Length Based Attack and Braid Groups: Cryptanalysis of Anshel-Anshel-Goldfeld Key Exchange Protocol

Alex D. Myasnikov and Alexander Ushakov

Department of Mathematical Sciences, Stevens Institute of Technology, Hoboken, New Jersey, USA, 07030,
[amyasnik,aushakov]@stevens.edu

Abstract. The length based attack on Anshel-Anshel-Goldfeld commutator key-exchange protocol [1] was initially proposed by Hughes and Tannenbaum in [9]. Several attempts have been made to implement the attack [6], but none of them had produced results convincing enough to believe that attack works. In this paper we show that accurately designed length based attack can successfully break a random instance of the simultaneous conjugacy search problem for certain parameter values and argue that the public/private information chosen uniformly random leads to weak keys.

1 Introduction

Braid group cryptography has attracted a lot of attention recently due to several suggested key exchange protocols (see [1], [10]) using braid groups as a platform. We refer to [2], [5] for more information on braid groups.

In this paper we discuss the so-called Length Based Attack on the Anshel-Anshel-Goldfeld key exchange protocol [1] (subsequently called the AAG protocol). The Length Based Attack, LBA for short, was first introduced by Hughes and Tannenbaum in [9], however no actual experiments were performed and the real threat of the attack has not been evaluated. Since then there were several implementations of LBA published [6] but none of them produced a convincing evidence that LBA, indeed, breaks AAG. Finally, the authors of [6] make conclusion that AAG protocol is invulnerable to LBA.

We need to mention here that successful attacks on AAG were proposed in [7, 11, 14]. It is common believe now that AAG with original parameters is not secure. However, the scalability of the attacks has not been completely realized. This leads to speculations that AAG protocol may still be secure with a different set of parameters such as longer private keys, for example.

In the paper we analyze the reasons behind the failure of the previous implementations of LBA. We show that for slightly increased values of parameters LBA can be modified so it breaks AAG protocol with a very high rate of success. We also present an evidence that the keys generated uniformly randomly are not secure and suggest that a more cautious approach in selecting private information is necessary for AAG protocol to be immune to the length based attack.

Here we start out by giving a brief description of the Anshel-Anshel-Goldfeld key exchange protocol [1] (subsequently called the AAG protocol). 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 [1] 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*.

(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 $0 < s_i < N_1$ and $\varepsilon_i = \pm 1$ (for each $1 \leq i \leq L$). 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 $0 < t_i < N_1$ and $\delta_i = \pm 1$ (for each $1 \leq i \leq L$). The word B is called *Bob's private key*.

(5) Alice computes $b_i' = D(A^{-1} b_i A)$ $(1 \leq i \leq N_2)$ and transmits them to Bob. Here $D(w)$ denotes Dehornoy handle free form of a braid word w (see [4] for the definition of Dehornoy form of a braid).

(6) Bob computes $a_i' = D(B^{-1} a_i B)$ $(1 \leq i \leq N_1)$ 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 $K_A = A^{-1} B^{-1} A B$ 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 $K_B = A^{-1} B^{-1} A B$ in the group B_n.

Thus, Alice and Bob obtain the same element $K = K_A = K_B = A^{-1} B^{-1} A B$ of the group B_n. This K is now their *shared secret key*.

In the steps (5) and (6) of the protocol the so-called Dehronoy form is used do diffuse the public commutators. It is out of scope of this paper to define the Dehornoy form in detail. Informally, the Dehornoy form is a reduced braid word obtained as a result of a particular rewriting procedure. It is believed that Dehornoy forms are linearly computable and it is computationally infeasible to reconstruct the original braid from its Dehornoy form. For more details on the definition and the procedure to compute the Dehornoy form we refer to [4].

Note that for an intruder to get the shared secret key K, it is sufficient to find:

– an element $A' \in \langle a_1, \ldots, a_{N_1} \rangle$ such that $\overline{b}' = A'^{-1}\overline{b}A'$ in B_n;
– an element $B' \in \langle b_1, \ldots, n_{N_2} \rangle$ such that $\overline{a}' = B'^{-1}\overline{a}B'$ in B_n.

Such elements A' and B' successfully substitute Alice's and Bob's private keys A and B, in particular, $[A, B] = [A', B']$. For more information see [16]. Finding an element A' (and B') is an instance of the *subgroup-restricted simultaneous conjugacy search problem* (abbreviated SR-SCSP) which is a variation of *simultaneous conjugacy search problem* (SCSP) where it is required to find any conjugator for two conjugated tuples.

Therefore, we say that the security of AAG protocol is partially based (but not equivalent) on the assumption that SR-SCSP is hard. Below we describe several types of attacks on variations of simultaneous conjugacy problem.

A. There is only one attack aiming to break SR-SCSP directly – the length-based attack (initially proposed in [9]). It is a heuristic descend method for solving SR-SCSP. We discuss it at length in Section 2.
B. All other attacks are aiming at SCSP:
   1) *Summit Set Attack* [11]. This method starts by reducing conjugates to the minimal level with respect to the canonical length (called the summit set) and then performs the exhaustive search in that level.
   2) *Hofheinz-Stainwandt Attack* [7] which has the same first step as in the summit set attack and then uses a heuristic to obtain a solution in the minimal level.
   3) *Linear Attack* which uses presentations of braids by matrices, e.g., Burau or Kramer presentations (see [8]). This attack produces a conjugator in a matrix form and further lifting to braids is required.

A different type of heuristic attacks which is called *the subgroup attack* was presented in [14]. It does not solve any variation of the conjugacy problem. Instead it reduces the original problem to the one with shorter generators simplifying the conjugacy problem. In particular, using the subgroup attack it was shown that for parameters originally proposed by Anshel-Anshel-Goldfeld

– SCSP and SR-SCSP are equivalent for majority of random public sets;
– the majority of random public sets define the same subgroup which coincides with the whole group;

which justifies the success of attacks B.1), B.2), and B.3) which perform well, although with different success rates, on the original parameters suggested in [1]:

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

It is well accepted now that these values of parameters do not provide good level of security. In this paper we increase values of parameters L_1 and L_2 to

$$n = 80, \quad N_1 = N_2 = 20, \quad L_1 = 20, 30, 40, \quad L_2 = L_1 + 3, \quad L = 50.$$

and show that accurately designed LBA can crack a random instance of the SR-SCSP generated using these values of parameters. Notice that we increase lengths of generators of the public sets but decrease lengths of decompositions of the private keys to keep the size of private keys A and B within practical bounds. To be more precise we got the following results in our experiments:

L_1, L_2	10,13	20,23	30,33	40,43
Success rate	00%	51%	97%	96%

See Table 1 for more details.

The rest of the paper is organized as follows. In Section 2 we describe the idea of the length based attack and its variations. We give examples of potentially hard instances and explain what prevents LBA from being successful. We conclude Section 2 by showing that it is unlikely that a private key taken at random will be hard to break when values of L_1 and L_2 are sufficiently large. We argue that a naive approach of increasing the size of the key will not guarantee increase in the security of the protocol. In Section 3 we describe our version of the generalized length based attack for breaking AAG and present experimental results.

All the algorithms described in this paper are available at [3].

Acknowledgments. We are grateful to the Algebraic Cryptography Center at Stevens Institute of Technology for support of our research. Also, we would like to thank anonymous reviewers for their valuable comments and suggestions on the paper.

2 The Length Based Attack

The length based attack is a heuristic procedure for finding the Alice's (symmetrically Bob's) private key A (B). Following the notation of Section 1 let $\overline{a} = \{a_1, \ldots, a_{N_1}\}$, $\overline{b} = \{b_1, \ldots, b_{N_2}\}$, $A = a_{s_1}^{\varepsilon_1} \ldots a_{s_L}^{\varepsilon_L}$, and $\overline{b}' = \{b'_1, \ldots, b'_{N_2}\}$, where $b'_i = D(A^{-1} b_i A)$. Essentially each b'_i is a result of a sequence of conjugations of b_i by the factors of A:

$$\begin{array}{rc}
 & b_i \\
 & \downarrow \\
 & a_{s_1}^{-\varepsilon_1} \;\; b_i \;\; a_{s_1}^{\varepsilon_1} \\
 & \downarrow \\
 & a_{s_2}^{-\varepsilon_2} a_{s_1}^{-\varepsilon_1} \;\; b_i \;\; a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \\
 & \downarrow \\
 & \ldots \\
 & \downarrow \\
b'_i = & a_{s_L}^{-\varepsilon_L} \ldots a_{s_2}^{-\varepsilon_2} a_{s_1}^{-\varepsilon_1} \;\; b_i \;\; a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} \ldots a_{s_L}^{\varepsilon_L}
\end{array} \tag{1}$$

A conjugating sequence is the same for each b_i and is defined by the private key A. The main goal of the attack is to reverse the sequence (1) and going back from the bottom to the top recover each conjugating factor. If successful the procedure will result in the actual conjugator as a product of elements from $\overline{a}$.

2.1 LBA as a minimization problem

To achieve the goal outlined above we need some efficiently computable function whose values would guide us on the way from the bottom to the top of (1). The most natural idea is to find a function l such that

$$\text{for the majority of elements } a, b \in B_n \ l(a^{-1}ba) > l(b). \tag{2}$$

If such function exists then LBA can be set as a minimization problem and solved using some heuristic optimization methods.

The choice of the function l is crucial for the success of the attack. In the original paper [9] it was proposed to use a length function. There are several length functions available for braids. In [9] the authors do not specify the function explicitly, although their arguments are based on the work of Vershik et al. [17] where the length defined as the geodesic length, i.e. the length of the shortest path in the corresponding Cayley graph of a group.

Unfortunately there are no practically useful length functions are known in braid groups which satisfy the criteria (2). The geodesic length of a braid denoted by $|\cdot|$ seems to be the best candidate. However, there is no known efficient algorithm for computing $|\cdot|$. Moreover, it was shown in [15] that the set of geodesic braids in B_∞ is co-NP complete.

Some of length functions such as the canonical length of the Garside normal form $|\cdot|_\Delta$ and the canonical length of the Birman-Ko-Lee normal form $|\cdot|_\delta$ are efficiently computable but very crude, in a sense that many braids consisting of many crossings have very small lengths. For instance, permutation braids contain up to $1/2n(n-1)$ crossings but have canonical length $|\cdot|_\Delta$ equal 1.

In this paper we use the method to approximate geodesic length proposed in [14]. It does not guarantee the optimal result, although a series of experiments show that for braids used in AAG the results of the approximation satisfy the desired property given by the relation (2). From now on we denote by $|\cdot|$ the result of the approximation function. The experiments suggest that our approximation function $|\cdot|$ satisfies $|a^{-1}ba| > |b|$ for almost all a and b. Moreover, as the length of a and b grows we have $2|a| + |b| - |a^{-1}ba|$ significantly smaller than $2|a|$ which means that $|a^{-1}ba| > |b|$ and the difference is large. Figure 1 shows the distribution of $2|a| + |b| - |a^{-1}ba|$ in B_{80} for $|b| = 400$, $|a| = 5, 10, 20, 30, 40$. In particular, for $|a| = 5$ we see that in 90% of the cases cancellation in $|a^{-1}ba|$ is limited by 4 symbols which means that in 90% of the cases conjugation by the element of length 5 increases the length by at least 6. The small fraction of elements which do not satisfy $|a^{-1}ba| > 2|a| + |b|$ (negative values in the distribution) are caused by the errors of the approximation.

2.2 Variations of LBA

In this section we discuss several heuristic approaches to be used with the length function $|\cdot|$. All the algorithms in this section have the following input/output:

- INPUT: Tuples $\overline{a} = (a_1, \ldots, a_{N_1})$, $\overline{b} = (b_1, \ldots, b_{N_2})$, and $\overline{b}' = (b'_1, \ldots, b'_{N_2})$ such that $\overline{b}$ and $\overline{b}'$ are conjugate by an element from $\langle a_1, \ldots, a_{N_1} \rangle$.

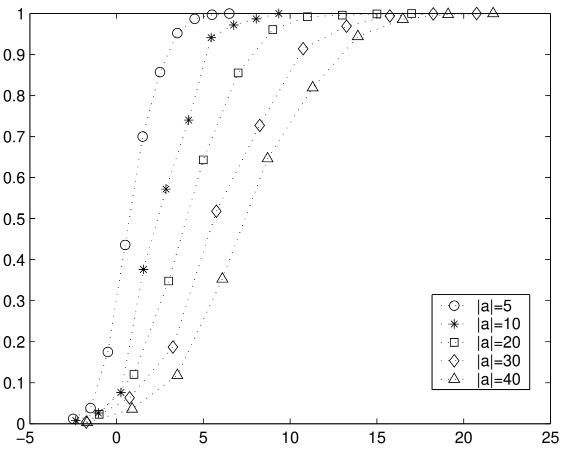

Fig. 1. Distribution of $2|a|+|b|-|a^{-1}ba|$ in B_{80} for $|b|=400$, $|a|=5,10,20,30,40$.

- OUTPUT: An element $x \in \langle a_1,\ldots,a_{N_1}\rangle$ such that $\overline{b}^x = \overline{b}'$ or $FAIL$ if algorithm is unable to find such x.

For an arbitrary tuple of braids $\overline{c} = (c_1,\ldots,c_k)$ denote by $|\overline{c}|$ its *total length* $\sum_{i=1}^k |c_i|$. Algorithm 1 (*LBA with backtracking*) enumerates all possible sequences of conjugations decreasing the length of a tuple. We maintain set S which contains tuples in work.

Algorithm 1 *(LBA with backtracking)*

A. Initialize a set $S = \{(\overline{b}', e)\}$, where e is the identity of B_n.
B. If $S = \emptyset$ then output $FAIL$.
C. Choose a pair $(\overline{c}, x) \in S$ with a minimal $|\overline{c}|$. Remove $(\overline{c}, x)$ from S.
D. For each $i = 1,\ldots,N_1$, $\varepsilon = \pm 1$ compute $\delta_{i,\varepsilon} = |\overline{c}| - |\overline{c}^{a_i^\varepsilon}|$.
E. If $\delta_{i,\varepsilon} > 0$ then add $(\overline{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$ into S.
F. If $\overline{c}^{a_i^\varepsilon} = \overline{a}$ them output xa_i^ε.
G. Otherwise goto B.

Algorithm 2 (*best descend LBA*) is a version of a length based attack where on each step we choose conjugator which gives the maximal decrease among all currently available tuples. It is weaker than Algorithm 1 but works well for certain parameter values as our experiments show. It has the same steps as Algorithm 1, except that on step E we add only the tuple corresponding to the maximal positive $\delta_{i,\varepsilon}$ to the set S. Thus at each time the set S contains at most 1 pair and no backtracking.

Algorithm 2 *(Best Descend)*

E. Choose the greatest positive $\delta_{i,\varepsilon} > 0$ (if exists) and add $(\overline{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$ into S.

The next version of the length based attack is so called *generalized LBA*. This is an LBA with backtracking in which we extend the set of elements in Bob's (respectively Alice's) public sets. It was conjectured in [9] that generalized length based attack can break the multiple conjugacy search problem for any parameter values. We need to mention here that one has to be cautious about the choice of the new elements as the complexity of each iteration of LBA depends on the number of elements in the public set $\bar{a}$.

Algorithm 3 *(Generalized LBA)*

A. Extend $\bar{a}$ with products $a_{i_1}^{\varepsilon_1} \ldots a_{i_j}^{\varepsilon_j}$ where j is limited by some constant.
B. Run Algorithms 1 or 2 with the obtained tuple $\bar{a}$ and tuples $\overline{b}$, $\overline{b}'$.

Algorithms 1-3 always halt because only tuples of total lengths smaller than the lengths of the public sets are considered. Note that all of the algorithms above are heuristic in their nature and may halt without producing the solution.

2.3 Peaks

In this section we define the notion of a peak and show that condition (2) on the length function in the platform group B_n is not enough for the success of LBA. We give examples of instances of AAG invulnerable to the length based attacks 2 and 1.

Example 1. (*Hard instance*) Consider B_{80} and two braids

$$a_1 = x_{39}^{-1} x_{12} x_7 x_3^{-1} x_1^{-1} x_{70} x_{25} x_{24}^{-1}$$

and

$$a_2 = x_{42} x_{56}^{-1} x_8 x_{18}^{-1} x_{19} x_{73} x_{33}^{-1} x_{22}^{-1}$$

which we think of as elements from Alice's public set. It is easy to check that

$$a_1^{-1} a_2^{-1} a_1 = x_7^{-1} \cdot a_2^{-1} \cdot x_7 = x_7^{-1} x_{22} x_{33} x_{73}^{-1} x_{19}^{-1} x_{18} x_8^{-1} x_{56} x_{42}^{-1} x_7$$

and

$$a_1^{-1} a_2^{-1} a_1 a_2 = x_7 x_8^{-1}.$$

Hence $|a_1| = 8$, $|a_1^{-1} a_2^{-1}| = 16$, $|a_1^{-1} a_2^{-1} a_1| = 10$, and $|a_1^{-1} a_2^{-1} a_1 a_2| = 2$. Now let $\overline{b} = (b_1, \ldots, b_N)$ be a random tuple of braids thought of as Bob's public set. As we saw, for the majority of braids conjugation increases the length by almost twice the length of a conjugator. Hence, for generic tuple $\overline{b}$ the following length growth would be expected:

$$\begin{array}{rcl} & \overline{b} & \\ & \downarrow & \\ |a_{s_1}^{-\varepsilon_1} & \overline{b} & a_{s_1}^{\varepsilon_1}| \quad \approx |\overline{b}| + 8N \\ & \downarrow & \\ |a_{s_2}^{-\varepsilon_2} a_{s_1}^{-\varepsilon_1} & \overline{b} & a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2}| \quad \approx |\overline{b}| + 16N \\ & \downarrow & \\ |a_{s_3}^{-\varepsilon_3} a_{s_2}^{-\varepsilon_2} a_{s_1}^{-\varepsilon_1} & \overline{b} & a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} a_{s_3}^{\varepsilon_3}| \quad \approx |\overline{b}| + 10N \\ & \downarrow & \\ |a_{s_4}^{-\varepsilon_4} a_{s_3}^{-\varepsilon_3} a_{s_2}^{-\varepsilon_2} a_{s_1}^{-\varepsilon_1} & \overline{b} & a_{s_1}^{\varepsilon_1} a_{s_2}^{\varepsilon_2} a_{s_3}^{\varepsilon_3} a_{s_4}^{\varepsilon_4}| \approx |\overline{b}| + 2N \end{array} \tag{3}$$

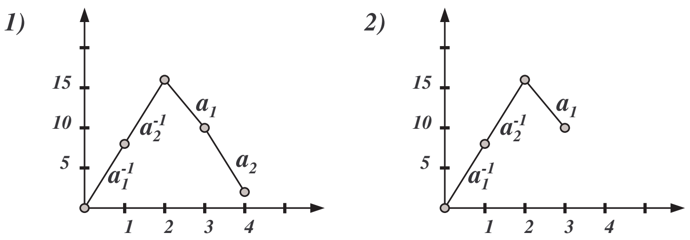

Fig. 2. 1) Commutator-type 4-peak $[a_1, a_2]$ from Example 1. 2) Conjugator-type 2-peak as in Example 1 for $a_1^{-1}a_2^{-1}a_1$.

Clearly, the length based attacks 2 and 1 fail for such element A because to guess the first correct conjugator it is required to increase the length of the tuple substantially (from $|\overline{b}| + 2N$ to $|\overline{b}| + 10N$).

The reason for the attack failure in the previous example is that Alice's private key $[a_1, a_2]$ forms a peak (*commutator-type peak*):

Definition 1. *(*Peak*) Let $G = \langle X; R\rangle$, l_G a length function on G, and $H = \langle w_1, \ldots, w_k\rangle$. We say that a word $w = w_{i_1} \ldots w_{i_n}$ is an n-*peak *in H relative to l_G if there is no $1 \le j \le n-1$ such that*

$$l_G(w_{i_1} \ldots w_{i_n}) \ge l_G(w_{i_1} \ldots w_{i_j}).$$

*We say that $w = w_{i_1} \ldots w_{i_n}$ is m-*hard *if there exist $s \in \{1, \ldots, n\}$ such that for each $j = 1, \ldots, k$*

$$l_G(w_{i_1} \ldots w_{i_{s+k-1}}) \ge l_G(w_{i_1} \ldots w_{i_{s+k-j}})$$

and m is maximal with such property.

Note that according to the definition of m-hardness each product $w_{i_1} \ldots w_{i_n}$ is at least 1-hard. To see the hardness of the word $w = w_{i_1} \ldots w_{i_n} \in H$ (given as a product of generators of H) it is often convenient to depict the function $k \to l_G(w_{i_1} \ldots w_{i_k})$ for $k = 0, \ldots, n$. See Figure 2 for the words from Example 1. The graphs explain the choice of term peak. On the other hand given $w \in H$ we do not know any way to compute its hardness other than to compute the decomposition of w in a product of generators, which is a very hard problem for some subgroups of a braid group.

After making lots of experiments we strongly believe that the computational hardness of SR-SCSP in braid groups is not an intrinsic property of conjugation, but comes from the structure of the corresponding subgroup. To defend against LBA it is necessary to choose a public set and m-hard private keys, where m is large compared to N_1, N_2. One can generate such keys using the Mihailova construction [12].

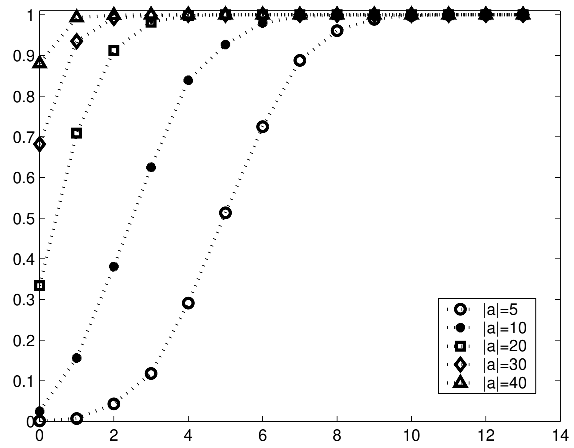

Fig. 3. Distribution of the number of peaks in private keys.

However, generating keys that are immune just to LBA is not sufficient for the security of the protocol. A generating procedure which provides keys secure against all known attacks is a difficult task and is a current research objective.

2.4 Peaks in randomly chosen private keys

Even though it is not hard to construct instances invulnerable to LBA, such instances are quite rare and it is very unlikely to generate one uniformly for certain parameter values. Figure 3 shows the distribution of the number of peaks in private keys for B_{80} and $L_1 = 5, 10, 20, 30, 40$, $L_2 = L_1 + 3$. Figure 4 shows the distribution of the maximal size of peaks in private keys for B_{80} and $L_1 = 5, 10, 20, 30, 40$, $L_2 = L_1 + 3$. The distributions shown in Figures 3 and 4 are obtained experimentally using the approximation of the geodesic length.

According to Figures 3 and 4 the greater the length of the generators the shorter and rarer the peaks are. Intuitively, we can distinguish 3 types of distribution of peaks depending on the parameter L_1 (for B_{80}):

1) *Short generators ($L_1 \in [5, 20]$).* A random private key contains several peaks, one or two of which are relatively long. The probability of a success of Algorithm 1 in this case is very low. To make Algorithm 3 work it requires extending the basis with a lot of elements, which suggests using subgroup attack. Note that this case is in the ballpark of the parameters suggested in [1]. LBA fails in this case.
2) *Long generators ($L_1 > 40$).* With probability 90% random private key contains no peaks. The LBA is expected to work smoothly.
3) *Middle sized generators ($L_1 \in [20, 40]$).* A random private key with probability 90% contains at most two short peaks. Experiments showed that almost all peaks are conjugator-type peaks $a_i^\varepsilon a_j^\delta a_i^{-\varepsilon}$ (for some indices i, j and powers $\varepsilon, \delta = \pm 1$). Also there are a few commutator-type peaks $a_i^\varepsilon a_j^\delta a_i^{-\varepsilon} a_j^{-\delta}$.

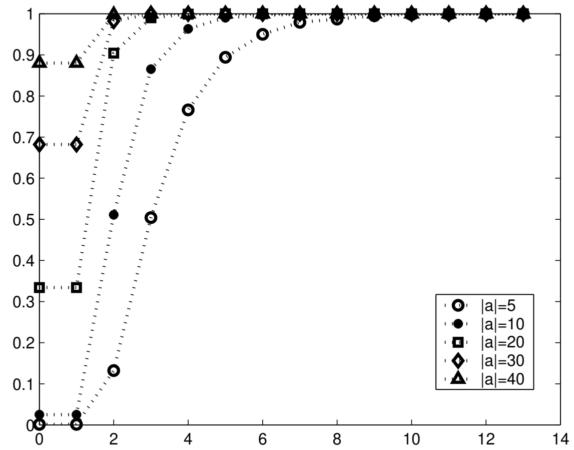

Fig. 4. Distribution of the size of maximal peaks in private keys.

In other ranks experiments show similar behavior with different interval values of L_1.

3 The Attack

Based on our observations from Section 2.4 on the structure of peaks we introduce a modification of the generalized length based attack which breaks the instances of AAG with middle to high lengths of generators.

The main idea behind the generalized LBA is to add elements from the corresponding subgroup to "cut" the peaks inside the private key as in the following example. Consider Alice's public tuple (a_1, a_2) from Example 1 and choose her private key to be $a_1^{-1}a_2^{-1}a_1$. Extending (a_1, a_2) with the product $a_2^{-1}a_1$ cuts the peak in Figure 2.(2) making the descend possible. Obviously any peak in the private key A can be cut by extending the tuple $\bar{a}$ with all the products of the length up to the length of the decomposition L. However, this is equivalent to breaking the system by the brute force approach. The number of such products depends exponentially on the product length L with respect to the rank of braid group. With the parameters considered in this paper the number of all such products is of order 20^{50}. Our goal is to introduce a relatively small set of short products which will eliminate most of the frequently occurring peaks.

As we discussed in Section 2.4 most of the peaks in a randomly generated word are of lengths 2 and 3, and most of them are of conjugator-type. Indeed, the expected number of conjugators $\mathrm{E}[C_L]$, given that the factors are sampled uniformly and independently, is estimated about $1/2N_2(L-2)$. For values $L = 50$ and $N_2 = 20$ we have $\mathrm{E}[C_{50}] \approx 1.2$, i.e. a conjugator is expected to occur at least once. It is also easy to see that the probability of a long peak to occur in a uniform random word is very small.

Hence, it is a natural idea to extend $\bar{a}$ with all conjugators and commutators of its elements (observe that this quadratically increases the size of the tuple). In general the decision of extending the input tuple with a set of products is based on the balancing of the tradeoff between the frequency of occurrences of corresponding peaks and the increase of complexity on each iteration. In our implementation we choose to add only conjugators as they seem to be inevitable, whereas commutators as well as other types of longer peaks are very rare in the key generated uniformly randomly.

3.1 Most significant generator heuristic

Adding all products of subgroup generators up to a certain length increases the size of a generating set by a polynomial with respect to the subgroup rank (N_1 or N_2). Although theoretically feasible, this introduces practical problems even in the case of small ranks. The following experimental observation can be used as a heuristic which helps to reduce the number of operations on each iteration.

Let δ_{k,ε_k} be the maximal length reduction obtained during an iteration I (see step D of Algorithm 1):

$$\delta_{k,\varepsilon_k} = \max\{\delta_{i,\varepsilon} \mid i = 1, \ldots, N_1\}.$$

The corresponding generator $a_k^{\varepsilon_k}$ is called the *most significant generator* of the iteration I. According to our experiments, the most significant generators almost always are either the correct generators, or are contained in corresponding peaks. The simple heuristic suggests to vary the tuple $\bar{a}$ on each iteration and extend it with elements which are the products containing the current most significant generator. In this case the number of operations performed during one iteration is still linear with respect to the subgroup rank N_1.

3.2 Algorithms

Based on the heuristics given above we introduce two new attacks on AAG protocol. Both procedures have the same input and output as described in Section 2.2.

The first attack is a relatively straightforward implementation of the generalized length based attack where the set of generators $\bar{a}$ is extended by adding all conjugations of the original generators.

Algorithm 4 *(Generalized LBA with conjugation)*

A. Extend $\bar{a}$ with all conjugators: $\bar{a} = \bar{a} \cup \{x_i x_j x_i^{-1} \mid x_i, x_j \in \bar{a}^{\pm 1}, i \neq j\}$.
B. Run Algorithm 1 with the obtained tuple.

The second attack uses the dynamic extension set based on the products containing the most significant generator. These products include conjugators and products of two generators from $\bar{a}$. It is possible that none of the generators a_i cause length reduction on the step D of the LBA procedure 1. In such situation we introduce all conjugators and two generator products, hoping to either cut a peak or reduce the length function approximation error.

Algorithm 5 *(LBA with dynamic set)*

A. *Initialize a set* $S = \{(\overline{b}', e)\}$, *where* e *is the identity of* B_n.
B. *If* $S = \emptyset$ *then output* $FAIL$.
C. *Choose a pair* $(\overline{c}, x) \in S$ *with a minimal* $|\overline{c}|$. *Remove* $(\overline{c}, x)$ *from* S.
D. *For each* $i = 1, \ldots, N_1$, $\varepsilon = \pm 1$ *compute* $\delta_{i,\varepsilon} = |\overline{c}| - |\overline{c}^{a_i^{\varepsilon}}|$.
E. *If* $\forall i$ $\delta_{i,\varepsilon} \leq 0$ *then define* $\bar{a}_{ext} = \bar{a} \cup \{x_i x_j x_i^{-1}, x_i x_j, x_i^2 \mid x_i, x_j \in \bar{a}^{\pm 1}, i \neq j\}$
F. *Else define* $\bar{a}_{ext} = \bar{a} \cup \{x_j x_m x_j^{-1}, x_m x_j, x_j x_m, x_m^2 \mid x_j \in \bar{a}^{\pm 1}, m \neq j\}$, *where* x_m *s.t.* $\delta_m = \max\{\delta_{i,\varepsilon} \mid i = 1, \ldots, N_1\}$.
G. *For all* $w \in \bar{a}_{ext}$ *compute* $\delta_w = |\overline{c}| - |\overline{c}^w|$, *if* $\delta_w > 0$ *add* $(\bar{c}, xw)$ *to* S.
H. *If* $\overline{c}^w = \overline{b}$ *them output* xw.
I. *Otherwise goto B.*

3.3 Experiments

We performed a series of experiments to test the heuristic approaches described in the previous sections. The following parameters were chosen: $B_n = B_{80}$, $N_1 = N_2 = 20$, $L = 50$ and parameters L_1, L_2 were varied to demonstrate the better success rate of the length based attack for instances with longer subgroup generators. There were 100 problems generated for each set of parameters.

L_1, L_2	10,13	20,23	30,33	40,43
Algorithm 2	00	05	45	60
Algorithm 4	00	51	80	64
Algorithm 5	00	30	97	96

Table 1. Success rate of the length based attack (%).

The attack was considered unsuccessful if an algorithm stopped and produced $FAIL$ or it has not terminated after 24 hours of execution. Experiments were performed on Dual 1 GHz Pentium III processors with 2GB of RAM.

The percentages of successful attacks are given in Table 1. According to the experiments Algorithms 4 and 5 almost never produce $FAIL$ indicating that the success rate could be improved by using more powerful computing or extending the termination time.

As expected, none of the attacks were successful on instances with short generators. However, keys obtained from long generators in many cases can be reconstructed successfully even using the naive best descend procedure (see Algorithm 2). The heuristics described in Section 3.1 seem to work well in cutting peaks contained in uniformly randomly generated keys, showing over 50% success rate even for instances with middle length generators.

References

1. I. Anshel, M. Anshel, D. Goldfeld, *An algebraic method for public-key cryptography*, Math. Res. Lett. **6** (1999), 287–291.
2. J. S. Birman, *Braids, links and mapping class groups*, Ann. Math. Studies **82**, Princeton Univ. Press, 1974.
3. CRyptography And Groups (CRAG), C++ and Python Library for computations in groups and group based cryptography, available at http://www.acc.stevens.edu/downloads.php.
4. P.Dehornoy, *A fast method for comparing braids*, Advances in 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, "Length-based conjugacy search in the Braid group", http://arxiv.org/abs/math.GR/0209267.
7. D. Hofheinz, R. Steinwandt. *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, Y.G. Desmedt, ed., vol. 2567 of Lecture Notes in Computer Science, pp. 187-198, Springer, 2002.
8. J. Hughes, "A Linear Algebraic Attack on the AAFG1 Braid Group Cryptosystem", ACISP 2002, Lecture Notes in Computer Science, vol. 2384, (2002), 176–189.
9. J. Hughes, A. Tannenbaum, *Length-based attacks for certain group based encryption rewriting systems.* In: Workshop SECI02 Securitè de la Communication sur Intenet, September 2002, Tunis, Tunisia.
10. 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.
11. 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.
12. K. A. Mihailova, "The occurrence problem for free products of groups", Math USSR-Sbornik 70, (1966), 241-251.
13. A. G. Myasnikov, V. Shpilrain, A. Ushakov. *A practical attack on some braid group based cryptographic protocols.* In CRYPTO 2005, Lecture Notes Comp. Sc. 3621 (2005), 86-96.
14. A. G. Myasnikov, V. Shpilrain, A. Ushakov. *Random subgroups of braid groups: an approach to cryptanalysis of a braid group based cryptographic protocol.* In PKC 2006, Lecture Notes Comp. Sc. 3958 (2006), 302-314.

15. M. Paterson, A. Razborov, *The set of minimal braids in co-NP-complete* J. Algorithms, **12** (1991), 393-408.
16. 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/
17. A. Vershik, S. Nechaev, R. Bikbov. *Statistical properties of braid groups in locally free approximation*. In Communications in Mathematical Physics, vol. 212, 2000, pp. 469–501.