak(a)$ remains a practical limit whereas an upper bound for the size is given by

$$ak(a) \leq \frac{rk(a)}{2}$$

Recall that the complexity of the *left greedy decomposition* is polynomial and this calculus is very efficient. Thus, we do not include the complexity of the *test of identity* into the complexity of the supplementary attack. The logarithmic complexity expression of the *rough completion* to find exactly the secret is:

$$\begin{array}{lcl} \textbf{CSP} & \longrightarrow & \log\big(min\big(n-1, rk\big)\big) * ak + 1 \\ \textbf{GCP} & \longrightarrow & \log\big(min\big(\frac{n}{2}-1, rk\big)\big) * ak + 1 \end{array} \tag{12}$$

It lacks $ak(a)$ generators to complete the secret and we can choose them in two sets:

- $\{\sigma_i, i \in [1, n-1]\}$
- set of generators belonging to the decomposition of $d^{-1}m$

The "+1" comes from Sect. 4.3: the fact that we do not know $inf(a)$ compel us to make twice the completion. We do not know evaluate the complexity of *absolute knowledge*. Indeed, the algorithm is deterministic and polynomial, but we do not see how to give a theoretical expression of its efficiency. Next, we propose some experimentation to its qualities and its absences.

5.1. Efficiency on current problems: CSP, SCP, GCP

In this section we consider the impact of the attack on few conjugacy problems. The random generators used are the Canonical Generator and the Positive Artin Random Generator. The given numbers are mean values for 1000 attacks of instances. It is important to notice that this value only refer to two precise random generators and that this average give only global data. Even if one protocol seems to be broken, some conjugacy instances can be strong and this attack can to determine it.

Conjugacy Search Problem and Simultaneous conjugacy problem Tables 2 and 3

We present in the same times both problems; the number of instances defines that: 1 for the CSP, 2 or 3 for the SCP.

We note that the attack is very efficient on an instance generated with the Canonical Random Generator. From a n-braid of several thousand of generators, it gives a divisor and a multiple with a precision of few generators. In the case of the Positive Artin Random Generator, the search after the multiple fails but the one after the divisor is again of high quality. In case of the *simultaneous conjugacy problem*, for both random generators, the attack can be considered total with only few instances.

Table 2 Attack on the SCP using the Canonical Random Generator

n of B_n	$cl(a) = cl(x)$	SCP	$l(a)$	$l(d^{-1}a)$	$l(a^{-1}m)$	$ak(a)$	Rough
20	3	1	285.7	16.2	16.2	11.2	48.6
		2		3.0	3.1	1.6	5.2
		3		0.8	0.8	0.3	1.2
60	9	1	7969.0	26.0	25.4	20.3	116.4
		2		3.7	3.8	2.5	8.3
		3		0.8	0.9	0.4	1.3
100	15	1	37163.1	36.8	36.7	30.7	191.3
		2		5.5	5.3	3.7	13.7
		3		1.2	1.2	0.6	1.8

Table 3 Attack on the SCP using the Positive Artin Random Generator

n of B_n	$l(a) = l(x)$	SCP	$l(d^{-1}a)$	$l(a^{-1}m)$	$ak(a)$	Rough
20	500	1	8.0	103.4	8.0	35.0
		2	2.7	32.7	2.5	11.6
		3	1.2	16.0	0.9	4.8
60	1000	1	22.5	712.0	22.5	133.4
		2	7.5	352.1	7.5	45.1
		3	3.1	139.3	3.0	18.6
100	1500	1	37.0	1947.3	37.0	246.3
		2	12.4	5738.1	12.4	83.2
		3	5.0	4614.3	5.0	34.1

Table 4 Attack on the GCP using the Canonical Random Generator

n	$cl(a) = cl(x)$	$l(a)$	$l(d^{-1}a)$	$l(a^{-1}m)$	$ak(a)$	Rough
40	3	257.2	9.5	9.9	6.4	28.2
120	9	7708.3	22.0	21.5	17.3	95.2
200	15	36372.5	34.3	33.5	28.3	173.2

Table 5 Attack on the GCP using the Positive Artin Random Generator

n	$l(a) = l(x)$	$l(d^{-1}a)$	$l(a^{-1}m)$	$ak(a)$	Rough
40	500	7.7	95.5	7.7	33.7
120	1000	22.2	725.8	22.2	131.6
200	1500	36.7	2288.8	36.7	244.3

Generalized Conjugacy Problem Tables 4 and 5

This problem includes the *Generalized Conjugacy Problem of the type Diffie–Hellman*. It is a fundamental problem for braid based cryptography. The presented algorithm works in the braid group B_n; however on the one hand, we can easily generalize it to this problem and consider that $a \in BL_{\lfloor \frac{n}{2} \rfloor}$. On the other hand, we can solve the Generalized Conjugacy Problem with the same algorithm: it is enough to consider "$a \in B_n$" and afterwards to make:

$$d := d \wedge \Delta_{BL_{\lfloor \frac{n}{2} \rfloor}}^{cl(a)} \qquad m := m \wedge \Delta_{BL_{\lfloor \frac{n}{2} \rfloor}}^{cl(a)} \tag{13}$$

Remark 11 To attack the CSP, we need the canonical length of the secret. For the GCP, we require in addition the infimum of the secret in $BL_{\lfloor \frac{n}{2} \rfloor}$. See Sect. 6.

The remarks about the attack of the *Conjugacy Problem* and the *Simultaneous Conjugacy Problem* apply again here.

5.2. Explanation of these results

One applies Remark 5 to the product of two *canonical factors*, $c, d \in S_n$: the *left greedy* of $b = cd$ gives a larger first factor and a shorter second. The more c is complex, the less the first factor of the *left greedy* product of b increases. It is exactly what has happened and that explains the difference in the results.
Consider the three random generators (with $inf(a) = 0$):

- CRG: A random draw of a canonical factor is, in fact, a random draw of a n-permutation which are transformed into a canonical factor. The *Artin length* of each canonical factor of the n-braid a, obtained from the CRG, is $\approx \frac{l(\Delta)}{2}$. It is a n-braid sufficiently complex and we get, from $X = \tau^{\alpha}(a)y\tau^{\alpha+\beta}(a^{\star})$, a **small right multiple**.
  Moreover the remark about the length of the canonical factors of a, is valid for $a^{\star}$. We have:
  $$\forall c \in S_n, \quad l(\partial(c)) = l(\Delta) - l(c) \qquad \text{and} \qquad l(a^{\star}) + l(a) = l(\Delta) * cl(a) \tag{14}$$
  Thus, the left multiple of $a^{\star}$ obtained with $r(X)$ is good too and gives a **large left divisor** of a.
- PARG: In practice, an element generated with the PARG has a large *canonical length* with regard to its *Artin length*. Then the *Artin length* of each of their canonical factors is $\ll \frac{l(\Delta)}{2}$, a is "sparce" what gives a **bad right multiple** from X: the normalization modifies much the first factors.
  However the length of factors of $a^{\star}$ is now $\gg \frac{l(\Delta)}{2}$; that gives a good left multiple of $a^{\star}$ and a **large left divisor** of a.
- ARG: We notice that in this case $l(a) \approx l(a^{\star}) \approx \frac{l(\Delta)*cl(a)}{2}$. This case seems similar to CRG, however the distribution of lengths of canonical factor is not uniform: the length of first factors is $\gg \frac{l(\Delta)}{2}$ whereas this one of last factors is $\ll \frac{l(\Delta)}{2}$; moreover, there are not a lot of factors having for length $\approx \frac{l(\Delta)}{2}$. That added to the fact that the size of the secret is large, gives a **bad multiple**. This explanation applies to the divisor, considering $a^{\star}$. We get a **bad divisor** too. See Table 6.

Table 6 Attack on the CSP and GCP using the Artin Random Generator

n	$l(a) = l(x)$	Problem	$l(a_{lgd})$	$cl(a_{lgd})$	$l(d^{-1}a)$	$l(a^{-1}m)$	$ak(a)$
20	500	CSP	3829.6	40.0	626.3	642.4	533.5
40	500	GCP	3768.4	40.0	557.9	719.5	518.2
100	1500	CSP	73134.8	29.3	39980.2	30188.4	35884.1
200	1500	GCP	72021.3	29.1	33583.4	41679.7	32462.4

(a_{lgd} means the n-braid a is transformed in its *left greedy decomposition*)

Table 7 Means on the *absolute knowledge* for the GCP

	CRG	$cl(a) = cl(x)$				PARG	$l(a) = l(x)$			
		4	10	40	Rough		500	1000	1500	Rough
	40	7.1	7.8	7.8	33	40	7.7	7.5	7.8	33
	80	13.2	13.0	12.4	67	80	15.0	14.3	15.1	80
n	120	17.1	17.1	17.4	95	120	21.9	22.0	22.5	130
	200	28.7	29.1	29.2	178	200	35.2	36.7	38.0	245
	400	59.4	58.8	57.5	415	400	67.0	69.9	72.2	535

Table 8 Means on the *absolute knowledge* for the GCP with n=100

	CRG	$cl(x)$				PARG	$l(x)$		
		4	10	40			500	1000	1500
	4	14.7	15.0	15.4		500	17.5	19.3	18.4
$cl(a)$	10	15.5	14.8	15.0	$l(a)$	1000	18.4	17.9	18.7
	40	15.0	14.5	14.4		1500	19.0	18.1	18.4

5.3. Complement on the efficiency and Conjecture

In this section, we show how behave the algorithm when we modify the input variables. The results of the following simulations are given for 200 attacked consecutive instances.

We consider generalized conjugacy instances; from Table 7, it seems that the size of the key is not a parameter of security; only the size of the group, designated by the *braid index*, seems to have an effect upon the results of the attack. The relative size between the secret key, "a", and the public key, "x", does not seem to reduce the efficiency, see Table 8.

The results obtained for conjugacy instances are similar. Since we can not prove this analysis, we formulate the following conjecture:

Conjecture 1 *We consider a conjugacy instance or a generalized conjugacy instance using the Canonical Random Generator or the Positive Artin Random Generator. The efficiency of the presented attack does not depend, on average, on the* (*relative*) *size of the keys but only on the size of the group: the braid index.*

5.4. Estimates of the fine complexity of the completion

To measure the efficiency of this attack, we propose an estimate of the fine complexity of the supplementary exhaustive attack. Consider the case of the Conjugacy Search Problem. Let (d, m) be the result of the attack on the instance (x, x') with secret a. We have given the logarithmic complexity expression of the rough completion:

$$\log\Big(min\big(n-1, rk\big)\Big) * ak + 1$$

However, this formula does not take the non-abelian aspect into account. Therefore, we can introduce a set to reduce the space of possible candidates:

$$I_{d^{-1}m,ak} = \big\{p\ ;\quad l(p) \leq ak(a) \quad \text{and} \quad \big(\, p \prec d^{-1}m \quad \text{or} \quad p \prec^{r} d^{-1}m\,\big)\big\} \qquad (15)$$

The logarithmic complexity expression of the *fine completion* is $log\big(\#(I_{d^{-1}m,ak})\big)$

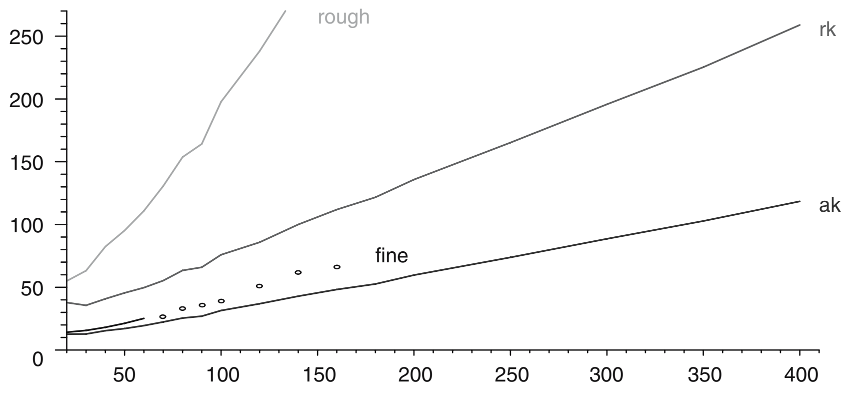

Fig. 3 $ak(n)$, $rk(n)$, $fine(n)$ and $rough(n)$ for the CSP using the CRG

In fact, knowing $d \prec a \prec m$, $I_{d,m,ak}$ is the set of all of n-braids p, such that p could complete d in $a = dp$ or such that p could reduce m in $a = mp^{-1}$. It is easy to enumerate elements of this set: it is enough to use the lexicographic order, see [17]. The complexity is clearly exponential but that gives a better result than (12). This method is all the more efficient if we have a large left divisor and a small right multiple. The Fig. 3 is obtained with 500 simulations for each point:

Conjecture 2 *We consider a conjugacy instance using the Canonical Random Generator, under Conjecture* 1*, on average ak and rk are linear in n.*

Remark 12 Using the Canonical Random Generator, we note that the braid $d^{-1}m$ is often quite sparse ($rk \leq \frac{n}{4}$ for the GCP and $rk \leq \frac{n}{2}$ for the CSP) and then its writing "commute": we note that the generators belonging to this word commute easily (the difference between their index is greater than 1). We can approximate $\#(I_{d^{-1}m,ak})$ by C_{rk}^{ak}.

In practice we have:

$$log\big(\#(I_{d^{-1}m,ak})\big) < log(C_{rk}^{ak}) \qquad (16)$$

5.5. Evaluation of existing protocols

We simulate our attack on the different proposed protocols in the literature, see Tables 9 and 10. Many are seriously weakened or broken. We consider that if the logarithmic complexity is less than 60, then the protocol is broken. If it is between 60 and 80 then the protocol is unsecured. Lastly, if it is upper than 80, then the protocol is secure.

Table 9 Attack of existing protocols using the PAR Generator

Ref	Problem	n	$l(a)$	$l(x)$	$ak(a)$	Rough
[2]	SCP_{20}	80	500	5	10.8	69.3
		80	1000	10	1.1	7.9
[7]	CSP	50	1000	1000	19.0	108.2
[25]	CSP	30	1000	1000	11.8	58.9
	GCP	60	1000	1000	11.3	56.4

Table 10 Attack of existing protocols using the CR Generator

Ref	Problem	n	$cl(a)$	$cl(x)$	l(a)	rk(a)	$ak(a)$	Rough	Fine
[4]	GCP	100	15	15	8818.1	37.4	14.4	76.2	20.2
		150	20	20	26996.7	52.7	21.3	122.8	≈ 29
		200	30	30	72735.7	67.2	28.5	174.0	≈ 38
		250	40	40	152517.4	84.7	35.9	230.9	≈ 58
[19]	GCP	50	5	3	686.9	23.7	8.2	38.4	11.9
		90	12	10	5669.2	35.0	13.3	69.2	18.8
[25]	CSP	30	15	15	3264.4	37.0	13.2	65.1	17.3
	GCP	60	15	15	3043.6	25.9	9.3	44.7	13.7
[18][a]	SCP_2	20	4	3	379.7	6.5	1.7	5.6	6
		24	4	3	551.8	6.1	1.7	5.4	6
		28	4	3	752.1	5.7	1.7	5.3	6

[a]We do not use the *Random Super Summit Braid Generator* but only the CRG.

The given numbers are mean values for 1000 simulations of attacked instances. It is important to understand that the value are given on average. A protocol supposed unreliable or broken can have some instance sufficiently secure; and a reliable protocol can have some instance unreliable. To know the strength of an instance, it is enough to apply this attack to the instance and to estimate its fine complexity, see Sect. 6.2 (Table 10).

6. Developments

6.1. Canonical length of the secret and management of the GCP

The presented algorithm requires the knowledge of the canonical length of the secret. Here, we give a lower bound. Let $(x = \Delta^{inf(x)}x^+, x' = axa^{-1})$ a conjugacy instance with $a \in B_n$ then we have:

$$cl(a) \geq max\left(inf(x) - inf(x'), sup(x') - sup(x)\right) \tag{17}$$

Proof of (17) Considering x' or $\tau(x')$, we can supposed that $inf(a) = 0$.

$$x' = \Delta^{inf(x)-cl(a)} \underbrace{\tau^{-sup(a)}\left(\tau^{inf(x)}(\ \Delta^{-inf(a)}a\ )\ \Delta^{-inf(x)}x\ a^{\star}\right)}_{\in B_n^+ \text{ and } sup(")\leq cl(a)+cl(x)+cl(a^{\star})} \Rightarrow inf(x') \geq inf(x) - cl(a)$$

Moreover $sup(x') \leq (inf(x) - cl(a)) + cl(a) + cl(x) + cl(a^{\star})$; thus from Corollary 1 we have $sup(x') \leq sup(x) + cl(a)$.

□

In practice, to guess exactly the *canonical length*, we can use a statistic study and try some values from the given relation.

Remark 13 Let L be the lower bound for the canonical length of (17). The algorithm, initialized with "$cl(a) = L$", gives a divisor of the secret as efficient as the one obtained by [15], for the presented random generators. Therefore, we do not need to know the canonical length of the secret to find a good divisor.

6.1.1. Case of the GCP*: infimum and canonical length of the secret*

We consider that the infimum and the canonical length of the secret in the sub-group $BL_{\lfloor \frac{n}{2} \rfloor}$ are not parameters of security. We propose a heuristic to estimate them, using the same value in the group B_n. In practice, we want to reverse the following table:

Let $a \in BL_{\lfloor \frac{n}{2} \rfloor}$,

in $BL_{\lfloor \frac{n}{2} \rfloor}$		Comment in B_n		
inf	sup	inf	cl	Structure
≥ 0	≥ 0	0	$sup(a)$	all factors are in $BL_{\lfloor \frac{n}{2} \rfloor}$
≤ 0	≥ 0	$inf(a) \mod 2$	$cl(a)$	the $\lvert inf(a) \rvert$ first factors are in B_n the $sup(a)$ last factors are in B_n
≤ 0	≤ 0	$inf(a) \mod 2$	$-inf(a)$	all factors are in B_n

A heuristic to retrieve the data in $BL_{\lfloor \frac{n}{2} \rfloor}$ is:

```
Input:   $r$ and $p$ the infimum and the canonical length of $a$ in $B_n$.
---------------------------------------------------------------------
         let $(d, m)$ result of attack on $(x, x')$ in $B_n$
         If $r = 0$ and $d \in BL_{\lfloor \frac{n}{2} \rfloor}$ then
             $r' \in \left[ max\{k;\, \Delta^k_{BL_{\lfloor \frac{n}{2} \rfloor}} \prec d\}, max\{k;\, \Delta^k_{BL_{\lfloor \frac{n}{2} \rfloor}} \prec m\} \right]$
             $p' := p - r'$
         EndIf
         If $r'$ and $p'$ are not defined then $r' := r$
             While $d \notin BL_{\lfloor \frac{n}{2} \rfloor}$ do
                 $r' := r' - 2$
                 Let $(d, m)$ result of attack on $(x, \Delta^{-r'}_{BL_{\lfloor \frac{n}{2} \rfloor}} x' \Delta^{r'}_{BL_{\lfloor \frac{n}{2} \rfloor}})$
             EndWhile
             $p' := \sup(m)$
         EndIf
---------------------------------------------------------------------
Output:  $r'$ and $p'$ the infimum and the canonical length of $a$ in $BL_{\lfloor \frac{n}{2} \rfloor}$.
```

To conclude, for this attack, we consider that a generalized conjugacy instance is not stronger than a conjugacy instance.

6.2. Test of weak key

To test a key, we propose to use the presented algorithm. Let (x, axa^{-1}) a conjugacy instance in B_n; we suppose that x, $cl(a)$ and $inf(a)$ are known. To evaluate the security of the key a, we determine (d, m) such that $d \prec a \prec m$ with the following process:

$$\begin{cases} m := RightM(\Delta^{cl(a)-inf(x)}x', cl(x), cl(a), inf(x) - cl(a) - inf(a), -inf(x)) \\ d := r\left(RightM(r(\Delta^{cl(a)-inf(x)}x'), cl(x), cl(a), -cl(a) - inf(a), inf(x))^\star\right) \end{cases}$$

Next, we compute $ak(a) = min\big(l(d^{-1}a), l(a^{-1}m)\big)$ and $rk(a) = l(d^{-1}m)$. We recommend $ak > 50$ and $rk > 150$. In practice, to have more precision, it is possible to make a table given the average fine complexity in terms of ak and rk, (15).

6.3. New (canonical) random generator

From Sect. 5.2, we note that three rules must be respected to have a good random generator of braids:

1. the properties required for the secret a, must be required also for $a^\star = a^{-1}\Delta^{sup(a)}$
2. $l(a) \approx l(a^\star) \approx \frac{l(\Delta)cl(a)}{2}$
3. the *Artin length*, $l(\,)$, of the first factors must be large $\left(\gg \frac{l(\Delta)}{2} = \frac{n(n-1)}{4}\right)$, whereas the one of the last factors must be short $\left(\ll \frac{l(\Delta)}{2}\right)$.

In the majority of schemes, it is preferable to have a random generator of normalized braid. Then we propose the new canonical random generator:

```
Input:   $L$ an integer
         $a := \mathrm{ARG}(k)$, $k$ chosen large such that $cl(a) \gg L$
         Compute the left-greedy decomposition $\Delta^{inf(a)} a_1 a_2 \cdots a_l$ of $a$
         $\alpha := min(i\ ;\ \forall j > i \quad l(a_j) < \frac{l(\Delta)}{2})$
         $\beta := min(\alpha + \lfloor \frac{L}{2} \rfloor, cl(a))$
         Return $a_{\beta-L+1} \cdots a_{\beta-1} a_\beta$
Output:  a braid with a canonical length $L$ verifying the three conditions
```

Recall: ARG is the *Artin Random Generator* defined in Sect. 2.1.

This new random generator is adapted to existing protocols and seems to bring more security with keys having a smaller size. Using it, the presented attack produces a bad divisor and a bad multiple, as expected. Moreover, it prevents also the attack of Hofheinz and Steinwandt (see [14]); indeed, in practice, the canonical length of the remaining secret is more than 2. To conclude, it seems to be faithful to considerations of Sibert and Ko (see [24]).

We recommend for this random generator the following parameters:

– Conjugacy Search Problem: B_{30} and canonical length 8 or 10.
– Generalized Conjugacy Problem: B_{50} and canonical length 8 or 10.

6.4. Consequence of this study on the *Decomposition Problem*

The presented attack does not apply to the Decomposition Problem. The conjugacy structure is necessary. However, Property 5 and Corollary 3 let us to find some indications.

Let $(a, b, x, y) \in B_n^4$ such that $a = \Delta^{inf(a)} a^+,\ b = \Delta^{inf(b)} b^+,\ x = \Delta^{inf(x)} x^+$ and $y = axb$

We can develop the decomposition of y

$$Y := \Delta^{-inf(a)-inf(x)-inf(b)} y = \tau^{inf(x)+inf(b)}(a^+)\tau^{inf(b)}(x^+)b^+$$

We assume know x, $cl(a)$, $cl(b)$, $inf(a)$ and $inf(b)$. Let $y_1 y_2 \cdots y_l$ and $z_1 z_2 \cdots z_l$ be the *left greedy* product of Y and $r(Y)$, Property 5 and Corollary 3 give a right multiple and a left divisor of a, and a left multiple and a right divisor of b (Recall τ is an involution):

$$\Delta^{inf(a)}\tau^{inf(x)+inf(b)}\left(r\left(z_{cl(b)+cl(x)+1} \cdots z_{l-1} z_l\right)\right) \prec a \prec \Delta^{inf(a)}\tau^{inf(x)+inf(b)} \times \left(y_1 y_2 \cdots y_{cl(a)}\right) \qquad (18)$$

$$\Delta^{inf(b)} y_{cl(a)+cl(x)+1} \cdots y_{l-1} y_l \prec^r b \prec^r \Delta^{inf(b)} r(z_1 z_2 \cdots z_{cl(b)}) \tag{19}$$

With a good random generator, these relations are quite useless; thus the *Decomposition Problem* is not endangered by this method.

7. Conclusion

The presented test gives some valuable information on the complexity of the secret. Its efficiency on certain random generators emphasizes the important part played by latters in security. We propose new criteria to produce a strength key and apply them in a proposal of a new random generator. It seems to answer a current problem and may replace (canonical) random generator used so far. It should be studied further.

For the new (canonical) generator, we can consider instance for B_{50} and a canonical length of 8 for the *Generalized Conjugacy Problem*; whereas we recommend $n > 250$ for the classical canonical random generator.

The conjugacy problem remains an important part in braid based cryptography. Nevertheless, we saw that at least two points can be developed. The first is the use of the *Decomposition Problem* that is almost not weakened. The second is to use reduced forms, which are adapted for the Artin Random Generator.

Acknowledgements The author would like to thank François Arnault, Thierry Berger, Philippe Gaborit and Marc Rybowicz for their valuable comments.

References

1. Anshel I, Goldfeld D (1999). An algebraic method for public-key cryptography. Math Res Lett 6: 287–291
2. Anshel I, Fischer B, Goldfeld D (2001). New key agreement protocols in braid group cryptography, RSA 2001, LNCS 2020, 1–15
3. Artin E (1947). Theory of braids. Anna Math 48:101–126
4. Cha JC, Ko KH, Lee SJ, Han JW, Cheon JH (2001). An efficient implementation of braid groups, Asiacrypt 2001, LNCS 2248, pp 144–156
5. Cheon JH, Jun B (2003). A polynomial time algorithm for the braid Diffie–Hellman conjugacy problem. Crypto 2003
6. Dehornoy P (1997). A fast method for computing braids. Adv Math 125(2):200–235
7. Dehornoy P (2004). Braid-based cryptography. Contemp Math Amer Math Soc 360:5–33
8. Elrifai EA, Morton HR (1994). Algorithms for positive braids. Quart J Math Oxford 45(2):479–497
9. Epstein D, Cannon J, Holt D, Levy S, Patterson M, Thurston W (1992). Word processing in groups. Jones et Barlett Publishers, Boston, chapter 9
10. Franco N, Gonzalez-Meneses J (2003). Conjugacy problem for braid groups and Garside groups. J Algebra 266(1):112–132
11. Garber D, Kaplan S, Teicher M, Tsaban B, Vishne U (2002). Length-based conjugacy search in the Braid group. http://arXiv.org/abs/math.GR/0209267
12. Garside FA (1969). The braid group and other groups. Quart J Math Oxford 20–78, 235–254
13. Gebhardt V (2003). A new approach to the conjugacy problem in Garside groups to appear in J Algebra: http://arXiv.org/abs/math.GT/0306199
14. Hofheinz D, Steinwandt R (2003). A practical attack on some braid group cryptographic primitives, PKC 2003, LNCS 2567, pp 187–198
15. Hughes J (2002). A Linear Algebraic Attack on the AAFG1 Braid Group Cryptosystem, ACISP 2002, LNCS 2384, 02, pp 176–189
16. Hughes J (2000). Allen Tannenbaum, Length-based attacks for certain group based encryption rewriting systems, Inst. for Math. and its Applic. Minneapolis

17. Jacquemard A (1990). About the effective classification of conjugacy classes of braids. J Pure Appl Algebra 63(2):161–169
18. Ko KH, Choi DH, Cho MS, Lee JW (2002). New Signature Scheme Using Conjugacy Problem: http://eprint.iacr.org/2002/168/
19. Ko KH, Lee SJ, Cheon JH, Han JW, Kang JS, Park C (2000). New public-key cryptosystem using braid groups, Crypto 2000, LNCS 1880, pp 166–184
20. Lee SJ, Lee EK (2002). Potential weakness of the commutator key agreement protocol based on braid groups, Eurocrypt 2002, LNCS 2332, pp 14–28
21. Lee E, Park JH (2003). Cryptanalysis of the public-key encryption based on braid groups, Eurocrypt 2003, LNCS 2656
22. Maffre S (2005). Reduction of conjugacy problem in braid groups, using two Garside structures, WCC 2005, pp 214–224
23. Michel J (1999). A note on words in braid monoids. J Algebra 215:366–377
24. Sibert H (2003). Algorithmique des groupes de tresses, Ph.D. Lab. LMNO
25. Sibert H, Dehornoy P, Girault M (2003). Entity authentification schemes using braid word reduction, WCC 2003, pp 153–163