Entity Authentication Schemes Using Braid Word Reduction

Hervé SIBERT, Patrick DEHORNOY, and Marc GIRAULT

Abstract. Artin's braid groups currently provide a promising background for cryptographical applications, since the first cryptosystems using braids were introduced in [**22, 2, 3, 18**]. A variety of key agreement protocols based on braids have been described, but few authentication or signature schemes have been proposed so far. We introduce three authentication schemes based on braids, two of them being zero-knowledge interactive proofs of knowledge. Then we discuss their possible implementations, involving normal forms or an alternative braid algorithm, called handle reduction, which can achieve good efficiency under specific requirements.

1. Introduction

In the recent years, beginning with [**22**], several authors proposed to build secure cryptographical schemes using noncommutative groups, in particular Artin's braid groups [**2, 3, 18, 6, 19**], a natural idea as, on the one hand, braid groups are more complicated than Abelian groups, but, on the other hand, they are not too complicated to be worked with. In particular, the conjugacy problem in braid groups is algorithmically difficult, and it consequently provides one-way functions.

The aim of this paper is twofold. Firstly, we propose three authentication schemes designed for braid groups: the braid schemes considered so far dealt with key exchange and confidentiality, thus not providing means of authentication. Secondly, we propose a new way of implementing braid operations, namely using braid words and the so-called handle reduction algorithm. Not only is such an implementation very efficient in practice, but it is also well suited for the schemes we shall describe, and, more generally, for all braid schemes where using unique representatives of the braids is not necessary. At this stage, we mainly wish to draw the attention of researchers on the potentialities of that method.

The paper is organized as follows. In Section 2, we state the difficult problems based on braid groups that we shall rely on. In Section 3, we describe our braid-based authentication schemes and list the security requirements to be achieved to make them secure. In Section 4, we present different classical ways of computing with braids, and analyze the security of the associated implementations of our protocols. In Section 5, we introduce a second, radically new approach consisting in using non-normal braid words together with a reduction algorithm, and we discuss its efficiency and security.

Key words and phrases. braid group, authentication, zero knowledge, handle reduction.

2. Difficult braid problems

2.1. Braid groups. For $n \geqslant 2$, Artin's braid group B_n is defined to be the group with presentation

$$\langle \sigma_1, \dots, \sigma_{n-1} \,;\, \sigma_i\sigma_j = \sigma_j\sigma_i \text{ for } |i-j| \geqslant 2,\ \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j \text{ for } |i-j| = 1 \rangle. \tag{2.1}$$

We refer the reader to any textbook about braids, for instance [**8**], for a geometrical interpretation of each element of the group B_n by an n-strand braid in the usual sense: the idea is that every n-strand braid diagram can be encoded in a word in the letters $\sigma_i^{\pm 1}$ by slicing it into a concatenation of elementary diagrams with one crossing, and using σ_i for the diagram where the ith strand crosses over the $(i+1)$st one. Then, the relations of (2.1) correspond to the notion of ambient isotopy, as shown by E. Artin in [**4**]. Notice that, when a braid b is specified by a word in the letters $\sigma_i^{\pm 1}$, the sum of the exponents ± 1 in the word is not modified when the braid relations of (2.1) are applied. This sum, which depends only on the braid itself, is called the *exponent sum* of b, and denoted by $e(b)$.

The first important point for our current purpose is that the braid groups are infinite noncommutative groups which are eligible for practical computations: there exist in particular efficient ways of specifying a braid and of computing with braids (see Section 4 below).

The second important point is the existence of difficult braid problems for which no feasible solution is known. We use several of them in the subsequent sections.

2.2. Conjugacy problems. One says that a braid b' is a *conjugate* of a braid b if we have $b' = sbs^{-1}$ for some braid s. The *conjugacy problem* is the question of algorithmically recognizing whether two given braids b, b' are conjugate or not. This problem has been proved to be decidable by Garside [**15**], but the algorithm he proposes, as well as all improvements proposed so far [**10, 13**], have a high cost, exponential in the length of the considered words and the number of strands.

The following related and even more difficult question is used in [**18**]:

Conjugacy Search Problem: *Assuming that the braid b' is a conjugate of the braid b, find a witness, i.e., find s satisfying $b' = sbs^{-1}$.*

In the state-of-the-art, it is considered infeasible to solve Conjugacy Search Problem for sufficiently large braids [**17**]. More precisely, for a generic braid b, $s'bs'^{-1} = sbs^{-1}$ is true only if the braid $s^{-1}s'$ belongs to the centralizer of the braid b. The latter depends on b, but it is known to be generically very small, namely to be generated by b and the unique additional central braid Δ_n^2 of [**15**]. Hence the probability for a random conjugate of b to be equal to b' is negligible. We will use a variant of this problem, when s lies in some subgroup of B_n.

We now introduce the

Diffie-Helmann-like Conjugacy Problem: *Given a braid b in B_n, and the braids $b' = sbs^{-1}$ and $b'' = rbr^{-1}$, where s and r lie in two subgroups of B_n that commute one with the other, find the braid $sb''s^{-1}$ $(= rb'r^{-1})$.*

The connection between this problem and the subgroup variant of the Conjugacy Search Problem is similar to the one between the Diffie-Helmann Problem and the Discrete Log Problem. The Diffie-Helmann-like Conjugacy Problem is obviously reducible to the Conjugacy Search Problem, but we assume that it is as hard. Hence checking that the Conjugacy Search Problem is hard for b is supposed to ensure that the Diffie-Helmann-like Conjugacy Problem also is.

2.3. Root extraction problems. The braid groups are torsion-free, *i.e.*, if b is a non-trivial braid, then b^2, and, more generally, b^e for every $e \geqslant 2$, is not trivial.

ROOT PROBLEM (for exponent e): *Assuming that the braid b' is an e-th power in B_n, find an e-th root of b', i.e., find b satisfying $b^e = b'$.*

It is proved in [**23**] that ROOT PROBLEM is decidable (see also [**21**]), but the only known algorithm consists in explicitly enumerating several conjugacy classes related with the initial braid b', a process which is exponential in essence, and therefore infeasible when braids of a sufficient size are considered. In practice, ROOT PROBLEM appears as even more difficult than CONJUGACY SEARCH PROBLEM.

3. Three authentication schemes

We now present several public-key authentication schemes (also called identification protocols), *i.e.*, systems designed to authenticate an entity. Scheme I is a two-pass scheme and is perfectly honest-verifier zero-knowledge. The other two schemes are iterated three-pass protocols, and they are zero-knowledge in a theoretical infinite framework.

In the version described below, the schemes enable B(ob) to check that A(lice) knows the secret key. The schemes meet necessary security compliance: they ensure the confidentiality of the private key, and the probability of finding another key that will behave like the private one is negligible. The level of security of the schemes can be parameterized by modifying the size of the braid specifiers in use, in particular the braid index n.

3.1. Scheme I. In [**18**], a key agreement scheme is proposed, which is the braid group version of the Diffie-Hellman key agreement protocol. This enables the authors of [**18**] to construct a public-key cryptosystem, by (in essence) carrying out the key agreement scheme in order to calculate a common secret key, and subsequently Exclusive-Oring a hashed image (*i.e.*, an image by a so-called ideal hash function) of this key with the message to conceal.

But Diffie-Hellman based key agreement schemes also allow to construct two-pass (or "challenge-response") authentication schemes by (again in essence) carrying out the key agreement scheme, and verifying that the secret key computed at the two ends (or rather a one-way/hashed image of this secret key, in order to thwart active attacks) is actually the same. Our Scheme I is related to this approach.

Note that there is also a standard way to turn any encryption scheme into an authentication scheme, which consists in sending to Alice both an encrypted version and a hashed image of the same message m, then requesting her to reply with the deciphered message m (she will do it only if the hashed image of the deciphered message is the same as the one sent by Bob). Transforming that way the cryptosystem from [**18**] is possible but leads to a slightly more complicated protocol, also less flexible in terms of implementation since a normal form (see Section 4) is required.

Scheme I is based on the difficulty of DIFFIE-HELMANN-LIKE CONJUGACY PROBLEM. It uses the trick of [**18**] that braids involving disjoint families of strands commute. The data consist of a public key, which is a pair of braids, and of A's private key, also a braid.

We assume that n is even, and denote by LB_n (resp. UB_n) the subgroup of B_n generated by $\sigma_1, \ldots, \sigma_{n/2-1}$, *i.e.*, braids where the $n/2$ lower strands only are braided (resp. in the subgroup generated by $\sigma_{n/2+1}, \ldots, \sigma_{n-1}$). The point is that every element in LB_n commutes with every element in UB_n, and alternative subgroups with this property could be used instead. We assume that H is a fixed collision–free hash function from braids to sequences of 0's and 1's or, possibly, to braids (see Appendix D).

• Phase 1. **Key generation**

- (i) Choose a public braid b in B_n such that the DIFFIE-HELMANN-LIKE CONJUGACY PROBLEM for b is hard enough;

- (ii) A(lice) chooses a secret braid s in LB_n, her private key; she publishes $b' = sbs^{-1}$; the pair (b, b') is the public key.

• Phase 2. **Authentication phase**

- (i) B(ob) chooses a braid r in UB_n, and sends the challenge $x = rbr^{-1}$ to A;

- (ii) A sends the response $y = H(sxs^{-1})$ to B, and B checks $y = H(rb'r^{-1})$.

The following proposition states the security of our scheme, in particular regarding the observation of the communication by a third party.

PROPOSITION 3.1. *Scheme I is a perfectly honest-verifier ZK interactive proof of knowledge of s.*

PROOF (SKETCH). *Completeness.* Assume that, at Step 2(ii), Alice sent y'. Then Bob accepts Alice's key if and only if we have $y' = H(rb'r^{-1})$. The latter relation is equivalent to

$$y' = H(r(sbs^{-1})r^{-1}). \tag{3.1}$$

By hypothesis, the braid s lies in LB_n while r lies in UB_n, so $rs = sr$ holds and (3.1) is equivalent to $y' = H(s(rbr^{-1})s^{-1})$, *i.e.*, to $y' = y$.

Soundness. Assume a cheater A' is accepted with non-negligible probability. This means that A' can compute $H(rb'r^{-1})$ with non-negligible probability. As H is supposed to be an ideal hash-function, this means that A' can compute a braid z satisfying $H(z) = H(rb'r^{-1})$ with non-negligible probability. There are two possibilities: either we have $z = rb'r^{-1}$, which contradicts the hypothesis that the DIFFIE-HELMANN-LIKE CONJUGACY PROBLEM for b is hard, or $z \neq rb'r^{-1}$, which means that A' and B are able to find a collision for H, contradicting the hypothesis that H is collision-free.

Honest-verifier zero-knowledge. Consider the probabilistic Turing machine defined as follows: it chooses random braids r using the same drawing as the honest verifier, and outputs the instances $(r, H(rb'r^{-1}))$. Then, the instances generated by this simulator follow the same probability distribution as the ones generated by the interactive pair (A, B). □

For active attacks, the security is ensured by the hash function H: if H is one-way, these attacks are ineffective. We discuss possible choices for H in Appendix D.

3.2. Scheme II. We describe now two authentication schemes belonging to the family of zero-knowledge schemes [**12, 16**]; they consist in repeating a three-pass process several times so as to guarantee the required level of security. The first of these schemes is based only on the difficulty of the CONJUGACY SEARCH PROBLEM, thus avoiding existing attacks against the variants of the CONJUGACY SEARCH PROBLEM, namely against the DIFFIE-HELMANN-LIKE CONJUGACY PROBLEM and the Multiple Conjugacy Problem [**2**].

• Phase 1. **Key generation** (as in Scheme I):

- (i) Choose a public braid b in B_n, so that the CONJUGACY SEARCH PROBLEM for b is hard;

- (ii) A(lice) chooses a secret braid s in B_n, her private key; she publishes $b' = sbs^{-1}$; the pair (b, b') is the public key.

• Phase 2. **Authentication phase**: Repeat the following exchanges k times, with k a polynomial function of the size of the braid specifiers :

- (i) A chooses a random braid r, and sends $x = rbr^{-1}$ to B(ob);

- (ii) B sends a random bit ϵ to A;

- (iii) For $\epsilon = 0$, A sends $y = r$ to B, and B checks $x = yby^{-1}$;

- (iii') For $\epsilon = 1$, A sends $y = rs^{-1}$ to B, and B checks $x = yb'y^{-1}$.

REMARK 3.2. Scheme II can be used without change when the braid group B_n is replaced with another (finite or infinite) noncommutative group G in which the conjugacy search problem is difficult.

In the general framework of a group G—so, in particular, in the case of B_n—we have the natural notion of a right-invariant probability measure, namely a measure P satisfying $P(A) = P(\{xa; x \in A\})$ for each a in G.

PROPOSITION 3.3. *Whenever the probability distribution of r at Step 2(i) is right-invariant, Scheme II (in B_n, or, more generally, in any group G where the* CONJUGACY SEARCH PROBLEM *is difficult) is a zero-knowledge interactive proof of knowledge of s.*

PROOF (SKETCH). – *Completeness:* In Step $2(iii)$, we have $x = rbr^{-1}$, *i.e.*, $x = yby^{-1}$. In Step $2(iii')$, using $y = rs^{-1}$ and $b' = sbs^{-1}$, we find

$$x = rbr^{-1} = (rs^{-1})(sbs^{-1})(rs^{-1})^{-1} = yb'y^{-1}.$$

Hence Bob accepts a correct answer at each repetition, so he accepts Alice's proof of identity with probability 1.

– *Soundness:* We follow the soundness proof in [**12**]. Suppose that an entity A' is authenticated with nonnegligible probability, and consider the truncated execution tree of (A', B). At each vertex, B may ask two questions. A *heavy* vertex is defined as a vertex with two sons. This means that at this vertex, A' has computed y and y' satisfying $yby^{-1} = y'b'y'^{-1}$. This implies $y'^{-1}yby^{-1}y' = b'$, which gives a solution to the CONJUGACY SEARCH PROBLEM for b' and b. Indeed, the braid $y'^{-1}y$ enables the impersonation of Alice in any execution of the scheme, as it behaves like s. As $1/2^k$ is assumed to be negligible, the rest of the argument is as in [**12**]: as the proof of A' is accepted with nonnegligible probability, some level of the tree has at least half of its vertices having two sons, *i.e.*, being heavy, thus yielding a polynomial algorithm that finds a heavy vertex. Hence, from the knowledge of A' one can extract in polynomial time a braid behaving like the private key of Alice, so Scheme II is sound.

– *Zero-knowledge:* We consider the following probabilistic Turing machine M:

- Step 1: M randomly selects a bit ε and a braid y;
- Step 2: For $\varepsilon = 0$, M computes $x = yby^{-1}$; for $\varepsilon = 1$, it computes $x = yb'y^{-1}$;
- Step 3: M initiates a protocol with B, sends x to B; B replies with the bit ε';
- Step 4: For $\varepsilon = \varepsilon'$, M outputs the triple (x, ε, y), otherwise it resets to Step 1.

Since the probability distribution of r in the authentication protocol is assumed to be right-invariant, we obtain the same probability distribution for the y's generated by M as for Alice's ones. Moreover, since in case $\varepsilon = 1$ we have $x = ysbs^{-1}y^{-1}$, *i.e.*, $x = (ys)b(ys)^{-1}$, using the same assumption, we obtain the same probability distribution for the x's arising for $\varepsilon = 0$ and those arising for $\varepsilon = 1$. As a consequence, B cannot distinguish the two cases, and the probability to have $\varepsilon = \varepsilon'$ is equal to $1/2$. This implies that the computation cost of the machine M will be in average $2k$ operations, and, therefore, that M is a polynomial time machine provided that k itself is polynomial in the size of the initial data. □

3.3. Scheme III. Our last scheme is similar to Scheme II, but it involves the ROOT PROBLEM together with the CONJUGACY SEARCH PROBLEM; more precisely, we use CONJUGACY SEARCH PROBLEM and ROOT PROBLEM for the same braid used as a private key. Here we give the description for exponent $e = 2$; other cases $e = 3$, etc. are similar.

• Phase 1. **Key generation**:

- A(lice) chooses a secret braid s in B_n and computes $b = s^2$, so that the CONJUGACY SEARCH PROBLEM for s and b, and the ROOT PROBLEM for b are difficult; public key is b, private key is s.

• Phase 2. **Authentication phase**: Repeat the following exchanges k times:

- (i) A chooses a random braid r, and sends $x = rbr^{-1}$ to B(ob);
- (ii) B sends a random bit ϵ to A;
- (iii) For $\epsilon = 0$, A sends $y = r$ to B; then B checks $x = yby^{-1}$;
- (iii') For $\epsilon = 1$, A sends $y = rsr^{-1}$ to B; then B checks $x = y^2$.

PROPOSITION 3.4. *Scheme III is a zero–knowledge interactive proof of knowledge of s, provided an element of the conjugacy class of the secret key is published and the probability distribution of r at Step 2(i) is right-invariant.*

PROOF (SKETCH). The completeness and soundness parts are mainly the same as for Scheme II. As for ZK-ness, we have to show that it is possible to simulate Bob's view of the communication with Alice. We assume that the randomization process is uniform (*i.e.*, right-invariant) in B_n. The accepted inputs are of the type $(x, 0, y)$ and $(x, 1, y)$. To generate the instances $(x, 0, y)$, we select a braid specifier y at random and take $x = yby^{-1}$. We obtain the same probability distribution as Alice for this type.

Let us turn to instances of the form $(x, 1, y)$. As conjugates of s will be disclosed in any case, we can assume, (almost) without loss of generality, that a (sufficiently complicated) braid $s' = qsq^{-1}$ of the conjugacy class of s is published at Phase 1. Then, in order to generate the instances $(x, 1, y)$, we choose a specifier r at random, and take $y = rs'r^{-1}$ and $x = y^2$. Then we have $y = (rq)s(rq)^{-1}$ and $x = (rq)s^2(rq)^{-1} = (rq)b(rq)^{-1}$, so, as $r \mapsto rq^{-1}$ is a bijection of B_n and the probability distribution is assumed to be right-invariant, we obtain the same probability distribution as for Alice's messages of this type. □

4. Implementations using normal forms

In practical implementations, we have to replace the infinite braid group B_n by some finite subset, and it is unclear how a right-measure could be defined on such a subset, which cannot be itself invariant under translation. So, in such implementations, Schemes II and III do not remain zero-knowledge, as the distribution of instances generated by a probabilistic polynomial-time Turing machine and that of instances of the conversation between the real Alice and Bob become distinguishable. However, we shall see here how to adapt the schemes so as to retrieve practical indistinguishability, and to ensure that the schemes are secure.

4.1. Specifying a braid. In order to discuss the implementation of our schemes, we first need to know how braids are represented. Several solutions exist. In this section, we consider the most common one, which consists in choosing a normal form, *i.e.*, choosing one distinguished representative for every braid; however, depending on the braid operations we consider, we can also work with arbitrary representatives, as we shall see in Section 5 below.

• **Greedy normal form.** When the braid group B_n is introduced as in Section 2, a braid is an equivalence class of words with respect to the equivalence relation $\equiv$ generated by the relations listed in (2.1). Every braid can be specified by different braid words, but we can avoid ambiguities by resorting to a normal form, *i.e.*, by defining distinguished words, called *normal*, so that each braid is represented by a unique normal word, *i.e.*, every braid word is equivalent to exactly one normal word. Various normal forms have been proposed in the past decades. In terms of applications, the most interesting one seems to be the so-called *greedy normal form* of [**9, 1, 11, 10**]. The latter is associated with an automatic structure,

which in particular implies that the normal form of a length ℓ braid word can be computed in $\mathcal{O}(\ell^2)$ steps—see Chap. 9 of [**11**]. For instance, the greedy normal form is the way braids are specified in [**18**].

• **Permutations.** There exists a natural surjective mapping of B_n to the set S_n of all permutations of n objects. This mapping is non-injective, but there exists a canonical way to choose for every permutation π a braid $e(\pi)$ that projects onto π. Now, let Δ_n be Garside's fundamental braid defined by $\Delta_n = (\sigma_1)(\sigma_2\sigma_1)\dots(\sigma_{n-1}\sigma_{n-2}\dots\sigma_2\sigma_1)$. Then, the normal n-strand braid words considered above happen to have the form $\Delta_n^k e(\pi_1)\dots e(\pi_\ell)$, where k is an integer, and $(\pi_1,\dots,\pi_\ell)$ is a sequence a permutations in S_n. Hence we can encode a normal n-strand braid word by the associated sequence $(k,\pi_1,\dots,\pi_\ell)$, with k an integer and $\pi_1,\dots,\pi_\ell$ permutations in S_n. Conversely, we can associate to every sequence $(k,\pi_1,\dots,\pi_\ell)$ as above the braid word $\Delta_n^k e(\pi_1)\dots e(\pi_\ell)$, thus defining a unique braid, and, if this word is normal, we naturally say that the sequence $(k,\pi_1,\dots,\pi_\ell)$ is normal. The point is that there exists an efficient algorithm [**11**] which, starting with an arbitrary sequence $(k,\pi_1,\dots,\pi_\ell)$, computes the unique normal sequence $(k',\pi'_1,\dots,\pi'_{\ell'})$ representing the braid $\Delta_n^k e(\pi_1)\dots e(\pi_\ell)$, and, therefore, sequences of permutations also provide a good solution for working with braids. In the sequel, if the braid b is represented by the normal sequence $(k,\pi_1,\dots,\pi_\ell)$—or, equivalently, by the normal word $\Delta_n^k e(\pi_1)\dots e(\pi_\ell)$—the number k will be called the *infimum length* of b, and denoted by $\inf(b)$, and the number $k+\ell$ will be called the *supremum length* of b, denoted by $\sup(b)$.

Let us observe that, even if representing braids using sequences of permutations is essentially equivalent to representing them using normal words, going from one representation to the other may have a noticeable algorithmical cost. Even measuring the length of the data can give different values: for instance, the braid $b = \sigma_1^\ell$ in B_n is represented by the length ℓ normal word $\sigma_1\dots\sigma_1$, and by the product of ℓ permutations $(2\ 1\ 3\dots n)$, while $b' = \Delta_n$ is represented by the above word of length $n(n-1)/2$, and by the unique permutation $(n\ n{-}1\ \dots 2\ 1)$: the specifiers of b and b' have respectively length ℓ and $n(n-1)/2$ in one case, while they have lengths $n\ell$ and n in the other.

4.2. A framework for secure implementations of Scheme II and III. Let us come back to the braid schemes of Section 3. The implementation of Scheme I does not alter its security properties, so we will focus on the other two schemes, and more specifically on Scheme II. The general framework is as follows. We use some finite alphabet $\mathcal{A}$ (consisting of the letters $\sigma_i^{\pm1}$, or of permutations plus one integer, or possibly of other symbols), in which we draw words with some length upper bound ℓ. Thus, using $\mathcal{A}^{\leqslant\ell}$ for the set of all words over $\mathcal{A}$ of length at most ℓ, the finite set of braids we consider is the set $\mathcal{B}_n^{\leqslant\ell}$ of all braids in B_n that can be specified (in at least one way) by a word in $\mathcal{A}^{\leqslant\ell}$, *i.e.*,

$$\mathcal{B}_n^{\leqslant\ell} = \{[w]\,;\, w\in\mathcal{A}^{\leqslant\ell}\},$$

where $[w]$ denotes the braid represented by w. We use $\mathcal{B}_n^\ell$ for $\{[w]\,;\, w\in\mathcal{A}^\ell\}$.

The probability distribution we consider is supposed to be uniform on $\mathcal{A}^{\leqslant\ell}$, so, as was said above, it does not achieve the right invariance condition required for Proposition 3.3, and there is a problem for designing a probabilistic Turing machine simulating the exchanges in Scheme II in the case $\varepsilon = 1$. *A priori*, two problems could arise when length is fixed, namely not being able to recreate all instances of the communication between A and B, or being able to create them but with a distinguishable probability distribution only. Let us observe that the first problem does not occur: for any prover A, there exists a probabilistic polynomial-time Turing machine M, which is able to recreate B's view of the communication in Step 2(iii). Indeed, we can consider the Turing machine which generates instances of the

form $(rbr^{-1}, 0, r)$ and $(rr'^{-1}b'r'r^{-1}, 1, rr'^{-1})$, where r is chosen the way it is by A, and r' is a random braid chosen with the same length the private key s lies. As a new r' is chosen for each instance, this ensures that every instance of the communication between A and B can be recreated.

4.3. Scheme II'. So, the problem lies in the probability distributions of the instances of the form $(x, 1, y)$: indeed, the distribution of the braids rs^{-1} with r in $\mathcal{B}_n^{\leqslant \ell}$ can be in general distinguished from that of random braids rr'^{-1} considered above. Moreover, the effective distribution of braids $x = rbr^{-1}$ can also be distinguished from that of braids $rr'^{-1}b'r'r^{-1}$. To get rid of the problem, we consider a variant of Scheme II, called Scheme II'. We show that Scheme II' is, in fact, as secure as Scheme II itself, and that it suits proper zero knowledge properties.

Let us assume here that braids are specified using normal sequences of permutations. So $\mathcal{A}$ denotes the set of all permutations of n objects, and Alice is supposed to choose random words uniformly in $\mathcal{A}^\ell$. We define Scheme II' by introducing the following changes in Scheme II:

- the private key s satisfies $\inf(s) = 0$, *i.e.*, there is no Δ_n in the normal word expansion of s, and $\sup(s) = \ell$; these values are public, and so is the exponent sum $e(s)$.
- the public key b' is chosen in $\mathcal{B}_n^\ell$, and b is deduced using $b = s^{-1}b's$.

The differences with Scheme II are that, in Scheme II, the private key s is chosen in $\mathcal{B}_n^{\leqslant \ell}$, so that we have $\sup(s) \leqslant \ell$ only, and the public key b is chosen first, thus being simpler (in some sense) than b', which is deduced by conjugacy.

PROPOSITION 4.1. *Scheme II' and Scheme II have the same security.*

The proof is in Appendix A.

By construction, Scheme II' has the same ideal zero knowledge properties as Scheme II, and, in particular, Proposition 3.3 still holds. We now give a statistical argument based on a length analysis supporting the claim that, for Scheme II' the distinguishability introduced by the restriction to a finite set of braids provides no useful information.

PROPOSITION 4.2. *For every prover A, there exists a probabilistic polynomial-time Turing machine, which recreates for each B its view of the communication between A and B with a probability distribution that is indistinguishable through statistical length analysis of the data.*

The proof is in Appendix A.

At last, the fact that the set $\mathcal{B}_n^{\leqslant \ell}$ is huge (namely, we have $\#\mathcal{B}_n^{\leqslant \ell} \geqslant (\lfloor \frac{n-1}{2} \rfloor !)^\ell$, see [**18**]), ensures us that the probability that Alice chooses two braids which are close is negligible. So, the accumulation of instances of the effective communication between A and B does not enable to deduce useful information about the private key.

Besides the previous theoretical result, the following practical observations show that Scheme II remains secure even in the finite case, despite the fact that they are not strictly zero-knowledge (in the sense of [**12**]). As was already observed, the instances $(x, 0, y)$ can be perfectly simulated by a probabilistic polynomial time Turing machine, so they do not give any clue on the private key. Now, in the case of Scheme II, the instances $(x, 1, y)$ have the form $(rbr^{-1}, 1, rs^{-1})$. The only way to gain information about s from rs^{-1} is to first gain information about r. Now, as the CONJUGACY SEARCH PROBLEM for b is assumed to be difficult, it is impossible to obtain any information about r (and, of course, about s) from rbr^{-1}.

We thus have discussed the security for an implementation of Scheme II when braids are specified using sequences of permutations, or, equivalently, of normal words. Let us

mention similar security results can be obtained when braids are specified using arbitrary braid words. Finally, as could be expected, analogous results hold when Scheme II is replaced with Scheme III.

4.4. Choice of the parameters. When braids are represented using a normal form, the choice of the public and private keys ensuring security of the schemes depends only on the bounds making the Conjugacy Search Problem the Diffie-Helmann-like Conjugacy Problem and the Root Problem difficult. From the considerations in [**18**], it arises that, when braids are specified using sequences of permutations, parameters ensuring sufficient security and efficiency are $n = 30$ ($n = 60$ in the case of Scheme I) and products of 15 random permutations. For such values, the operators are specially efficient. In particular, computing the product, which determines the complexity of the process, is very fast [**11**], and computing the inverse can be done in linear time, so it does not increase the complexity. This representation gives a very efficient framework.

When braids are represented using greedy normal braid words, for the same values of n, choosing braid words of length at least $1,000$ guarantees at least the same level of security as the values given for products of permutations.

5. Implementations using braid reduction

We describe now implementations of a completely new type, namely implementations based on a braid algorithm called *handle reduction*. Such implementations are more flexible, but require specifications to ensure a good level of security.

5.1. Non unique specifiers. The implementations of Section 4 involve unique expressions of the braids: in each case, a braid is encoded into a unique object, and, eventually, it is specified by a *unique* word over some alphabet, whose letters may be σ_i's (in the case of the greedy normal form), or permutations (as in 4.1), or still other symbols (as in the case of curve diagrams in [**14**]).

At least in some cases, this uniqueness is not needed. The point is as follows: all algorithms using braids involve the group structure of B_n, *i.e.*, they appeal to the product and inverse operations. When we use normal words, or any alternative representation, computing the product of two braids is not trivial, but checking the equality of two braids is. Indeed, if two braids b, b' are specified by normal words w, w' respectively, then specifying the product bb' requires computing the normal form of the word ww', which has a noticeable algorithmical cost, while checking the possible equality $b = b'$ is just checking that the words w and w' are equal, since the normal form is unique by hypothesis.

Assume now that we use arbitrary braid words. Then, assuming that w is one of the specifiers of b and w' is one of the specifiers of b', we directly obtain a specifier of the braid bb', namely the concatenated word ww': here the product operation (and, similarly, the inverse operation) has a negligible cost. Now, deciding the equality $b = b'$ requires deciding the equivalence $w \equiv w'$, *i.e.*, we have to solve the so-called *word problem* of the considered presentation of B_n. Provided efficient solutions to the latter problem are known—and this is the case—it can be algorithmically more efficient to appeal to the latter approach.

Using unique or non necessarily unique braid specifiers may depend on the procedures we have in mind. If we wish to encipher data in braids, then using non-unique specifications seems ill-adapted, as this could result in an ambiguity about the enciphered data. But, if we use braids for authenticating entities—as in the schemes described here—then using non-unique representatives may be appropriate.

5.2. Handle reduction of braids. The previous principles can be implemented by using an efficient solution for the word problem of B_n that involves no normal form, namely the handle reduction method of [**7**]. Handle reduction is an algorithmic procedure that takes a braid word w as input and returns an equivalent braid word $\mathrm{red}(w)$. From an algorithmic point of view, reducing a braid word is (much) simpler than computing its greedy normal form, but, on the other hand, reduction does not yield a normal form, as $w \equiv w'$ does not imply $\mathrm{red}(w) = \mathrm{red}(w')$ in general. We refer to Appendix B for a description of the method. For our current purpose, the point is the following direct consequence of Proposition B.1:

PROPOSITION 5.1. *Let w, w' be braid words. Then w and w' are equivalent if and only if the braid word $w^{-1}w'$ reduces to the empty word.*

Let $\mathcal{A}$ denote here the alphabet $\{\sigma_1^{\pm 1}, \dots, \sigma_{n-1}^{\pm 1}\}$, and let $\mathcal{B}_n$ denote the set of all n-strand braid words, *i.e.*, the set of all words over $\mathcal{A}$. As previously, for w in $\mathcal{B}_n$, we denote by $[w]$ the braid represented by w. Let us consider Scheme II. The data used by Alice and Bob are now braid words, and equality is replaced by equivalence everywhere. So, Phase 1 becomes

• Phase 1. **Key generation**:
- (i) Choose a public braid word b;
- (ii) A(lice) chooses a secret braid word s, her private key; she publishes $b' \equiv sbs^{-1}$; the pair of braid words (b, b') is the public key.

Here something is missing, namely the way the public key b' is chosen. Should we use unique braid specifiers, then, by definition, there would be one possible choice for b' only. Now we use arbitrary braid words, so we must explain how to choose b' among the various braid words equivalent to sbs^{-1}. This means that we have to choose some function S of $\mathcal{B}_n$ to itself mapping every braid word to an equivalent braid word. This function S will be called the *scrambling* function here, as the security of the exchanges will rely on the impossibility of recovering the word w from the word $S(w)$. So, our implementation of Scheme II takes the final form

• Phase 1. **Key generation**:
- (i) Choose a public braid word b;
- (ii) A(lice) chooses a secret braid word s, and publishes $b' = S(sbs^{-1})$; public key is the pair of words (b, b'); private key is the word s.

• Phase 2. **Authentication phase**: Repeat the following exchanges k times:
- (i) A chooses a random braid word r, and sends $x = S(rbr^{-1})$ to B(ob);
- (ii) B sends a random bit ϵ to A;
- (iii) For $\epsilon = 0$, A sends $y = r$ to B, and B checks that $x^{-1}yby^{-1}$ reduces to ε;
- (iii') For $\epsilon = 1$, A sends $y = S(rs^{-1})$, and B checks that $x^{-1}yb'y^{-1}$ reduces to ε.

5.3. Choice of a scrambling function. Defining the scrambling function S to be, say, the identity function on $\mathcal{B}_n$ is impossible: if the words b and sbs^{-1} are public, then the word s can be read directly as the unique prefix of sbs^{-1} of the convenient length. We have seen that the security of the schemes described in Section 3 relies on requirements about the involved braid problems. When adapted to the current context, these requirements become:

(5.1) Recovering the braid $[s]$ from the words b and $S(sbs^{-1})$ must be infeasible.

(5.2) Recovering the braid $[s]$ from the word $S(s^2)$ must be infeasible.

A first, natural solution is to define the scrambling function S to be a unique, distinguished braid representative, for instance

- Define $S(w)$ to be the normal form of w.

For such a choice, (5.1) and (5.2) are merely equivalent to their braid counterparts, and the security of our schemes is guaranteed for typical values of the parameters similar to those considered in [**18**] and mentioned above.

We argued that the advantage of using arbitrary braid words and handle reduction is to avoid computing normal forms, so using normal form for the scrambling process may appear paradoxical. Let us observe that such a choice could be relevant, especially in the case of the non-symmetric Schemes II and III. Here Alice (the prover) would have to compute normal forms—which we said has a non-negligible computation cost—but Bob (the verifier) only has to check braid equivalences, what he can easily do using braid word reduction, which, in particular, requires very little memory.

A second, different, solution is to construct a scrambling function using handle reduction. What makes Requirement (5.1) more difficult than its braid counterpart is that the word $S(sbs^{-1})$ may contain information about the prefixes of sbs^{-1}, so in particular about s or its equivalence class $[s]$. Now assume that we can define S so that the following condition holds:

(5.3) No proper prefix of $S(w)$ is equivalent to a (proper) prefix of w.

Then the only piece of information we can extract from, say, the words b and $S(sbs^{-1})$ is the braid $[b]$ and the braid $[sbs^{-1}]$, and we are back to Conjugacy Search Problem for finding $[s]$. The argument is similar for $S(s^2)$, so we can consider that, if a function S satisfying Condition (5.3) is available, then the requirements of Section 3 are satisfied under the same theoretical assumptions as above about Conjugacy Search Problem and Root Problem.

We claim that this can be (nearly) achieved using handle reduction for scrambling, namely

- Define $S(w)$ to be $\mathrm{red}(w)$, *i.e.*, the result of reducing w.

This definition makes sense, as the braid word $\mathrm{red}(w)$ is equivalent to w in every case. Condition (5.3) is not fulfilled in general: if the initial word w is reduced, *i.e.*, contains no handle, then $\mathrm{red}(w)$ is equal to w, so, in particular, w and $\mathrm{red}(w)$ have many prefixes in common. However, we show in Appendix C how to choose the parameters so as to avoid the problem, namely by defining the public key b to be a word of the form $b_1^{-1}b_2$, where b_1 and b_2 are positive braid words (no letter σ_i^{-1}) of equal length and, moreover, their classes are large with respect to some linear ordering of B_n (see Appendix C).

We refer to Appendix E for still another possible choice of the scrambling function involving another braid algorithm called word reversing.

5.4. Zero knowledge property. In Section 4, we adapted Scheme II and defined Scheme II' so as to guarantee that a probabilistic Turing machine can generate instances that are indistinguishable from those used in the communication between Alice and Bob. Scheme II' is still relevant when non-unique specifiers are used, and the result about the indistinguishability of the braids is still valid. However, in the current case, we have to distinguish between the words that are exchanged, and the braids they represent, and check indistinguishability for words, and not only for braids. Now, as the scrambling function is public, it suffices that we define our Turing machine so that it applies this function whenever Alice does. Then it just remains to check that the exchanged reduced (or scrambled) words are indistinguishable in terms of length and of common subwords. The first point follows from the fact that the braid word s' used in the proof of Prop. 4.2 has the same length as

the private key s; the second point is true provided the private key is chosen as explained in Appendix C. Thus Scheme II' satisfies the same zero knowledge properties in the framework of non-unique specifiers as in a framework of normal forms.

5.5. Normal form vs. reduction. The implementations we have proposed consist in really different approaches. We wish to put some emphasis on the use of braid word reduction here, as implementations of the braid group background already appear in the literature, and they are immediately suitable for our schemes. Anyway, comparing these different implementations directly is not easy, as the associated notions of a random braid are not the same. When we think of using braid reduction, it is advisable to restrict to relatively short braid words representing braids with a long normal form (or, equivalently, a long product of permutations). For instance, we can consider those braid words such that, after an occurence of $\sigma_i^{\pm 1}$, the next letter is $\sigma_{i-1}^{\pm 1}$, $\sigma_i^{\pm 1}$, or $\sigma_{i+1}^{\pm 1}$; we can also randomly increase the number of squares in a braid word by replacing letters σ_i with σ_i^2, which dramatically increases the length of the normal form, whereas the overall cost of reduction almost remains the same. Let us observe that attacks like the one considered in [**17**] demand to use braids with long greedy normal forms: using special braids words of the type considered above together with handle reduction is then an efficient solution.

6. Conclusion

In this paper, we have proposed the first authentication schemes specially designed for braid groups. The first of them is a two-pass protocol relying on a specific version of the conjugacy search problem, while the two other ones are iterated three-pass protocols based on the Conjugacy Search Problem and/or Root Problem. We have discussed the security of these schemes and given evidence that none of them leaked any useful information on the secret key, even though traditional zero knowledge models cannot apply to the braid groups, which are infinite. Finally, we have addressed in a detailed manner the issue of implementing these protocols, and more particularly have pointed out the relevance of the so-called handle reduction, which allows to efficiently verify the equivalence of two braid words, without requiring to compute a normal form of any of them, thus being particularly relevant for authentication schemes.

References

[1] S.I. Adjan, *Fragments of the word Delta in a braid group*, Mat. Zam. Acad. Sci. SSSR; 36-1; 1984; 25–34; translated Math. Notes of the Acad. Sci. USSR 36-1 (1984) 505–510.
[2] I. Anshel, M. Anshel & D. Goldfeld, *An algebraic method for public-key cryptography*, Math. Research Letters 6 (1999) 287–291.
[3] I. Anshel, M. Anshel, B. Fisher & D. Goldfeld, *New key agreement schemes in braid group cryptography*, RSA 2001.
[4] E. Artin, *Theory of Braids*, Ann. of Math. 48 (1947) 101–126.
[5] J. Birman, K.H. Ko & S.J. Lee, *A new approach to the word problem in the braid groups*, Advances in Math. 139-2 (1998) 322-353.
[6] J.C. Cha, K.H. Ko, S.J. Lee, J.W. Han, J.H. Cheon, *An efficient implementation of braid groups*, AsiaCrypt 2001.
[7] P. Dehornoy, *A fast method for comparing braids*, Advances in Math. 125 (1997) 200–235.
[8] P. Dehornoy, *Braids and Self-Distributivity*, Progress in Math. vol. 192 Birkhäuser (2000).
[9] P. Deligne, *Les immeubles des groupes de tresses généralisés*, Invent. Math. 17 (1972) 273–302.
[10] E. A. Elrifai & H. R. Morton, *Algorithms for positive braids*, Quart. J. Math. Oxford 45-2 (1994) 479–497.
[11] D. Epstein & *al.*, *Word Processing in Groups*, Jones & Bartlett Publ. (1992).
[12] U. Feige, A. FIat & A. Shamir, *Zero–knowledge proofs of identity*, J.of Cryptology 1 (1988) 77–94.
[13] N. Franco & J. Gonzales-Meneses, *Conjugacy problem for braid groups and Garside groups*, http://xxx.lanl.gov/abs/math.GT/0112310 (2001).

[14] R. Fenn, M.T. Greene, D. Rolfsen, C. Rourke & B. Wiest, *Ordering the braid groups*, Pacific J. of Math. 191 (1999) 49–74.
[15] F. A. Garside, *The braid group and other groups*, Quart. J. Math. Oxford 20-78 (1969) 235–254.
[16] S. Goldwasser, S. Micali & C. Rackoff, *Knowledge complexity of interactive proof systems*, Proc. 17th STOC (1985) 291–304.
[17] J. Hughes, *The left SSS attack on Ko-Lee-Cheon-Han-Kang-Park key agreement scheme in* B_{45}, Rump session Crypto 2000.
[18] K.H. Ko, S.J. Lee, J.H. Cheon, J.W. Han, J.S. Kang, C. Park, *New public-key cryptosystem using braid groups*, Crypto 2000 166–184.
[19] E.K. Lee, S.J. Lee, S.G. Hahn, *Pseudorandomness from braid groups*, Crypto 2001.
[20] M.S. Paterson & A. A. Razborov, *The Set of minimal braids is co-NP-complete*, J. of Algorithms 12 (1991) 393–408.
[21] H. Sibert, *Extraction of roots in Garside groups*, Comm. in Algebra 30 (6) 2002 2915–2927.
[22] V.M. Sidelnikov, M.A. Cherepnev, V.Y. Yashcenko, *Systems of open distribution of keys on the basis of noncommutative semigroups*, Ross. Acad. Nauk Dokl. 332-5 (1993); English translation: Russian Acad. Sci. Dokl. Math. 48-2 (1194) 384–386.
[23] V.B. Styshnev, *The extraction of a root in a braid group (English)*, Math. USSR Izv. 13 (1979) 405–416.

Appendix A: Proofs of Proposition 4.1 and 4.2 of Section 4

Proof of Proposition 4.1. It is obvious that Scheme II is at least as hard as Scheme II', so we are left with the question of proving that, conversely, Scheme II' is not significantly easier than Scheme II.

Firstly, notice that restricting to private keys s with the property that Δ_n does not occur in the normal form of s does not modify the security of the scheme, because, for each integer k, the braids s and $\Delta_n^k s$ are equivalent from the point of view of conjugacy: the braid Δ_n^2 belongs to the center of B_n, so, if k is even, $b' = (\Delta_n^k s)b(\Delta_n^k s)^{-1}$ is merely equivalent to $b' = sbs^{-1}$; if k is odd, $b' = (\Delta_n^k s)b(\Delta_n^k s)^{-1}$ is equivalent to $\phi_n(b') = sbs^{-1}$, where ϕ_n denotes the flip automorphism of B_n that exchanges σ_i and σ_{n-i} for every i, and, in all cases, solving Conjugacy Search Problem for (b, b') and for $(b, \phi_n(b'))$ has the same complexity.

The second requirement, namely that b is chosen after b', does not change the security, as, in any case, we demand the relation $b' = sbs^{-1}$. Here is the point of the argument. When we are given a normal sequence $(k, \pi_1, \dots, \pi_{\ell-1})$, there are at least $\lfloor \frac{n-1}{2} \rfloor$ permutations π such that $(k, \pi_1, \dots, \pi_{\ell-1}, \pi)$ is normal. This implies $\#\mathcal{B}_n^\ell \geqslant \lfloor \frac{n-1}{2} \rfloor \#\mathcal{B}_n^{\ell-1}$ for every ℓ, and, therefore, the number of private keys of a given length ℓ is equivalent to the number of private keys of length $\leqslant \ell$. We can reasonably assume the security of the scheme grows with the number of private keys of a given length. We deduce that solving Scheme II' for private keys of supremum length ℓ is not easier than solving it for all keys of supremum length $\leqslant \ell$, hence not easier than solving Scheme II with the same parameter ℓ. □

(In practical implementations, the parameter ℓ is given relatively small values, typically 20, so another argument showing that Scheme II' is not easier than Scheme II is that we can solve Scheme II for ℓ by systematically solving Scheme II' for $\ell' \leqslant \ell$.)

Proof of Proposition 4.2. Let us consider the Turing machine M used in the proof of Proposition 3.3. We construct a new Turing machine M', modifying M as follows: before starting communication, M' draws a random braid s', with $\sup(s)' = \sup(s)$ and $e(s') = e(s)$ (those numbers are public). Note that such a drawing can be made in polynomial time. Then, M' randomly selects a bit ε and a braid r (step 1). For $\varepsilon = 0$, M' computes the triple $(rbr^{-1}, 0, r)$; for $\varepsilon = 1$, M' computes $(rs'^{-1}b's'r^{-1}, 1, rs'^{-1})$ (step 2), always choosing r in

the way A does, *i.e.*, it simulates A's answers but using the random key s' with the same length parameters as s. Then M' performs the same steps 3 and 4 as M.

We already observed that there is no problem with the distribution of the triples $(x, 0, y)$. For the triples $(x, 1, y)$, the instances produced by M' are indistinguishable from the original instances $(rbr^{-1}, 1, rs^{-1})$ generated by A, according to the statistical results we obtained by implementing the normal form as described in [**6**]. For n sufficiently large ($n \geqslant 30$), the infimum and supremum lengths behave like additive lengths. Moreover we observed, for every random braid r, the following relation : $\inf(r^{-1}) = -\sup r$, and $\sup(r^{-1}) = -\inf r$. So we have, for an overwhelming proportion of pairs of braids (r, a), $\inf(ra) = \inf r + \inf a$ and $\sup(ra) = \sup r + \sup a$, and similar relations for conjugacy (see Table 1). Then, straight computation shows that the braids effectively transmitted by A, and those generated by the Turing machine, have the same infimum and supremum lengths and exponent sum, so they are indistinguishable by statistical means. Moreover, the same computation shows that the probability distributions for the x's in the triples (x, ε, y) generated by M' in the cases $\varepsilon = 0$ and $\varepsilon = 1$ are indistinguishable.

The results are the same when braids are specified using normal words, because the random braid s' and the private key s have the same exponent sum, and, as there is no Δ_n in their normal expansion, they are represented by positive words of the same length.

Moreover, in any case, relative comparison of several instances of the form $(yb'y^{-1}, 1, y)$ does not yield anything, as the distribution of y is simply translated from the distribution of random braids. □

Table 1 shows, for different values of the braid index n, the average supremum length of the braids rs^{-1} and rbr^{-1}, where r, s and b are braids with infimum length 0. For $n = 30$ and above, we observe that the supremum length behaves just like an additive length —to which corresponds the rows labelled "awaited value"—, as claimed in the proof of Proposition 4.2. Similar results hold regarding the infimum length.

n	$\sup(r) = \sup(s)$	$\sup(b)$	$\sup(rbr^{-1})$/ awaited value	$\sup(rs^{-1})$/ awaited value
15	5	10	14.83 / 15	4.94 / 5
15	30	60	88.65 / 90	29.4 / 30
20	10	20	29.98 / 30	9.99 / 10
20	40	80	119.93 / 120	39.98 / 40
30	10	20	30.00 / 30	10.00 / 10
30	40	80	120.00 / 120	40.00 / 40
50	15	30	45.00 / 45	15.00 / 15
50	50	100	150.00 / 150	50.00 / 50
100	20	40	60.00 / 60	20.00 / 20
100	50	100	150.00 / 150	50.00 / 50
200	30	60	90.00 / 90	30.00 / 30
200	100	200	300.00 / 300	100.00 / 100

TABLE 1. Supremum length of rbr^{-1} and rs^{-1} when r, b, s are random positive (*i.e.*, satisfying $\inf r = \inf s = \inf b = 0$) braids not containing Δ_n of given sup (samples of 10,000 braids).

Appendix B: Handle reduction

Handle reduction is an algorithmic procedure that takes a braid word w as input and returns another equivalent braid word $\mathrm{red}(w)$ with the property that w represents the trivial braid if and only if the word $\mathrm{red}(w)$ is empty. From an algorithmic point of view, reducing a braid word is (much) simpler than computing its greedy normal form. The counterpart to this efficiency is that reduced braid words do not yield a normal form: there exist equivalent braid words w, w' such that the words $\mathrm{red}(w)$ and $\mathrm{red}(w')$ are not equal.

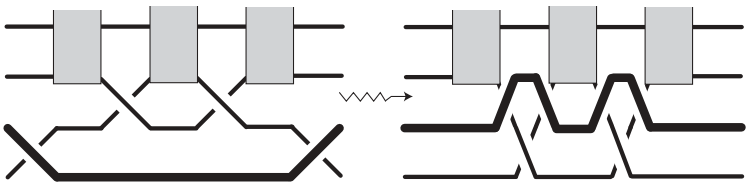
Handle reduction is a direct generalization of the usual free reduction process consisting in iteratively deleting the patterns xx^{-1} or $x^{-1}x$ in a word. Let us say that w freely reduces to w' if w' is obtained from w by iteratively deleting subwords $\sigma_i\sigma_i^{-1}$ or $\sigma_i^{-1}\sigma_i$. Because every braid word $\sigma_i\sigma_i^{-1}$ or $\sigma_i^{-1}\sigma_i$ is equivalent to the empty word, w being freely reducible to w' implies $w \equiv w'$. In particular, if w is freely reducible to the empty word, then $w \equiv \varepsilon$ holds. The converse implication is not true, because the braid groups are not free: for instance, we have $\sigma_1\sigma_3\sigma_1^{-1}\sigma_3^{-1} \equiv \varepsilon$, but the word $\sigma_1\sigma_3\sigma_1^{-1}\sigma_3^{-1}$ cannot be freely reduced to ε.

Definition. A *σ_i-handle* is defined to be a braid word of the form $\sigma_i^e v \sigma_i^{-e}$ where e is ± 1 and v is a braid word that contains no letter σ_k with $k \leqslant i$ and, moreover, either σ_{i+1} or σ_{i+1}^{-1} does not occur in v.

We see that a braid word of the type $\sigma_i\sigma_i^{-1}$ or $\sigma_i^{-1}\sigma_i$ is a special case of handle. More generally, a handle corresponds under the standard geometrical interpretation of braids to the following geometrical pattern, which explains the name:

Definition. Assume that u is a σ_i-handle, say $u = \sigma_i^e v \sigma_i^{-e}$. We define $\mathrm{red}(u)$ to be the braid word obtained from v by replacing each letter $\sigma_{i+1}^{\pm 1}$ with $\sigma_{i+1}^{-e}\sigma_i^{\pm 1}\sigma_{i+1}^{e}$, and keeping the other letters unchanged. If w, w' are braid words, we say that w' is obtained from w by handle reduction if we can transform w into w' by iteratively replacing some handles u with the corresponding words $\mathrm{red}(u)$.

It is clear that handle reduction generalizes free reduction, and that reducing a braid word yields an equivalent braid word, as shows the figure below:



On the other hand, it is not clear that braid word reduction has to terminate, as the length of the words may increase during reduction. The following result is proved in [**7**]:

Proposition B.1. *Let w be a braid word of length ℓ. Then w is equivalent to the empty word ε if and only if w reduces to ε if and only if every sequence of reductions starting from w finishes with ε. Moreover, every sequence of reductions starting from w terminates in exponential time at most.*

Thus, braid word reduction always detects triviality: a braid word is trivial, *i.e.*, equivalent to the empty word, if and only if it reduces to the empty word. Several reduction strategies have been proposed; as one can expect, those based on a divide-and-conquer principle are more efficient in practice. In the sequel, we assume that such a strategy is fixed, and we denote by $\mathrm{red}(w)$ the unique reduced word obtained from w. The complexity upper bound stated in Theorem 6 seems to be far from optimal: we have no example of a worst case complexity higher than ℓ^2, and the average case complexity seems to grow approximately as $\ell^{1.5}$, at least for $\ell \leqslant 5{,}000$: see Table 2 for some statistics on the number of reduction steps and the average time needed to reduce a random braid in terms of the braid index (*i.e.*, the size of the alphabet) and the number of crossings (*i.e.*, the length of the words).

	$n = 8$	$n = 16$	$n = 32$	$n = 64$
$\ell = 64$	0.061 (7)	0.026 (2.1)	0.023 (0.7)	0.018 (0.3)
$\ell = 256$	0.85 (88)	0.35 (22)	0.18 (6.4)	0.07 (2.1)
$\ell = 1,024$	28 (1,974)	7 (385)	2.5 (81)	1.3 (21)
$\ell = 4,096$	1,333 (35,966)	621 (18,617)	88 (2,188)	24 (358)
$\ell = 16,384$	44,784 (381,938)	69,773 (627,177)	11,401 (116,900)	1,404 (15,520)

TABLE 2. Statistics for handle reduction: average CPU time in millisec. and (bracketed) average number of reduction steps in terms of the braid index n and the length ℓ of the words

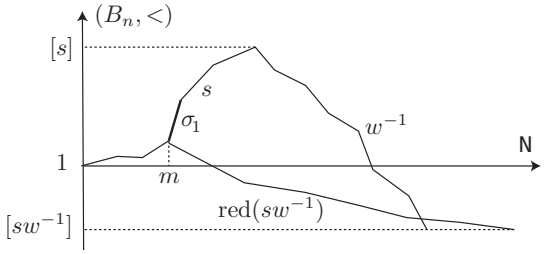
Appendix C: Avoiding common prefixes in handle reduction

The braid groups B_n are equipped with a linear ordering $<$ compatible with multiplication on the left and admitting the following characterization [**8**]: $[w] > 1$ holds if and only if the word $\mathrm{red}(w)$ contains σ_1 (in which case it cannot contain σ_1^{-1}), or no $\sigma_1^{\pm 1}$ and σ_2 (in which case it cannot contain σ_2^{-1}), etc. We then have the following theoretical result:

PROPOSITION C.1. *Assume that s is a random positive braid word, containing at least one σ_1, and $[w] > [s]$ holds. Then the average length of the maximum common prefix of $\mathrm{red}(sw^{-1})$ and s has upper bound $n - 2$, and no prefix of $\mathrm{red}(sw^{-1})$ is equivalent but not equal to some prefix of s.*

PROOF. Let m denote the length of the longest prefix of s not containing σ_1. As we assume $[s] < [w]$, the word $\mathrm{red}(sw^{-1})$ does not contain σ_1, so the longest common prefix of s and $\mathrm{red}(sw^{-1})$ has length at most m. Computing the average value of m gives the first result. Define, for each braid word u of length ℓ, the characteristic function $\chi_u : \{0, 1, \dots, \ell\} \to B_n$, which associates to k the class of the length k prefix of u in B_n. Then, for $k > m$, we have $\chi_s(k) > \chi_{\mathrm{red}(sw^{-1})}(k')$ for every k', since the braid $\chi_s(k)$ has an expression where σ_1 occurs, while $\chi_{\mathrm{red}(sw^{-1})}(k')$ does not, for, otherwise, $\mathrm{red}(sw^{-1})$ should also contain σ_1, contradicting the hypothesis $[w] > [s]$. This proves that no prefix of the word $\mathrm{red}(sw^{-1})$, excepted possibly one of length m at most, may be equivalent to a prefix of s. □

Proposition C.1 enables us to fulfill enough of Condition (5.3) for our purposes. What we need is to be able to choose the braid words s and b so that s cannot be retrieved from b and $\mathrm{red}(sbs^{-1})$. We can do this as follows: we define b as a word of the form $b_1^{-1}b_2$, where b_1 and b_2 are positive braid words (no letter σ_i^{-1}) of equal length and, moreover, their classes are



large in $(B_n, <)$, and s is a random positive braid word. We assume that handle reduction is implemented according to a divide-and-conquer strategy. Then consider the reduction of the word sbs^{-1}, *i.e.*, of $sb_1^{-1}b_2s^{-1}$. By construction, it consists in separately reducing sb_1^{-1} and b_2s^{-1}, and merging the results. Provided $[s] < [b_1]$ and $[s] < [b_2]$ hold, which follows from our assumption that $[b_1]$ and $[b_2]$ are large in $(B_n, <)$, Proposition C.1 implies that no prefix of s is neither equivalent to a prefix of $\mathrm{red}(sb_1^{-1})$ (excepting possibly the m first prefixes), nor to a prefix of $\mathrm{red}(sb_2^{-1})$, *i.e.*, equivalently, no suffix of s^{-1} is equivalent to a suffix of $\mathrm{red}(b_2s^{-1})$. Finally, an ordering argument similar to the one in the proof of Proposition C.1 shows that reducing the word $\mathrm{red}(sb_1^{-1})\mathrm{red}(b_2s^{-1})$ cannot re-introduce common prefixes with s, and our construction meets the needed requirements.

Remark. The previous construction guarantees that one cannot retrieve the secret key s, or even the braid associated with s or a prefix of s, from $S(sbs^{-1})$. However, it could still happen that some subword of s appears in $S(sbs^{-1})$. It is doubtful that an attack could be based on such coincidences, as there seems to be no way for combining such possible common subwords into useful information about s itself. However, we can get rid of such subwords, and, more generally, require that the words b and $S(b)$ have in common no subword of a prescribed (small) length, using handle reduction. For instance, we can introduce an additional scrambling step by replacing b in $\mathcal{B}_n$ with

$$S_0(b) = \mathrm{red}(\sigma_{1,n}^{-1}\mathrm{red}(\sigma_{1,n}b\sigma_{1,n}^{-1})\sigma_{1,n}),$$

where $\sigma_{1,n}$ is $\sigma_1\sigma_2\cdots\sigma_{n-1}$ (in practice, it is advisable to cut b into fragments of medium length, say 100, to apply S_0 to these fragments, and concatenate the results). The word $S_0(b)$ is equivalent to $\sigma_{1,n}^{-1}\sigma_{1,n}b\sigma_{1,n}^{-1}\sigma_{1,n}$, *i.e.*, to b, and statistics show that the average value of the length of the maximal common subword between b and $S_0(b)$ is generically very small (typically 4), as it is for random words of the same length. Hence, the words $S_0(sbs^{-1})$ do not resemble sbs^{-1} more than random words, and we can suspect S_0 does not preserve any information, apart from braid word equivalence.

Appendix D: Choosing a one-way function in Scheme I

There are several ways of conceiving a one-way hash function suitable for Scheme I, depending on the choice of the implementation.

Hashing from the braid group to natural numbers. In order to construct a hash function H from braids to sequences of 0's and 1's, we have to require that equivalent specifiers are mapped to the same value.

When using permutations as braid specifiers, the most practical way of hashing consists in computing the normal form, and then apply a collision-free hash function from the set of normal forms to the set of natural numbers.

When it comes to other specifiers, computing a normal form is not as fast as with permutations. Then, we can replace the normal form with some braid invariant that is faithful enough. For example, from a braid word, we compute the Burau matrix (which is a matrix with polynomial coefficients) representing a braid, and then we use a collision-free hash function over the matrices with polynomial coefficients. As long as the unfaithfulness of the invariant remains negligible (and it should strongly do in the case of the Burau representation), the completeness and soundness of Scheme I are preserved, and computing H should be faster than computing a normal form.

Hashing from the braid group to the braid group. Another possibility is to consider a hash function H from B_n to B_n itself. Indeed, let us set $H(z) = zbz^{-1}$, where b is the public key of Scheme I. Then, Bob's verification step of Scheme I, namely $H(sxs^{-1}) = H(rb'r^{-1})$, becomes $(sxs^{-1})b(sxs^{-1})^{-1} = (rb'r^{-1})b(rb'r^{-1})^{-1}$.

The completeness and soundness of the scheme still hold, as they rely on the Conjugacy Search Problem. Let us now consider the question of whether it is possible to generate x in order to be able to recover s from $(x, b, sbs^{-1}, (sxs^{-1})b(sxs^{-1})^{-1})$. The only information gained on s that depends on x is the braid $(sxs^{-1})b(sxs^{-1})^{-1}$, which can be called a double conjugate of b relative to x. So this attack amounts to solving the following

Double Conjugacy Search Problem: *Assuming that the braid y is a double conjugate of b with respect to x, find a witness s, i.e., find s verifying $y = (sxs^{-1})b(sxs^{-1})^{-1}$.*

To our knowledge, there is no method to solve this problem other than solving the Conjugacy Search Problem for b and then for x, so, in the state-of-the-art, it is as difficult as the Conjugacy Search Problem for b and x.

Appendix E: Scrambling using random word reversing

We have described in Section 5.2 various strategies for obfuscating braid words by using normal forms or handle reduction. It turns out that handle reduction is not an atomic process in that each elementary step of handle reduction can be decomposed into several steps of a more basic transformation called word reversing [**7**]. A possibility for defining scrambling functions is to appeal to word reversing directly.

Definition. Assume that w is a braid word in $\mathcal{B}_n$. We say that w is *reversible* to w' if w' can be obtained from w by iteratively applying any of the following transformations:

- (i) Replacing some subword $\sigma_i\sigma_j$ with $\sigma_j\sigma_i$ (case $|i-j| \geqslant 2$), or some subword $\sigma_i\sigma_j\sigma_i$ with $\sigma_j\sigma_i\sigma_j$ (case $|i-j| = 1$);
- (ii) Replacing some subword $\sigma_i^{-1}\sigma_j^{-1}$ with $\sigma_j^{-1}\sigma_i^{-1}$ (case $|i-j| \geqslant 2$), or some subword $\sigma_i^{-1}\sigma_j^{-1}\sigma_i^{-1}$ with $\sigma_j^{-1}\sigma_i^{-1}\sigma_j^{-1}$ (case $|i-j| = 1$);
- (iii) Replacing some subword $\sigma_i^{-1}\sigma_j$ with $\sigma_j\sigma_i^{-1}$ (case $|i-j| \geqslant 2$), or some subword $\sigma_i^{-1}\sigma_j$ with $\sigma_j\sigma_i\sigma_j^{-1}\sigma_i^{-1}$ (case $|i-j| = 1$);
- (iv) Replacing some subword $\sigma_i\sigma_j^{-1}$ with $\sigma_j^{-1}\sigma_i$ (case $|i-j| \geqslant 2$), or some subword $\sigma_i\sigma_j^{-1}$ with $\sigma_j^{-1}\sigma_i^{-1}\sigma_j\sigma_i$ (case $|i-j| = 1$);
- (v) Deleting some subword $\sigma_i\sigma_i^{-1}$ or $\sigma_i^{-1}\sigma_i$.

Each of the above elementary transformations replaces a braid word with an equivalent braid word, and reduction is a special case of reversing. Observe that we do not allow to introduce any new factor $\sigma_i\sigma_i^{-1}$ or $\sigma_i^{-1}\sigma_i$, which implies that reversing is not a symmetric process: for instance, the empty word reverses to itself only. Handle reduction turns out to

be a special case of word reversing: if w can be transformed into w' using handle reduction, then w can be transformed into w' using word reversing.

We then propose to define an alternative scrambling function as follows:

- Construct $S'(w)$ from w by applying word reversing randomly until the Hamming distance between w and $S'(w)$ reaches some prescribed (large) value.

The advantage of using S' is that it modifies the braid words quickly and easily. The disadvantage is that it is a probabilistic process, and we can expect no secure criteria to ensure that the word $S'(w)$ does not have a long prefix equivalent to some prefix of w.

REMARK. As the braid equivalence relation $\equiv$ is generated by the relations of Presentation (2.1), which correspond to transformations (i) above, together with the relations $\sigma_i\sigma_i^{-1} \equiv \sigma_i^{-1}\sigma_i \equiv \varepsilon$, we could simply think of using the latter relations randomly. This choice would be unwise, for it would lead to introducing many trivial subwords $\sigma_i\sigma_i^{-1}$ and $\sigma_i^{-1}\sigma_i$, and the probability of generating the transformations (ii) for instance would be very low. The interest of reversing lies in that it is a symmetric process and, provided we restrict to freely reduced words, it increases the length of the words by at most a linear factor.

LABORATOIRE DE MATHÉMATIQUES NICOLAS ORESME, UNIVERSITÉ DE CAEN BP 5186, 14032 CAEN, FRANCE
E-mail address: sibert@math.unicaen.fr
URL: //www.math.unicaen.fr/~sibert

LABORATOIRE DE MATHÉMATIQUES NICOLAS ORESME, UNIVERSITÉ DE CAEN BP 5186, 14032 CAEN, FRANCE
E-mail address: dehornoy@math.unicaen.fr
URL: //www.math.unicaen.fr/~dehornoy

FRANCE TELECOM R&D, 42 RUE DES COUTURES, BP 6243, 14066 CAEN, FRANCE
E-mail address: marc.girault@rd.francetelecom.com