Random Subgroups of Braid Groups: An Approach to Cryptanalysis of a Braid Group based Cryptographic Protocol

Alexei Myasnikov[1], Vladimir Shpilrain[2], and Alexander Ushakov[3]

[1] Department of Mathematics, McGill University, Montreal, Quebec H3A 2T5
alexeim@math.mcgill.ca ⋆
[2] Department of Mathematics, The City College of New York, New York, NY 10031
shpilrain@yahoo.com ⋆⋆
[3] Department of Mathematics, Stevens Institute of Technology, Hoboken, NJ 07030
aushakov@mail.ru

Abstract. Motivated by cryptographic applications, we study subgroups of braid groups B_n generated by a small number of random elements of relatively small lengths compared to n. Our experiments show that "most" of these subgroups are equal to the whole B_n, and "almost all" of these subgroups are generated by positive braid words. We discuss the impact of these experimental results on the security of the Anshel-Anshel-Goldfeld key exchange protocol [2] with originally suggested parameters as well as with recently updated ones.

1 Introduction

Braid group cryptography has attracted a lot of attention recently due to several suggested key exchange protocols (see [2], [11]) using braid groups as a platform. We refer to [3], [5] for more information on braid groups.

Here we start out by giving a brief description of the Anshel-Anshel-Goldfeld key exchange protocol [2] (subsequently called the AAG protocol) to explain our motivation.

Let B_n be the group of braids on n strands and $X_n = \{x_1, \ldots, x_{n-1}\}$ the set of standard generators. Thus,

$$B_n = \langle x_1, \ldots, x_{n-1};\ x_i x_{i+1} x_i = x_{i+1} x_i x_{i+1},\ x_i x_j = x_j x_i \text{ for } |i-j| > 1\rangle.$$

Let $N_1, N_2 \in \mathbb{N}$, $1 \leq L_1 \leq L_2$, and $L \in \mathbb{N}$ be preset parameters. The AAG protocol [2] is the following sequence of steps:

(1) Alice randomly generates an N_1-tuple of braid words $\overline{a} = \{a_1, \ldots a_{N_1}\}$, each of length between L_1 and L_2, such that each generator of B_n non-trivially occurs in $\overline{a}$. The tuple $\overline{a}$ is called *Alice's public set.*

⋆ Partially supported by the NSF grant DMS-0405105.
⋆⋆ Partially supported by the NSF grant DMS-0405105.

(2) Bob randomly generates an N_2-tuple of braid words $\overline{b} = \{b_1, \ldots b_{N_2}\}$, each of length between L_1 and L_2, such that each generator of B_n is non-trivially involved in $\overline{b}$. The tuple $\overline{b}$ is called *Bob's public set.*

(3) Alice randomly generates a product $A = a_{s_1}^{\varepsilon_1} \ldots a_{s_L}^{\varepsilon_L}$, where $1 \le s_i \le N_1$ and $\varepsilon_i = \pm 1$ (for each $1 \le i \le L$). The word A is called *Alice's private key.*

(4) Bob randomly generates a product $B = b_{t_1}^{\delta_1} \ldots b_{t_L}^{\delta_L}$, where $1 \le t_i \le N_2$ and $\delta_i = \pm 1$ (for each $1 \le i \le L$). The word B is called *Bob's private key.*

(5) Alice computes $b_i' = D(A^{-1} b_i A)$ $(1 \le i \le N_2)$ and transmits them to Bob. Here $D(w)$ denotes Dehornoy's form of a braid word w (see the beginning of the next Section 2).

(6) Bob computes $a_i' = D(B^{-1} a_i B)$ $(1 \le i \le N_1)$ and transmits them to Alice.

(7) Alice computes $K_A = A^{-1} a_{s_1}'^{\varepsilon_1} \ldots a_{s_L}'^{\varepsilon_L}$. It is straightforward to see that $K_A = A^{-1}B^{-1}AB$ in the group B_n.

(8) Bob computes $K_B = b_{t_L}'^{-\delta_L} \ldots b_{t_1}'^{-\delta_1} B$. Again, it is easy to see that $K_B = A^{-1}B^{-1}AB$ in the group B_n.

Thus, Alice and Bob end up with the same element $K = K_A = K_B = A^{-1}B^{-1}AB$ of the group B_n. This K is now their common secret key.

Note that for an intruder to get the common secret key K, it is sufficient to find any element $C = a_{r_1}^{\tau_1} \ldots a_{r_m}^{\tau_m}$ such that $\overline{b}' = C^{-1}\overline{b}C$ in the group B_n (see e.g. [11], [15]). Finding such an element is an instance of the following problem (call it *subgroup-restricted conjugacy search problem* for future reference):

Let G be a group, A a subgroup of G generated by some $\{a_1, \ldots a_r\}$, and let $\overline{g} = (g_1, \ldots g_k)$, $\overline{h} = (h_1, \ldots h_k)$ be two tuples of elements of G. Find $x \in A$, as a word in $\{a_1, \ldots a_r\}$, such that $\overline{h} = x^{-1}\overline{g}x$, provided that at least one such x exists.

Without the restriction $x \in A$, this would be a well-known (multiple simultaneous) *conjugacy search problem.* While the latter problem for braid groups is not known to have polynomial-time solution, some important recently made inroads [7], [12] suggest that it may be solved quite efficiently by a deterministic algorithm for at least some inputs, e.g. if one of the tuples $\overline{g}$ or $\overline{h}$ consists of positive braid words only. Thus, having the above subgroup $A \le B_n$ significantly different from B_n should be important for the security of the AAG protocol.

In the present paper, we experimentally show that the parameters

$$N = 80,\ N_1 = 20,\ N_2 = 20,\ L_1 = 5,\ L_2 = 8,\ L = 100$$

for the AAG protocol suggested in [1] may not provide sufficient level of security because the relevant subgroup $A \le B_n$ is either the whole B_n or is "very close" to the whole B_n.

More specifically, out of 100 experiments that we performed, a randomly selected tuple $\overline{a} = (a_1, \ldots, a_{N_1})$ with parameters as above (see our Section 6 for details on producing random tuples) generated the whole group B_n in 63 experiments. In the remaining 37 experiments, the subgroups were "close" to the whole group B_n, and in 36 of them, the subgroups were generated by positive braid words. See Section 4 for more details.

Similar results were obtained in [9] using homomorphisms of braid groups onto permutation groups. In this paper we go further and extend these results to recently suggested greater parameter values; this is discussed later in this section. Our approach to cryptanalysis of the AAG protocol (we call it the "subgroup attack") is rather general and can be used in cryptanalysis of commutator key exchange schemes based on other groups.

In the AAG protocol, there are two subgroups $\overline{a} = (a_1, \ldots, a_{N_1})$ and $\overline{b} = (b_1, \ldots, b_{N_2})$ each of which is generated independently of the other. The following procedure can be used to attack the AAG protocol:

(1) Given two tuples $\overline{a}$ and $\overline{b}$, simplify them using the procedure(s) in our Section 3.

(2) Both simplified tuples will consist of positive braid words with probability 98% (99% each), see the list in the beginning of our Section 4. In that case, the corresponding multiple simultaneous conjugacy search problem can be efficiently solved by the method of [12] (using super summit sets).

(3) With probability 98% (99% each), the centralizer of Alice's and Bob's subgroup coincides with the center of B_n. Therefore, any solution of the multiple simultaneous conjugacy search problem obtained by using, say, the method of [12] mentioned above, will differ from the actual Alice's (Bob's) private key by a factor lying in the center of B_n. This will yield the correct common secret key K because K is the commutator $K = A^{-1}B^{-1}AB$, and therefore its value does not change if either A or B or both are multiplied by elements from the center of the ambient group B_n. Thus, one does not have to solve the subgroup-restricted conjugacy search problem in this case.

The above claim that the centralizer of any subgroup (except the last one) on the list in the beginning of Section 4 coincides with the center of B_n, follows from the following fact: any element in the group B_n that commutes with x_i^k for some positive k, also commutes with x_i. This, in turn, follows from the results of [8].

Thus, it appears that with probability at least $98\% \cdot 98\% \approx 96\%$, the AAG protocol (with parameters as in [1]) can be successfully attacked by the procedure outlined above.

We note that by increasing the crucial parameters L_1 and L_2 (and therefore increasing the lengths of the private keys), it is probably possible to downsize the relevant subgroup so that the method of [12] would not work. However, for public sets with longer elements, length-based attacks, as described in [6], [9],

[10], may become a threat, although it seems that the existing experimental base is insufficient to draw any definitive conclusions on using longer keys in the AAG protocol.

Another possible way of improving security of the AAG protocol might be increasing the rank of the ambient braid group. However, we have run similar experiments with $N = 150,\ N_1 = 20,\ N_2 = 20,\ L_1 = 10,\ L_2 = 13,\ L = 100$ and arrived at similar results: with probability at least 92%, the AAG protocol with these parameters can be successfully attacked by our procedure.

The arrangement of the paper is as follows. In Section 2, we introduce some more notation and describe an algorithm from [13] producing a shorter word representing a given braid word. In Section 3, we describe a heuristic procedure which allows us to simplify a given set of generators of a subgroup in B_n. In Section 4, we describe results of our experiments. In Section 5, we explain how these experimental results affect the security of the AAG protocol. Finally, in Section 6, we describe our procedure for producing random subgroup generators as in the AAG protocol.

2 Preliminaries

Let $F(X_n)$ be the free group generated by X_n. An element of $F(X_n)$ is a reduced word over $X_n^{\pm 1}$ referred to as a *braid word*. For a braid word $w = w_1 \dots w_k \in F(X_n)$ we will denote by $|w|$ its length k and by $|w|_{B_n}$ the length of a shortest braid word w' defining the same element of B_n as w does. There is no efficient way to compute $|w|_{B_n}$; in [14] the authors prove that the problem of computing a geodesic for a braid word is co-NP-complete. We will employ Algorithm 1 from [13] to obtain a shorter word representing a given braid word w; description of this algorithm is given below, for the sake of completeness. For relatively short words w considered in this paper, one almost always has $|Shorten(w)| = |w|_{B_n}$ (where $|Shorten(w)|$ is the output of Algorithm 1 in [13]; see [13] for more information).

By Dehornoy's form of a braid we mean a braid word without any "handles", i.e. a completely reduced braid word in the sense of [4]. The procedure that computes Dehornoy's form for a given word chooses a specific ("permitted") handle inside of the word and removes it (see [1] or [4]). This can introduce new handles but the main result about Dehornoy's forms states that any sequence of handle reductions eventually terminates. Of course, the result depends on how one chooses the handles at every step. Let us fix any particular strategy for selecting handles. For a word $w = w(X_n)$ we denote by $D(w)$ the corresponding Dehornoy's form (i.e., the result of handle reductions where handles are chosen by the fixed strategy).

Now we describe Algorithm 1 from [13]. This algorithm tries to minimize a given braid word. It uses the property of Dehornoy's form that for a "generic" braid word one has $|D(w)| < |w|$.

Algorithm 1 *(*Minimization of braids*)*
SIGNATURE. $w' = Shorten(w)$.

Input. *A word $w = w(x_1, \ldots, x_{n-1})$ in generators of the braid group B_n.*
Output. *A word w' such that $|w'| \le |w|$ and $w' = w$ in B_n.*
Initialization. *Put $w_0 = w$ and $i = 0$.*
Computations.

A. Increment i.
B. Put $w_i = D(w_{i-1})$.
C. If $|w_i| < |w_{i-1}|$ then
  1) Put $w_i = w_i^{\Delta}$.
  2) Goto A.
D. If i is even then output w_{i+1}^{Δ}.
E. If i is odd then output w_{i+1}.

3 Subgroup simplification

In this section we describe a heuristic procedure which allows us to simplify a given set of generators of a subgroup in B_n.

3.1 Reducing generating sets

Let S be a set of words in the alphabet X_n. We say that the set S is *reduced* if:

1) $|w| = |w|_{B_n}$ for each $w \in S$, i.e., each word from S is geodesic in B_n.
2) For each pair of words $u, v \in S$ and any numbers $\varepsilon, \delta \in \{-1, 1\}$, one has

$$|u^{\varepsilon} v^{\delta}|_{B_n} > ||u|_{B_n} - |v|_{B_n}|.$$

(Otherwise, the total length of elements of S can be reduced by replacing the longer of the words u, v by $u^{\varepsilon} v^{\delta}$).

Let $\langle S \rangle$ denote the subgroup generated by S. We say that two sets $S, T \subseteq B_n$ are *equivalent* if $\langle S \rangle = \langle T \rangle$ in B_n.

The following algorithm tries to reduce a given set S, i.e., tries to find a reduced set equivalent to S. As mentioned above, the problem of finding a geodesic for a given braid word is computationally hard. Instead, we are using here the procedure *Shorten* (Algorithm 1 in [13]) to minimize the length of braid words. Thus, in general, the output of Algorithm 2 may not be a reduced set of braid words, but for generating sets meeting the requirements in [1], this is usually the case.

Algorithm 2 *(*Reduction of a generating set*)*
Signature. *$T = Reduce(S)$.*
Input. *A finite set S of braid words.*
Output. *A finite reduced set T of braid words which is equivalent to S.*
Initialization. *Put $T = S$.*
Computations.

A. *For each word $w \in T$, replace w with the word Shorten(w) (cf. Algorithm 1 in [13]). Remove the empty word if produced.*

B. *For each pair of words $u, v \in T$ and numbers $\varepsilon, \delta \in \{-1, 1\}$, compute $w =$ Shorten$(u^\varepsilon v^\delta)$.*

  1) If $|w| = ||u| - |v|| = 0$ then remove v from the current set T.

  2) If $|w| = ||u| - |v|| \neq 0$ then remove from T the longer of the words u, v and add w to T.

C. *When all pairs of words $u, v \in T$ are handled (including the new words) output the current set T.*

Proposition 1. *Algorithm 2 terminates on any finite subset S of $F(X_n)$. Furthermore, if $T =$ Reduce(S), then $\langle T \rangle = \langle S \rangle$.*

Proof. Since each reduction decreases the total length of the generating set, the number of reductions Algorithm 2 performs is finite and limited by $L(S)$, the total length of elements of S.

To prove the second statement observe that the transformations used in Algorithm 2 are Nielsen transformations; they do not change the subgroup generated by a given set.

Let $(\overline{a}, \overline{a}')$ be a pair of conjugate tuples of braid words and $(\overline{z}, \overline{z}')$ be a pair of conjugate tuples of braid words. We say that tuples $(\overline{a}, \overline{a}')$ and $(\overline{z}, \overline{z}')$ are equivalent if the following conditions hold:

(E1) The tuples $\overline{a}$ and $\overline{z}$ define the same subgroup (i.e., $\langle \overline{a} \rangle = \langle \overline{z} \rangle$).

(E2) For any braid word $x \in B_n$ $x^{-1}\overline{a}x = \overline{a}'$ if and only if $x^{-1}\overline{z}x = \overline{z}'$.

Observe that from (E1), (E2), and the fact that tuples are conjugate follows that $\langle \overline{a}' \rangle = \langle \overline{z}' \rangle$.

Now assume that we have two conjugate tuples $\overline{a}$ and $\overline{a}'$ of braid words as in the AAG protocol. The next algorithm reduces the pair $(\overline{a}, \overline{a}')$.

Algorithm 3 *(*Reduction of conjugate tuples*)*
Signature. $(\overline{z}, \overline{z}') =$ *Reduce*$(\overline{a}, \overline{a}')$.
Input. *Conjugate tuples $\overline{a} = \{a_1, \ldots, a_{N_1}\}$ and $\overline{a}' = \{a'_1, \ldots, a'_{N_1}\}$ of braid words.*
Output. *Conjugate tuples $(\overline{z}, \overline{z}')$ equivalent to $(\overline{a}, \overline{a}')$.*
Initialization. *Put $\overline{z} = \overline{a}$ and $\overline{z}' = \overline{a}'$.*
Computations.

A. *Replace each word $z_i \in \overline{z}'$ with the word Shorten(z_i) (cf. Algorithm 1 in [13]) and each $z'_i \in \overline{z}'$ with Shorten(z'_i). Remove empty words if produced.*

B. *For each pair of words $z_i, z_j \in \overline{z}$ $(i \neq j)$ and numbers $\varepsilon, \delta \in \{-1, 1\}$, compute $w =$ Shorten$(z_i^\varepsilon z_j^\delta)$.*

  1) If $|w| = ||z_j| - |z_i|| = 0$, then remove z_i from $\overline{z}$ and remove z'_i from $\overline{z}'$.

  2) If $|w| = |z_j| - |z_i| > 0$, then replace $z_j \in \overline{z}$ with w and replace $z'_j \in \overline{z}'$ with Shorten$(z_i'^\varepsilon z_j'^\delta)$.

C. Repeat Step B. while applicable (i.e., while the set S' keeps changing).
D. Output the obtained set S'.

Proposition 2. *Let $(\overline{a}, \overline{a}')$ be a pair of conjugate tuples. Algorithm 3 terminates on $(\overline{a}, \overline{a}')$. Furthermore, if $(\overline{z}, \overline{z}') = Reduce(\overline{a}, \overline{a}')$, then $(\overline{z}, \overline{z}')$ is equivalent to $(\overline{a}, \overline{a}')$.*

Proof. The transformations used in Algorithm 3 are Nielsen transformations; they do not change the subgroup generated by a given set. Hence (E1) holds. Furthermore, by transforming $z_i \in \overline{z}$, we transform $z_i' \in \overline{z}'$ the same way. Thus, the property (E2) holds and the output $(\overline{z}, \overline{z}')$ is equivalent to the input $(\overline{a}, \overline{a}')$.

3.2 Extending generating sets

We say that a set $S \cup S'$ is an *extension* of S. The next algorithm heuristically extends a reduced set of generators S by adding words (one at a time) of length 2 from the subgroup $\langle S \rangle$, and then reduces the set. Basically, the algorithm generates words from $\langle S \rangle$ using a few patterns and, in case a new word has length 2, adds it to the current set and reduces the result.

Algorithm 4 *(*Extension of a generating set*)*
SIGNATURE. *$T = Extend(S)$.*
INPUT. *A set S of braid words.*
OUTPUT. *A reduced set S' of braid words equivalent to S.*
INITIALIZATION. *Put $T = S$.*
COMPUTATIONS.

A. For each pair of words $(u, v) \in T$, and each pair of numbers $\varepsilon, \delta \in \{-1, 1\}$:
   1) Compute $w = Shorten(v^{2\varepsilon} u^{\delta} v^{-\varepsilon} u^{-\delta} v^{-\varepsilon})$ and $T' = Reduce(T \cup \{w\})$. If $|w| = 2$ and $T \neq T'$, then put $T = T'$.
   2) Compute $w = Shorten(v^{\varepsilon} u^{\delta})$ and $T' = Reduce(T \cup \{w\})$. If $|w| = 2$ and $T \neq T'$, then put $T = T'$.
B. When all pairs of words $u, v \in T$ are handled (including the new ones), output the current set T.

Proposition 3. *Algorithm 4 terminates on any finite set S of braid words and, if $T = Extend(S)$, then $\langle S \rangle = \langle T \rangle$.*

Proof. The latter statement is obviously true by Proposition 1 and since each braid word we add to T defines an element of $\langle S \rangle$.

Note that Algorithm 4 extends the current set T with braid words w of length 2 only. Moreover, a word $w = x_i^{\varepsilon} x_j^{\delta}$ of length 2 cannot be added twice (the second time $T' = T$). Thus, Algorithm 4 can add at most $4n^2$ new words to T.

Now assume that we have two conjugate tuples $\overline{a}$ and $\overline{a}'$ of braid words as in the AAG protocol. The next algorithm computes an extended conjugated pair of tuples $(\overline{z}, \overline{z}')$ equivalent to $(\overline{a}, \overline{a}')$. In Algorithm 5, for a tuple $\overline{a} = (a_1, \ldots, a_k)$ and a braid word w, by $\overline{a} \cup w$ we denote a tuple $(a_1, \ldots, a_k, w)$.

Algorithm 5 *(*Extension of conjugate tuples*)*
SIGNATURE. $(\overline{z}, \overline{z}') = Extend(\overline{a}, \overline{a}')$.
INPUT. *Conjugate tuples* $\overline{a} = \{a_1, \ldots, a_k\}$ *and* $\overline{a}' = \{a_1', \ldots, a_k'\}$ *of braid words.*
OUTPUT. *Conjugate "extended" tuples* $(\overline{z}, \overline{z}')$ *equivalent to* $(\overline{a}, \overline{a}')$.
INITIALIZATION. *Put* $\overline{z} = \overline{a}$ *and* $\overline{z}' = \overline{a}'$.
COMPUTATIONS.

A. For each distinct pair of words $(z_i, z_j) \in \overline{a}$, *and each pair of numbers* $\varepsilon, \delta \in \{-1, 1\}$:
   1) *Perform the following:*
      – *Compute* $w = Shorten(z_i^{2\varepsilon} z_j^{\delta} z_i^{-\varepsilon} z_j^{-\delta} z_i^{-\varepsilon})$.
      – *Compute* $w' = Shorten(z_i'^{2\varepsilon} z_j'^{\delta} z_i'^{-\varepsilon} z_j'^{-\delta} z_i'^{-\varepsilon})$.
      – *Compute* $(\overline{y}, \overline{y}') = Reduce(\overline{z} \cup \{w\}, \overline{z}' \cup \{w'\})$. *If* $|w| = 2$ *and* $(\overline{z}, \overline{z}') \neq (\overline{y}, \overline{y}')$, *then put* $(\overline{z}, \overline{z}') = (\overline{y}, \overline{y}')$.
   2) *Perform the following:*
      – *Compute* $w = Shorten(z_i^{\varepsilon} z_j^{\delta})$.
      – *Compute* $w' = Shorten(z_i'^{\varepsilon} z_j'^{\delta})$.
      – *Compute* $(\overline{y}, \overline{y}') = Reduce(\overline{z} \cup \{w\}, \overline{z}' \cup \{w'\})$. *If* $|w| = 2$ *and* $(\overline{z}, \overline{z}') \neq (\overline{y}, \overline{y}')$, *then put* $(\overline{z}, \overline{z}') = (\overline{y}, \overline{y}')$.

B. When all pairs of words $z_i, z_j \in S$ *are handled (including the new words), output the current pair* $(\overline{z}, \overline{z}')$.

Proposition 4. *Let* $(\overline{a}, \overline{a}')$ *be a pair of conjugate tuples. Algorithm 5 terminates on* $(\overline{a}, \overline{a}')$. *Furthermore, if* $(\overline{z}, \overline{z}') = Extend(\overline{a}, \overline{a}')$, *then* $(\overline{z}, \overline{z}')$ *is equivalent to* $(\overline{a}, \overline{a}')$.

Proof. Each time we extend the tuples $(\overline{z}, \overline{z}')$ with elements w, w' which follows from the tuples (i.e., $w \in \langle \overline{z} \rangle$ and $w' \in \langle \overline{z}' \rangle$). So, now the property (E1) follows from Proposition 2. Furthermore, braid words w and w' were obtained the same way. Thus, the property (E2) holds and the output $(\overline{z}, \overline{z}')$ is equivalent to the input $(\overline{a}, \overline{a}')$.

We therefore have

Proposition 5. *Let* $(\overline{a}, \overline{a}')$ *and* $(\overline{b}, \overline{b}')$ *be two pairs of conjugated tuples as in AAG-protocol. Let* $(\overline{y}, \overline{y}') = Extend(\overline{a}, \overline{a}')$ *and* $(\overline{z}, \overline{z}') = Extend(\overline{b}, \overline{b}')$. *Then to break the AAG protocol with* $(\overline{a}, \overline{a}')$ *and* $(\overline{b}, \overline{b}')$ *it is sufficient to break AAG-protocol with* $(\overline{y}, \overline{y}')$ *and* $(\overline{z}, \overline{z}')$.

Proof. Obvious.

The main point of Proposition 5 is that the obtained instance $(\overline{y}, \overline{y}')$ and $(\overline{z}, \overline{z}')$ of the AAG protocol is easier to break than the original $(\overline{a}, \overline{a}')$ and $(\overline{b}, \overline{b}')$. (It will be clear from the experimental results described in the next section.) Furthermore, $(\overline{y}, \overline{y}')$ and $(\overline{z}, \overline{z}')$ can be computed quite efficiently.

4 Experimental results

We performed a series of 100 experiments with randomly generated subgroups of B_{80}. In each experiment we

1) Generated Alice's and Bob's public and private keys $\overline{a}$, $\overline{b}$, A, B (as described in the Introduction).
2) Computed $\overline{a}'$ and $\overline{b}'$.
3) Computed $(\overline{y}, \overline{y}') = Extend(\overline{a}, \overline{a}')$.

The obtained sets of results are as follows:

1) In 63 cases, $\overline{y} = (x_1, \ldots, x_{79})$.
2) In 25 cases, $\overline{y} = (x_1, \ldots, x_{i-1}, x_i^2, x_{i+1}, \ldots, x_{79})$ for some i.
3) In 5 cases, $\overline{y} = (x_1, \ldots, x_{i-1}, x_i^2, x_{i+1}, \ldots, x_{j-1}, x_j^2, x_{j+1}, \ldots, x_{79})$ for some i, j.
4) In 5 cases, $\overline{y} = (x_1, \ldots, x_{i-1}, x_i^2, x_i x_{i+1}^2 x_i, x_{i+2}, \ldots, x_{79})$ for some i.
5) In 1 case,
$\overline{y} = (x_1, \ldots, x_{i-1}, x_i^2, x_i x_{i+1}^2 x_i, x_{i+1}, \ldots, x_{j-1}, x_j^3, x_{j+1}, \ldots, x_{79})$ for some i, j.
6) In 1 case, $\overline{y} = (x_1, \ldots, x_{i-1}, x_i^{-1} x_{i+1} x_i, x_{i+2} \ldots, x_{79})$ for some i.

Thus, randomly generated tuples of braid words $\overline{a}$ and $\overline{b}$ of "AAG-type" generate either the whole B_n or a subgroup which is "close" to the whole group B_n.

To explain this phenomenon, consider two particular braid words in B_{80}:

$$w_1 = x_{71} x_{47} x_{11} x_{45}^{-1} x_9 x_6 x_{72}^{-1} \quad \text{and} \quad w_2 = x_{64} x_{32}^{-1} x_{39}^{-1} x_{17} x_8 x_{26} x_{31}^{-1} x_{78}.$$

It is easy to check that $w_1^2 w_2 w_1^{-1} w_2^{-1} w_1^{-1} = x_9^2 x_8 x_9^{-1} x_8^{-1} x_9^{-1} = x_9 x_8^{-1}$. This happens, basically, because all generators in w_1 commute with all generators in w_2 except x_8 which does not commute with x_9.

In general, if we pick two random braid words w_1 and w_2 (of length $5-8$ over the alphabet $\{x_1, \ldots, x_{79}\}$) in $\overline{a}$ such that w_1 contains some fixed generator $x_i^{\pm 1}$ and w_2 contains $x_{i+1}^{\pm 1}$, then there is a big chance that all other generators that occur in w_1 or w_2 commute with each other and with x_i and x_{i+1}. In other words, for each $1 \le i \le 79$, with significant probability, there are two words w_1 and w_2 such that

1. $w_1 = w_1' x_i^{\pm 1} w_1''$;
2. $w_2 = w_2' x_{i+1}^{\pm 1} w_2''$;
3. x_i commutes with w_1', w_2', w_1'', and w_2'';
4. x_{i+1} commutes with w_1', w_2', w_1'', and w_2'';
5. w_1' commutes with w_2' and w_2'', and w_1'' commutes with w_2' and w_2''.

In this case, for some $\varepsilon, \delta \in \{-1, 1\}$, we have $w_1^{2\varepsilon} w_2^{\delta} w_1^{-\varepsilon} w_2^{-\delta} w_1^{-\varepsilon} = x_i^{\varepsilon} x_{i+1}^{-\varepsilon}$.

Somewhat informally, Algorithm 5 works as follows. First, a lot of words of the form $x_i^{\varepsilon} x_{i+1}^{-\varepsilon}$ are being produced (using the pattern $v^{2\varepsilon} u^{\delta} v^{-\varepsilon} u^{-\delta} v^{-\varepsilon}$). Then, using generators of the form $x_i^{\varepsilon} x_{i+1}^{-\varepsilon}$, Algorithm 4 produces all kinds of generators of the form $x_i^{\varepsilon} x_j^{\delta}$ (using the pattern $v^{\varepsilon} u^{\delta}$). Finally, after sufficient number of words of length 2 is produced, the algorithm reduces the initial subgroup generators to generators of the whole group B_n.

Remark 1. We note that increasing the parameters L_1 and L_2 decreases the probability for pairs of words w_1, w_2 to satisfy the properties (1)–(5) above. However, if the increase is moderate, then it is quite likely that w_1 and w_2 will contain two pairs of non-commuting generators, say, x_i, x_j in w_1 and $x_{i\pm1}$, $x_{j\pm1}$ in w_2. Then, for some $\varepsilon, \delta \in \{-1, 1\}$, we have $w_1^{2\varepsilon} w_2^{\delta} w_1^{-\varepsilon} w_2^{-\delta} w_1^{-\varepsilon} = x_i^{\varepsilon} x_{i\pm1}^{-\varepsilon} x_j^{\varepsilon} x_{j\pm1}^{-\varepsilon}$ which is a word of length 4. In this case, performance of the algorithm 5 can be improved by allowing to add words of length 4 to the generating set (at the cost of somewhat reducing the speed of computation). As the parameter values L_1 and L_2 are increased further, the pattern $w_1^{2\varepsilon} w_2^{\delta} w_1^{-\varepsilon} w_2^{-\delta} w_1^{-\varepsilon}$ produces longer and longer words, and for some of these words the algorithm may fail to prove that the relevant subgroup is the whole group B_n (sometimes the subgroup may actually be different from B_n).

We used this modification of the algorithm to test the following parameters: $N = 80$, $N_1 = 20$, $N_2 = 20$, and $L_1 = 11$, $L_2 = 13$. Even with the generators that long, many subgroups do generate the whole B_N. As we have mentioned before, further increase of the length of the generators can make the protocol vulnerable to length-based attacks.

5 The impact of the experimental results on the security of the AAG protocol

As described in the previous section, our experiments show that with the choice of parameters for the AAG protocol suggested in [1], the subgroups generated by $\overline{a}$ and $\overline{b}$ tend to have the following properties:

(G1) They are either the whole group B_n or "almost" the whole B_n.
(G2) They have cyclic centralizer which coincides with the center of B_n. (The latter is generated by the element Δ^2.)
(G3) They are generated by short (of length up to 3) positive braid words.

Furthermore, Algorithm 5 efficiently transforms an initial generating tuple into a simplified generating tuple of type (G1)–(G3). In this section, we explain how these results affect the security of the AAG protocol. The techniques used in this section were developed by S. J. Lee and E. Lee in [12] and by J. Gonzalez-Meneses in [7]. We refer the reader to these two papers for more information on the algorithms; here we just recall some notation that we need.

For $a \in B_n$, the number $\inf(a)$ denotes the maximum integer k such that $a = \Delta^k p$ in the group B_n, where $\Delta \in B_n$ is the half-twist braid and p is a positive braid. For an r-tuple of braids $\overline{a} = (a_1, \ldots, a_r)$, denote by $C^{inf}(S)$ the set of all r-tuples $(b_1, \ldots, b_r) \in B_n^r$ such that $\inf(b_i) \geq \inf(a_i)$ $(i = 1, \ldots, r)$ and there exists $w \in B_n$ such that $\overline{b} = w^{-1}\overline{a}w$.

The following algorithm combines two ingredients: the subgroup simplification algorithm of the present paper and the summit attack of [7], [12] into one attack on Alice's (or Bob's) key.

Algorithm 6 *(Attack on AAG-protocol)*
SIGNATURE. $w = GetConjugator(\overline{a}, \overline{a}')$.
INPUT. *Conjugate tuples* $(\overline{a}, \overline{a}')$ *of AAG-type.*
OUTPUT. *A braid word* w *such that* $\overline{a}' = w^{-1}\overline{a}w$.
COMPUTATIONS.

A. *Compute* $(\overline{a}_1, \overline{a}_1') = Extend(\overline{a}, \overline{a}')$.
B. *Using technique from [12], compute* $(\overline{a}_2, \overline{a}_2')$*,* u*, and* v *satisfying the following properties:*
   1) $\overline{a}_2 \in C^{inf}(\overline{a}_2') \subseteq C^{inf}(\overline{a}_1')$ *and* $\overline{a}_2' \in C^{inf}(\overline{a}_2) \subseteq C^{inf}(\overline{a}_1)$.
   2) $\overline{a}_2 = u^{-1}\overline{a}_1 u$.
   3) $\overline{a}_2' = v^{-1}\overline{a}_1' v$.
C. *Using technique from [7], compute a braid word* s *such that* $\overline{a}_2 = s^{-1}\overline{a}_2' s$.
D. *Output* $us^{-1}v^{-1}$.

By Theorem 2 of [12], the step B of Algorithm 6 can be performed very efficiently (by a polynomial time algorithm). The time complexity of the step C is proportional to the size of $C^{inf}(\overline{a}_1)$ which is large in general, but for all subgroups obtained in our experiments these sets were small. For instance, if the tuple $\overline{a}$ consists of all generators of B_n, then $|C^{inf}(\overline{a}_1)| = 2$ as shown in the next proposition.

Proposition 6. *Let* $\overline{x} = (x_1, \ldots, x_{n-1})$*. Then* $C^{inf}(\overline{x}) = \{\overline{x}, \Delta^{-1}\overline{x}\Delta\}$.

Proof. Let $c_0 \in B_n$ be such that $c_0^{-1}\overline{x}c_0 \in C^{inf}(\overline{x})$. Then for each $i = 1, \ldots, n-1$ one has

$$c_0^{-1}x_i c_0 = x_{s_i}$$

for some $1 \leq s_i \leq n-1$. Since conjugation is an automorphism, it is easy to see that either $(s_1, \ldots, s_{n-1}) = (1, \ldots, n-1)$ or $(s_1, \ldots, s_{n-1}) = (n-1, \ldots, 1)$ which proves the proposition.

For other generating tuples obtained in our experiments the sizes of the summit set $C^{inf}(\overline{x})$ are small, too. Therefore, we can say that Algorithm 6 is efficient on a randomly generated subgroup as described in the AAG protocol. We should mention that the obtained conjugator may not be exactly Alice's (Bob's) private key; we compute it up to the centralizer of Bob's (Alice's) subgroup. However, since in almost all examples the centralizer is generated by the element Δ^2 (i.e., coincides with the center of B_n), this is not a problem. We would like to point out that without the first step the attack may not be efficient since the size of the summit set would be huge.

Now, with Algorithm 6 it is easy to find the shared key obtained by Alice and Bob in the AAG protocol:

Algorithm 7 *(Attack on the AAG protocol)*
SIGNATURE. $w = GetSharedKey(\overline{a}, \overline{a}', \overline{b}, \overline{b}')$.
INPUT. *Conjugate tuples* $(\overline{a}, \overline{a}')$ *and* $(\overline{b}, \overline{b}')$ *of as in the AAG protocol.*
OUTPUT. *The shared key* K.
COMPUTATIONS.

A. Let $w_a = GetConjugator(\overline{a}, \overline{a}')$.
B. Let $w_b = GetConjugator(\overline{b}, \overline{b}')$.
C. Output $w_a^{-1} w_b^{-1} w_a w_b$.

6 Appendix: generating random subgroups

The question of how one could produce a random generating set of a required type for a subgroup of B_n is by no means trivial. We used the following procedure for producing random subgroup generators as in the AAG protocol. In the description of the algorithm below, when we say "uniformly choose an integer" from a given interval, that means all integers from this interval are selected with equal probabilities.

Algorithm 8 *(*Subgroup generator*)*
INPUT. *The rank* n *of the braid group, the rank* k *of a subgroup, and numbers* L_1, L_2 *such that* $L_1 < L_2$.
OUTPUT. *Braid words* $w_1, \ldots, w_k$ *over* X_n *such that* $L_1 \le |w_i| \le L_2$ *and each generator* $x \in X_n$ *non-trivially occurs in at least one of the* w_i*'s.*
COMPUTATIONS.

A. For each $1 \le i \le k$, *uniformly choose an integer* l_i, $L_1 \le l_i \le L_2$, *and compute* $L = \sum_{i=1}^{k} l_i$.
B. Construct a sequence $\{a_1, \ldots, a_L\} \in (X^{\pm 1})^*$ *the following way:*
  1) For each $1 \le i \le n-1$, *uniformly choose* $\varepsilon_i \in \{-1, 1\}$ *and put* $a_i = x_i^{\varepsilon_i}$.
  2) For each $n \le i \le L$, *uniformly choose* $j_i \in \{1, \ldots, n-1\}$ *and* $\varepsilon_i \in \{-1, 1\}$, *and put* $a_i = x_{j_i}^{\varepsilon_i}$.
C. Randomly permute elements in $\{a_1, \ldots, a_L\}$.
D. For each $1 \le j \le k$, *compute* $s_j = \sum_{i=1}^{j-1} l_i$ *and put* $w_j = Shorten(a_{(s_j)+1} \ldots a_{s_{(j+1)}})$.
E. If some braid generator x_i *does not occur in the obtained sequence* $w_1, \ldots, w_k$, *then repeat all the steps.*

Note that, in theory, Algorithm 8 might go into an infinite loop if the subgroup generators $\{w_1, \ldots, w_k\}$ do not involve some braid generator x_i. But in real life, such a situation is extremely rare. In fact, the greatest number of iterations Algorithm 8 performed in our experiments was 5.

References

1. I. Anshel, M. Anshel, B. Fisher, D. Goldfeld, *New Key Agreement Protocols in Braid Group Cryptography.* In: Progress in Cryptology – CT-RSA 2001, 13–27. Lecture Notes Comp. Sc., vol. 2020. Berlin Heidelberg New York Tokyo: Springer 2001.
2. I. Anshel, M. Anshel, D. Goldfeld, *An algebraic method for public-key cryptography,* Math. Res. Lett. **6** (1999), 287–291.

3. J. S. Birman, *Braids, links and mapping class groups*, Ann. Math. Studies **82**, Princeton Univ. Press, 1974.
4. P. Dehornoy, *A fast method for comparing braids*, Adv. Math. **125** (1997), 200–235.
5. D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson, W. P. Thurston, *Word processing in groups.* Jones and Bartlett Publishers, Boston, MA, 1992.
6. D. Garber, S. Kaplan, M. Teicher, B. Tsaban, U. Vishne, *Probabilistic solutions of equations in the braid group*, preprint. http://arxiv.org/abs/math.GR/0404076
7. J. Gonzalez-Meneses, *Improving an algorithm to solve Multiple Simultaneous Conjugacy Problems in braid groups*, Contemp. Math., Amer. Math. Soc. **372** (2005), 35–42.
8. J. Gonzalez-Meneses and B. Wiest, *On the structure of the centraliser of a braid*, Ann. Sci. École Norm. Sup. **37** (5) (2004), 729–757.
9. Hofheinz, D., Steinwandt, R., *A practical attack on some braid group based cryptographic primitives.* In: Public Key Cryptography, 6th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2003 Proceedings, 187–198 (Y. G. Desmedt, ed., Lecture Notes Comp. Sc., vol. 2567) Berlin Heidelberg New York Tokyo: Springer 2002.
10. Hughes, J., Tannenbaum, A., *Length-based attacks for certain group based encryption rewriting systems.* In: Workshop SECI02 Securitè de la Communication sur Intenet, September 2002, Tunis, Tunisia. http://www.network.com/˜hughes/
11. K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J. Kang, C. Park, *New public-key cryptosystem using braid groups.* In: Advances in cryptology – CRYPTO 2000 (Santa Barbara, CA), 166–183 (Lecture Notes Comp. Sc., vol. 1880) Berlin Heidelberg New York Tokyo: Springer 2000.
12. S. J. Lee, E. Lee, *Potential Weaknesses of the Commutator Key Agreement Protocol Based on Braid Groups.* In: Advances in cryptology – EUROCRYPT 2002, 14–28 (Lecture Notes Comp. Sc., vol. 2332) Berlin Heidelberg New York Tokyo: Springer 2002.
13. A. Myasnikov, V. Shpilrain, A. Ushakov, *A practical attack on some braid group based cryptographic protocols.* In: Advances in cryptology – CRYPTO 2005 (Santa Barbara, CA). Lecture Notes Comp. Sc. **3621** (2005), 86–96.
14. M. Paterson, A. Razborov, *The set of minimal braids is co-NP-complete*, J. Algorithms **12** (1991), 393–408.
15. V. Shpilrain and A. Ushakov, *The conjugacy search problem in public key cryptography: unnecessary and insufficient*, Applicable Algebra in Engineering, Communication and Computing, to appear. http://eprint.iacr.org/2004/321/