Article:

Kahrobaei, Delaram, Garber, David and Lam, Ha (2015) Length-based attacks in polycyclic groups. Groups Complexity Cryptology. pp. 33-43. ISSN: 1869-6104

https://doi.org/10.1515/jmc-2014-0003

Research Article

David Garber, Delaram Kahrobaei and Ha T. Lam

Length-based attacks in polycyclic groups

Abstract: The Anshel–Anshel–Goldfeld (AAG) key-exchange protocol was implemented and studied with the braid groups as its underlying platform. The length-based attack, introduced by Hughes and Tannenbaum, has been used to cryptanalyze the AAG protocol in this setting. Eick and Kahrobaei suggest to use the polycyclic groups as a possible platform for the AAG protocol. In this paper, we apply several known variants of the length-based attack against the AAG protocol with the polycyclic group as the underlying platform. The experimental results show that, in these groups, the implemented variants of the length-based attack are unsuccessful in the case of polycyclic groups having high Hirsch length. This suggests that the length-based attack is insufficient to cryptanalyze the AAG protocol when implemented over this type of polycyclic groups. This implies that polycyclic groups could be a potential platform for some cryptosystems based on conjugacy search problem, such as non-commutative Diffie–Hellman, El Gamal and Cramer–Shoup key-exchange protocols. Moreover, we compare *for the first time* the success rates of the different variants of the length-based attack. These experiments show that, in these groups, the memory length-based attack introduced by Garber, Kaplan, Teicher, Tsaban and Vishne does better than the other variants proposed thus far in this context.

Keywords: Polycyclic groups, non-commutative public key, length-based attack

MSC 2010: 68-XX, 20-XX

David Garber: Department of Applied Mathematics, Faculty of Sciences, Holon Institute of Technology, 52 Golomb st., PO Box 305, 58102 Holon, Israel, e-mail: garber@hit.ac.il
Delaram Kahrobaei: PhD Program in Computer Science, CUNY Graduate Center, City University of New York, 365 Fifth Ave, New York, NY 10016; and Mathematics Department, New York City College of Technology, 300 Jay Street, Brooklyn, NY 11201, USA, e-mail: dkahrobaei@gc.cuny.edu
Ha T. Lam: Department of Mathematics, CUNY Graduate Center, City University of New York, 365 Fifth Ave, New York, NY 10016, USA, e-mail: hlam@gc.cuny.edu

Communicated by: Spyros Magliveras

1 Introduction

The Anshel–Anshel–Goldfeld (AAG) key-exchange protocol was introduced in 1999 [1]. Following its introduction, the AAG protocol was extensively studied using different groups as its underlying platform. Ko et al. [15] used braid groups. Moreover, Myasnikov and Ushakov [18] studied the security of the AAG protocol with respect to several attacks on any platform groups satisfying some theoretic properties (exponentially generic free basis property).

Hughes and Tannenbaum [11] introduced the length-based attack (LBA) on the AAG protocol with its implementation in braid groups. They emphasized the importance of choosing the correct length function. Later, Garber et al. [6] gave several realizations of this approach, particularly a length function for the braid group and experimental results suggesting that the attack fails for the parameters suggested in existing protocols. However, Garber et al. [5] also suggested an extension of the length-based attack which uses memory which succeeded in cryptanalyzing the AAG protocol. A similar attack was implemented against a system based on the Thompson group [19]. Most recently, Myasnikov and Ushakov [17] analyzed the reasons behind the failure of the previous implementations of the LBA, such as the occurrence of *commutator-type peaks*, and gave an experimental evidence that the LBA can be modified to cryptanalyze the AAG protocol with high success rate. However, this work is again done the braid groups as the underlying platform.

Eick and Kahrobaei [3] have suggested a different platform for the AAG protocol – the *polycyclic group*. In polycyclic groups, the word problem can be solved efficiently [7], but known solutions to the conjugacy problem are much less efficient. Using experimental results, Eick and Kahrobaei showed that while the conjugacy problem can be solved within seconds using polycyclic groups with small Hirsch length, the conjugacy problem in polycyclic groups with high Hirsch length requires a much longer time for its solution.

Taking inspiration from this result, we investigate the success rate of the length-based attack on the AAG protocol, where the underlying platform is the polycyclic groups, especially those with high Hirsch length. Toward this end, we first construct polycyclic groups of high Hirsch length using a method introduced by Holt et al. [10]. Then, we implement the different variants of the LBA presented in [5, 6, 17]. The experimental results that we collect suggest that the LBA is insufficient to cryptanalyze the AAG protocol, when we use the polycyclic groups with high enough Hirsch length as the underlying platform. Consequently, the polycyclic group is the *first* underlying platform which the LBA is insufficient for cryptanalyzing the AAG protocol on this platform, whereas the solution for the word problem is quite efficient. A suggestion for concrete parameters appears in the last section.

Moreover, we compare *for the first time* on any platform the success rates of the different variants of the LBA.

As a wider application, we note that the conjugacy search problem is the basis for various cryptographic protocols besides AAG, such as the non-commutative Diffie–Hellman key-exchange [15], the non-commutative El Gamal key-exchange [12], the non-abelian Cramer–Shoup key-exchange [2] and the non-commutative digital signatures [13]. The LBA can be applied to all these protocols; therefore, a platform group which experimental results show that the LBA is insufficient for cryptanalyzing the AAG protocol over this platform, such as polycyclic groups, can help instantiate them.

The paper is organized as follows. In Section 2, we introduce the Anshel–Anshel–Goldfeld key-exchange protocol. In Section 3, we give a short review of polycyclic groups and the construction that we have used. In Section 4, we review the length-based attack, and in Section 5, we present the experiments, their results and corresponding conclusions.

2 The Anshel–Anshel–Goldfeld key-exchange protocol

Following [17], we present here the Anshel–Anshel–Goldfeld key-exchange protocol (for more details, see [1]). As usual, we use two entities, called Alice and Bob, for presenting the two parties which plan to communicate over an insecure channel.

Let G be a group with generators $g_1, \dots, g_n$. First, Alice chooses her public set $\overline{a} = (a_1, \dots, a_{N_1})$, where $a_i \in G$, and Bob chooses his public set $\overline{b} = (b_1, \dots, b_{N_2})$, where $b_i \in G$. They both publish their sets. Alice then chooses her private key $A = a_{s_1}^{\varepsilon_1} \cdots a_{s_L}^{\varepsilon_L}$, where $a_{s_i} \in \overline{a}$ and $\varepsilon_i \in \{\pm 1\}$. Bob also chooses his private key $B = b_{t_1}^{\delta_1} \cdots b_{t_L}^{\delta_L}$, where $b_{t_i} \in \overline{b}$ and $\delta_i \in \{\pm 1\}$. Alice computes $b_i' = A^{-1} b_i A$ for all $b_i \in \overline{b}$ and sends it to Bob. Bob also computes $a_i' = B^{-1} a_i B$ for all $a_i \in \overline{a}$ and sends it to Alice. Now, the shared secret key is $K = A^{-1}B^{-1}AB$. Alice computes this key by

$$\begin{aligned} K_A &= A^{-1}(a_{s_1}'^{\varepsilon_1} \cdots a_{s_L}'^{\varepsilon_L}) \\ &= A^{-1}(B^{-1}a_{s_1}B)^{\varepsilon_1} \cdots (B^{-1}a_{s_L}B)^{\varepsilon_L} \\ &= A^{-1}B^{-1}(a_{s_1}^{\varepsilon_1} \cdots a_{s_L}^{\varepsilon_L})B \\ &= A^{-1}B^{-1}AB = K. \end{aligned}$$

Similarly, Bob can compute $K_B = B^{-1}(b_{t_1}'^{\delta_1} \cdots b_{t_L}'^{\delta_L}) = B^{-1}A^{-1}BA$, and then he knows the shared secret key by $K = K_B^{-1}$.

In order to find K, it is enough for the eavesdropper either to find $A' \in \langle a_1, \dots, a_{N_1} \rangle$ such that $\overline{b'} = A'^{-1}\overline{b}A'$ or to find $B' \in \langle b_1, \dots, b_{N_2} \rangle$ such that $\overline{a'} = B'^{-1}\overline{a}B'$ (an incompatible sufficient condition can be found in [14]).

Thus, the security of the AAG protocol is based on the assumption that the subgroup-restricted simultaneous conjugacy search problem is hard.

3 Polycyclic groups

In this section, we give a short review for polycyclic groups and describe the construction of polycyclic groups of high Hirsch length. For more details, see [10].

3.1 The polycyclic presentation

Recall that G is a *polycyclic group* if it has a polycyclic series, i.e., a subnormal series

$$G = G_1 \rhd G_2 \rhd \cdots \rhd G_{n+1} = \{1\},$$

with non-trivial cyclic factors. The *polycyclic generating sequence* of G is the n-tuple $(g_1, \ldots, g_n)$, such that $G_i = \langle g_i, G_{i+1} \rangle$ for $1 \leq i \leq n$.

Any polycyclic group has a finite presentation of the form

$$\langle g_1, \ldots, g_n \mid g_j^{g_i} = w_{ij},\ g_j^{g_i^{-1}} = v_{ij},\ g_k^{r_k} = u_{kk} \text{ for } 1 \leq i < j \leq n \text{ and } k \in I \rangle,$$

where w_{ij}, v_{ij}, u_{kk} are words in the generators $g_{i+1}, \ldots, g_n$ and I is the set of indices $i \in \{1, \ldots, n\}$ such that $r_i = [G_i : G_{i+1}]$ is finite. Here a^b stands for $b^{-1}ab$.

It is known by induction that each element of G defined by this presentation can be uniquely written as $g = g_1^{e_1} \cdots g_n^{e_n}$ where $e_i \in \mathbb{Z}$ for $1 \leq i \leq n$, and $0 \leq e_i < r_i$ for $i \in I$. We call $g = g_1^{e_1} \cdots g_n^{e_n}$ the *normal form* of an element in G. If every element in the group can be uniquely presented in the normal form, then the polycyclic presentation is called *consistent*. Note that every polycyclic group has a consistent polycyclic presentation (see [10]).

The *Hirsch length* of a polycyclic group is the number of indices i such that $r_i = [G_i : G_{i+1}]$ is infinite. This number is invariant of the chosen polycyclic sequence.

3.2 Constructing polycyclic groups using number fields

There are several ways for constructing polycyclic groups. For the purpose of this paper, we construct polycyclic groups by semidirect products of the maximal order and the unit group of a number field. This construction follows [10].

Let $f(x) \in \mathbb{Z}[x]$ be an irreducible polynomial. The polynomial f defines a field extension F over $\mathbb{Q}$. The *maximal order* or the *ring of integers* O_F of the number field F is the set of algebraic integers in F,

$$O_F = \{a \in F \mid \text{there exists a monic polynomial } f_a(x) \in \mathbb{Z}[x] \text{ such that } f_a(a) = 0\}.$$

The *unit group* of F is

$$U_F = \{a \in O_F \mid a \neq 0 \text{ and } a^{-1} \in O_F\}.$$

For constructing the polycyclic group by the maximal order and the unit group of a number field F where $[F : \mathbb{Q}] = n$, we recall two results. First, the maximal order O_F forms a ring whose additive group is isomorphic to $\mathbb{Z}^n$ (see [20]). Second, Dirichlet's unit theorem states that given $n = s + 2t$, where s and $2t$ are the numbers of real field monomorphisms $F \to \mathbb{R}$ and complex field monomorphisms $F \to \mathbb{C}$ respectively, then the unit group U_F is a finitely-generated abelian group of the form $U_F \cong \mathbb{Z}^{s+t-1} \times \mathbb{Z}_m$ for some even m (see [20]). Here, we use the fact that the unit group is a finitely-generated abelian group and hence U_F is also polycyclic.

Let G be a group and $N \trianglelefteq G$, it is easy to see that if N and G/N are both polycyclic, then the group G is also polycyclic by putting together the polycyclic series of N and the series induced by the polycyclic series of G/N. Since the above results guaranteed that the maximal order is a polycyclic group and the unit group, which is isomorphic to G/O_F, is also polycyclic, the group $G = O_F \rtimes U_F$ is polycyclic. The action which defines the semidirect product is a multiplication from the right of U_F on O_F.

If $N \trianglelefteq G$, the Hirsch length of a polycyclic group G is $h(G) = h(N) + h(G/N)$; in our case,

$$h(G) = h(O_F) + h(U_F),$$

where $h(O_F)$ is n, which is the degree of the generating polynomial f. Hence, for constructing a polycyclic group of high Hirsch length, we have to find an irreducible polynomial of high enough degree, and then the polycyclic group constructed by the above method will have Hirsch length larger than the degree of the polynomial.

3.3 Polycyclic groups as platform groups for the AAG protocol

Polycyclic groups are suitable as platform groups for the AAG protocol for several reasons. First, the word problem can be solved efficiently using the collection algorithm [7], see also [3]. Second, the conjugacy search problem has no efficient solution in general polycyclic groups. This assessment is due to Eick and Kahrobaei [3], using the following experiment: Let $K = \mathbb{Q}[x]/(f_w)$ be an algebraic number field for a cyclotomic polynomial f_w, where w is a primitive r-th root of unity. Let $G(w) = O \rtimes U$, where O is the maximal order and U the unit group of K, r the order of w and $h(G(w))$ the Hirsch length. The average time used for 100 applications of the collection algorithm on random words and the average time used for 100 applications of the conjugacy algorithm on random conjugates is

r	$h(G(w))$	Collection	Conjugation
3	2	0.00 seconds	9.96 seconds
4	2	0.00 seconds	9.37 seconds
7	6	0.01 seconds	10.16 seconds
11	14	0.05 seconds	> 100 hours

We can see that the collection algorithm works very fast even for polycyclic groups of high Hirsch length, and therefore the word problem has an efficient solution. On the other hand, the solution to the conjugacy problem is not efficient for polycyclic groups having high Hirsch length.

4 The length-based attack

The length-based attack (LBA) is a probabilistic attack against the conjugacy search problem in general, and against the AAG protocol in particular, with the goal of finding Alice's (or Bob's) private key. It is based on the idea that a conjugation of the correct element should decrease the length of the captured package. Using the notations of Section 2, the captured package is $\overline{b'} = (b'_1, \dots, b'_{N_2})$, where $b'_i = A^{-1} b_i A$. If we conjugate $\overline{b'}$ with elements from the group $\langle a_1, \dots, a_{N_1} \rangle$ and the length of the resulting tuple has been decreased, then we have found a candidate for the conjugating factor. The process of conjugation is then repeated with the decreased-length tuple until a longer candidate for the conjugating factor is found. The process ends when the conjugated captured package is the same as $\overline{b} = (b_1, \dots, b_{N_2})$, which is known. Then, the conjugate can be recovered by reversing the sequence of conjugating factors. For more details on the LBA, see [5, 6, 9, 16, 17].

4.1 Variants of the LBA

In [5, 6, 17, 19], several variants of the LBA are presented. Here, we give four variants of the LBA that we implemented against the AAG protocol having the polycyclic group as its underlying platform. In all these variants, the following input and output are expected:

- **Input**: $\overline{a} = (a_1, \dots, a_{N_1})$, $\overline{b} = (b_1, \dots, b_{N_2})$ and $\overline{b'} = (b'_1, \dots, b'_{N_2})$, such that $b'_i = b_i^A$ for $i = 1, \dots, N_2$.
- **Output**: An element $A' \in \langle a_1, \dots, a_{N_1} \rangle$ such that $b'_i = b_i^{A'}$ for $i = 1, \dots, N_2$, or FAIL if the algorithm cannot find such A'.

We will use the following notation: if $\overline{c} = (c_1, \dots, c_k)$, then its *total length* $|\overline{c}|$ is $\sum_{i=1}^{k} |c_i|$ (the length of c_i, $|c_i|$, will be discussed in Section 4.2).

4.1.1 LBA with backtracking

The most straight-forward variant of LBA (see Algorithm 1) conjugates $\overline{b'}$ directly with $a_i^{\pm 1} \in \{a_1, \dots, a_{N_1}\}$. This is termed “LBA with backtracking” by Myasnikov and Ushakov [17].

4.1.2 LBA with a dynamic set

Through analysis, Myasnikov and Ushakov [17] concluded that different types of peaks make LBA unsuccessful. To overcome this, they suggested a new version of the algorithm, which they termed “LBA with a dynamic set”. Here (see Algorithm 2), if a generator a_i causes a length reduction, only the conjugates and products involving a_i are added to the dynamic set. On the other hand, if no generator causes a length reduction, all conjugates and two generators products are added. Their experimental results suggest that this algorithm works especially well in the case of keys composed from long generators, but it is not worse than the naive algorithm in the other cases. The algorithm presented here is a modified version of their algorithm, which we implemented to attack the AAG protocol having the polycyclic group as its underlying platform.

4.1.3 Memory-LBA

Another variant, presented in [5], is also considered. In this variant (see Algorithm 3), we allocate an array S of a fixed size M. The array S holds M tuples every round. In every round, all elements of S are conjugated, but only the M smallest conjugated tuples (with respect to their length) are inserted back into S. For the halting condition, we use a predefined time-out.

4.1.4 LBA* (with memory)

We present a different variant of memory-LBA which is again based on a fixed-size array allocated for the algorithm. Here (see Algorithm 4), S holds M tuples every round and is sorted by the first element (with respect to the length of conjugated element) of each tuple. In every round, only the smallest element of S is removed and conjugated by all the generators and their inverses. The conjugated tuples are inserted back into S depending on whether there is a free place in S. If there is no more places in S, and the conjugated tuple is smaller than the largest element in S, swap them and re-sort S. Since S is always kept sorted, any operation to find the “smallest element” costs constant time. As in the previous variant, we use a predefined time-out as the halting condition.

Algorithm 1. LBA with backtracking

```
Initialize $S = \{(\overline{b'}, \mathrm{id}_G)\}$.
while $S \neq \emptyset$ do
    Choose $(\bar{c}, x) \in S$ such that $|\bar{c}|$ is minimal. Remove $(\bar{c}, x)$.
    for $i = 1, \dots, N_1$ and $\varepsilon = \pm 1$ do
        Compute $\delta_{i,\varepsilon} = |\bar{c}| - |\bar{c}^{a_i^\varepsilon}|$.
        if $\bar{c}^{a_i^\varepsilon} = \bar{b}$ then output inverse of $xa_i^\varepsilon$ and stop.
        if $\delta_{i,\varepsilon} > 0$ then                                ▷ length has been decreased
            Add $(\bar{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$ to $S$.
        end if
    end for
end while
Otherwise, output FAIL.                                   ▷ no more elements to conjugate
```

Algorithm 2. LBA with a dynamic set

```
Initialize $S = \{(\overline{b'}, \mathrm{id}_G)\}$.
while $S \neq \emptyset$ do
    Choose $(\bar{c}, x) \in S$ such that $|\bar{c}|$ is minimal. Remove $(\bar{c}, x)$.
    for $i = 1, \dots, N_1$ and $\varepsilon = \pm 1$ do
        Compute $\delta_{i,\varepsilon} = |\bar{c}| - |\bar{c}^{a_i^\varepsilon}|$
    end for
    if $\delta_{i,\varepsilon} \leq 0$ for all $i$ then
        Define $\bar{a}_{\mathrm{ext}} = \bar{a} \cup \{x_i x_j x_i^{-1}, x_i x_j, x_i^2 \mid x_i, x_j \in \bar{a}^{\pm 1}, i \neq j\}$.
    else Define $\bar{a}_{\mathrm{ext}} = \bar{a} \cup \{x_j x_m x_j^{-1}, x_m x_j, x_j x_m, x_m^2 \mid x_j \in \bar{a}^{\pm 1}, m \neq j\}$
        where $x_m$ is such that $\delta_m = \max\{\delta_{i,\varepsilon} \mid i = 1, \dots, N_1\}$.
    end if
    for all $w \in \bar{a}_{\mathrm{ext}}$ do
        Compute $\delta_w = |\bar{c}| - |\bar{c}^w|$.
    end for
    if $\bar{c}^w = \bar{b}$ then output inverse of $xw$ and stop.
    if $\delta_w > 0$ then                                          ▷ length has been decreased
        Add $(\bar{c}^w, xw)$ to $S$.
    end if
end while
Otherwise, output FAIL.                                   ▷ no more elements to conjugate
```

Algorithm 3. Memory-LBA

```
Initialize $S = \{(|\overline{b'}|, \overline{b'}, \mathrm{id}_G)\}$.
while not time-out do
    for $(|\bar{c}|, \bar{c}, x) \in S$ do
        Remove $(|\bar{c}|, \bar{c}, x)$ from $S$.
        Compute $\bar{c}^{a_i^\varepsilon}$ for all $i \in \{1 \dots N_1\}$ and $\varepsilon \in \{\pm 1\}$.
        if $\bar{c}^{a_i^\varepsilon} = \bar{b}$ then output inverse of $xa_i^\varepsilon$ and stop.
        Save $(|\bar{c}^{a_i^\varepsilon}|, \bar{c}^{a_i^\varepsilon}, xa_i^\varepsilon)$ in $S'$.
    end for
    After finished all conjugations, sort $S'$ by the first element of every tuple.
    Copy the smallest $M$ elements into $S$ and delete the rest of $S'$.
end while
Otherwise, output FAIL.
```

Algorithm 4. LBA* (with memory)

```
Initialize $S = \{(|\overline{b'}|, \overline{b'}, \mathrm{id}_G)\}$.
while not time-out do
    Choose $(|\bar{c}|, \bar{c}, x) \in S$ such that $|\bar{c}|$ is minimal. Remove $(|\bar{c}|, \bar{c}, x)$.
    for $i = 1, \dots, N_1$ and $\varepsilon = \pm 1$ do
        Compute $\bar{c}^{a_i^\varepsilon}$.
        if $\bar{c}^{a_i^\varepsilon} = \bar{b}$ then output inverse of $xa_i^\varepsilon$ and stop.
        if $\mathrm{Size}(S) < M$ then
            Add $\left(\left|\bar{c}^{a_i^\varepsilon}\right|, \bar{c}^{a_i^\varepsilon}, xa_i^\varepsilon\right)$ to $S$ and sort $S$ by first element of every tuple.
        else                                                ▷ no more space in S
            if $\left|\bar{c}^{a_i^\varepsilon}\right|$ is smaller than first element of all tuples in S then swap them
        end if
    end for
end while
Otherwise, output FAIL.                                     ▷ no more elements to conjugate
```

The name LBA* comes from the general idea of *A* search algorithm* [8], which uses a best-first search (as we are doing here – taking the smallest element of S and conjugated it). We should note that a very similar algorithm was independently introduced by Tsaban [21], and the difference between the two variants is that our variant starts the search from $\overline{b'}$, while Tsaban's variant starts the search from both directions: $\overline{b'}$ and $\overline{b'}$ (using the idea of "meet in the middle").

4.2 The length function

In the implementation of the LBA, the choice of the length function is important (see [5, 9]). In our case, the length of a word is chosen to be the sum of the absolute values of the exponents in its normal form. We choose this function because the experimental results presented below show that it satisfies the requirement $\ell(a^{-1}ba) \gg \ell(b)$ (as needed for a length function used for LBA).

The first step of the experiments is the construction of a polycyclic group G of a given Hirsch length $h(G)$, following the construction in Sections 3.2 and 5.1. Then, an element b of length between 10 and 13 is randomly chosen; we choose elements of this length for consistency with the LBA parameters. Another random element a satisfying the same length interval is chosen and b^a is computed, and finally, we compute $|b^a| - |b|$. We performed 100 tests for each group and the average difference is recorded.

Polynomial	$h(G)$	Average difference
$x^2 - x - 1$	3	79.92
$x^5 - x^3 - 1$	7	80.17
$x^{11} - x^3 - 1$	16	44.93

As we can see, the average difference is large; specifically $|b^a| - |b|$ is significantly larger than $|a|$, indicating that the condition $\ell(a^{-1}ba) \gg \ell(b)$ is indeed satisfied.

5 Experimental results

Our goal is to apply the LBA on the AAG protocol having the polycyclic group as its underlying platform. To that end, we implemented the four variants of the LBA presented in Section 4 and performed experiments on several polycyclic groups having different Hirsch lengths.

5.1 Implementation details

Each polycyclic group is constructed by choosing an irreducible polynomial f over $\mathbb{Z}$, thus f defines an algebraic field F over $\mathbb{Q}$. Let O_F be its maximal order and U_F be its unit group, thus $O_F \rtimes U_F$ is the desired polycyclic group. This construction follows [10] and is a part of the Polycyclic package of GAP (see [4]).

A random element a_i, for Alice's public set, or b_i, for Bob's public set, is generated by taking either some random generators of the group or their inverses and multiplying them together, while maintaining that the length of the element is between a predefined minimum and maximum. By this method, we take control over the length of the element.

Alice's private key A is generated by taking a fixed number of random elements in $\overline{a} = (a_1, \dots, a_{N_1})$ and multiplying them together. Here we forgo control over length to preserve interesting cases of conjugations actually decreasing the length of b_i, such as a commutator-type peak. The way for choosing the keys is similar to what has been used in [17]. This way also reflects the characterization of the polycyclic group.

5.2 Results

We performed several sets of tests, all of which were run on an Intel Core I7 quad-core 2.0 GHz computer with 12 GB of RAM, running Ubuntu Version 12.04 with GAP Version 4.5 and 10 GB of memory allocation. In all these tests, the polycyclic group G having Hirsch length $h(G)$ is constructed by the above method using polynomial f. The sizes of Alice's and Bob's public sets are both $N_1 = N_2 = 20$.

5.2.1 The effect of the Hirsch length

In the first set of tests, the length of each random element a_i or b_i is in the interval $[L_1, L_2] = [10, 13]$ and Alice's private key is the product of $L = 5$ random elements in Alice's public set. The time for each batch of 100 tests is recorded together with its success rate. In each case, a time-out of 60 minutes is enforced for each test. The following results are obtained by LBA with a dynamic set

Polynomial	$h(G)$	Time	Success rate of LBA with a dynamic set
$x^2 - x - 1$	3	0.20 hours	100%
$x^5 - x^3 - 1$	7	76.87 hours	35%
$x^7 - x^3 - 1$	10	94.43 hours	8%
$x^9 - 7x^3 - 1$	14	95.18 hours	5%
$x^{11} - x^3 - 1$	16	95.05 hours	5%

From this table, we can see that with a small Hirsch length, the LBA cryptanalyzes the AAG protocol easily with high success rate. However, as the Hirsch length is increased to 7, the success rate decreases. In polycyclic groups with higher Hirsch lengths, we can see the effect of the time-out more prominently as the total time did not increase much more, but the success rate is dropped to 5%. Although a success rate of 5% is not negligible, note that we use a very small value for L. Based on the current experimental results, we expect that increasing the value of L will reduce the success rate to 0%.

5.2.2 The effect of the key length

In the second set of tests, we vary the number of elements L that compose Alice's private key. Myasnikov and Ushakov [17] suggested that the LBA with a dynamic set has a high success rate with long generators, i.e. random elements have longer length $[L_1, L_2]$. Therefore, we also vary the length of random elements according to the parameters in [17].

The following results are obtained by LBA with a dynamic set, with a time-out of 30 minutes:

Polynomial	$h(G)$	[10, 13]	[20, 23]		[40, 43]
		$L = 10$	$L = 10$	$L = 20$	$L = 50$
$x^7 - x^3 - 1$	10	2%	0%	0%	0%
$x^9 - 7x^3 - 1$	14	0%	0%	0%	0%
$x^{11} - 3x^3 - 1$	17	0%	0%	0%	0%

The results of this set of tests indicate that just by increasing the number of generators of Alice's private key from 5 (as in the previous set of tests) to 10, the LBA already fails with polycyclic groups having Hirsch length as small as 10.

5.2.3 Comparing the four variants of the LBA

In this paper, we compare the success rates of the four variants of the LBA *for the first time* on any platform. For comparing the success rates of the four variants of the LBA, we purposely choose the value of the test parameters to be very small in this set of tests. They are as follows: $N_1 = N_2 = 20$, $[L_1, L_2] = [5, 8]$, $L = 5$, there is a time-out of 30 minutes and a memory of size $M = 500$. The polynomial used is $f = x^3 - x - 1$, constructing a polycyclic group of Hirsch length 4.

Algorithm	**Time**	**Success rate**
LBA with backtracking	0.57 hours	58%
LBA with a dynamic set	37.35 hours	95%
Memory-LBA (with memory $M = 500$)	4.01 hours	92%
LBA* (with memory $M = 500$)	32.00 hours	36%

Algorithm LBA with a dynamic set gives the best success rate but took much longer than Algorithm Memory-LBA which gives a similar success rate in much shorter time. We conclude that with a sufficient size of memory, Algorithm Memory-LBA is the best variant of the LBA.

5.2.4 Using the four variants of the LBA on our test parameters

In the fourth set of tests, we want to see the effect of the four different variants of the LBA presented in Section 4.1 applied to our test parameters. Therefore, we keep the following parameters for all the algorithms: the length of each random element is in the interval $[L_1, L_2] = [10, 13]$, Alice's private key is the product of 10 elements and the length of both public sets are $N_1 = N_2 = 20$. There is a time-out of 30 minutes per test and in the case of the two memory variants of the LBA, Algorithm Memory-LBA and Algorithm LBA*, a memory of size $M = 1000$ is used. The same polycyclic group G having Hirsch length 14 constructed by the polynomial $x^9 - 7x^3 - 1$ is used for all the variants of the LBA.

Algorithm	**Time**	**Success rate**
LBA with backtracking	48.68 hours	0%
LBA with a dynamic set	50.04 hours	0%
Memory-LBA (with memory $M = 1000$)	49.35 hours	3%
LBA* (with memory $M = 1000$)	50.00 hours	0%

As we can see, Memory-LBA algorithm has the best performance in this set of parameters, but even then, it has only 3% success rate. To further test Memory-LBA algorithm, we run another set of tests where we increase the length of random elements to $[L_1, L_2] = [20, 23]$ and increase the number of factors of the private key to $L = 20$. To give it a chance of success, we increase the size of the memory M to 40,000. The result is 0% success rate.

5.2.5 The effect of increasing the time-out

Since it is possible that the time-out of 30 minutes for each test is too short, we run another set of tests, where the time-out is 4 hours for each test. Memory-LBA algorithm showed the most promise, so we chose it with the following parameters: the length of random elements is in the interval $[L_1, L_2] = [20, 23]$, the number of factors of the private key is $L = 20$ and the size of the memory M is 1000. The polynomial used is $x^9 - 7x^3 - 1$ producing a polycyclic group of Hirsch length 14. Due to the long time-out, we performed only 50 tests. We still get 0% success rate.

Based on the above experimental results, we conclude that the LBA is insufficient for cryptanalyzing the polycyclic groups of high enough Hirsch lengths. One can suggest the following parameters: $h(G) = 16$, $L = 20$ and $[L_1, L_2] = [20, 23]$ for achieving an AAG protocol based on the polycyclic group, which the known variants of the LBA have 0% success rate for cryptanalyzing this protocol.

5.2.6 Additional experimental results concerning LBA with a dynamic set algorithm

Here, we present some additional experimental results for LBA with a dynamic set. The time-out for each test is 1 hour. The polynomials used are f and $h(G)$ is the Hirsch length of the corresponding polycyclic group. The sizes of Alice's and Bob's public sets are N_1, N_2 respectively. Each random element a_i or b_i has length in $[L_1, L_2]$ and Alice's private key is the product of $L = 5$ random elements in Alice's public set. The success rate of a batch of 100 tests is recorded.

Polynomial	$h(G)$	$N_1 = N_2 = 5$		$N_1 = N_2 = 20$
		$[5, 8]$	$[15, 18]$	$[10, 13]$
$x - 1$	1	98%		98%
$x^2 - x - 1$	3	98%	96%	100%
$x^3 - x - 1$	4	95%		100%
$x^5 - x^3 - 1$	7			35%
$x^7 - x^3 - 1$	10			8%
$x^9 - 7x^3 - 1$	14			5%
$x^{11} - x^3 - 1$	16	59%	53%	5%

Acknowledgement: We would like to thank the anonymous referees for many useful suggestions, which were implemented in the text.

Funding: Delaram Kahrobaei is partially supported by the Office of Naval Research grant N000141210758, a PSC-CUNY grant from the CUNY Research Foundation, and the City Tech Foundation.

References

[1] I. Anshel, M. Anshel and D. Goldfeld, An algebraic method for public-key cryptography, *Math. Res. Lett.* **6** (1999), 287–291.
[2] M. Anshel and D. Kahrobaei, Decision and search in non-abelian Cramer–Shoup public key cryptosystem, *Groups Complex. Cryptol.* **1** (2009), 217–225.
[3] B. Eick and D. Kahrobaei, Polycyclic groups: A new platform for cryptology?, preprint (2004), http://arxiv.org/abs/math/0411077.
[4] B. Eick and W. Nickel, *Polycyclic: Computation with polycyclic groups, a GAP 4 package*, www.gap-system.org/Packages/polycyclic.html.

[5] D. Garber, S. Kaplan, M. Teicher, B. Tsaban and U. Vishne, Probabilistic solutions of equations in the braid group, *Adv. App. Math.* **35** (2005), 323–334.

[6] D. Garber, S. Kaplan, M. Teicher, B. Tsaban and U. Vishne, Length-based conjugacy search in the braid group, *Contemp. Math.* **418** (2006), 75–87.

[7] V. Gebhardt, Efficient collection in infinite polycyclic groups, *J. Symb. Comput.* **34** (2002), 213–228.

[8] P. E. Hart, N. J. Nilsson and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybern.* **4** (1968), no. 2, 100–107.

[9] M. Hock and B. Tsaban, Solving random equations in Garside groups using length functions, in: *Combinatorial and Geometric Group Theory* (Dortmund, Ottawa and Montreal 2007), Birkhäuser, Basel (2010), 149–169.

[10] D. F. Holt, B. Eick and E. A. O'Brien, *Handbook of Computational Group Theory*, Chapman & Hall/CRC Press, Boca Raton, 2005.

[11] J. Hughes and A. Tannenbaum, Length-based attacks for certain group based encryption rewriting systems, in: *Workshop SECI02 Securite de la Communication sur Internet* (2002), 5–12.

[12] D. Kahrobaei and B. Khan, A non-commutative generalization of El-Gamal key exchange using polycyclic groups, in: *Proceedings of the Global Telecommunications Conference 2006* (2006), 1–5.

[13] D. Kahrobaei and C. Koupparis, Non-commutative digital signatures, *Groups Complex. Cryptol.* **4** (2012), no. 2, 377–384.

[14] A. Kalka, B. Tsaban and G. Vinokur, Complete simultaneous conjugacy invariants in Garside groups, preprint (2014), http://arxiv.org/abs/1403.4622.

[15] K. H. Ko, S. J. Lee, J. H. Cheon, J. W. Han, J. Kang and C. Park, New public-key cryptosystem using braid groups, in: *Advances in Cryptology* (CRYPTO 2000), Lect. Notes in Comp. Sci. 1880, Springer, Heidelberg (2000), 166–183.

[16] A. Myasnikov, V. Shpilrain and A. Ushakov, *Non-Commutative Cryptography and Complexity of Group-Theoretic Problems*, American Mathematical Society, Providence, 2011.

[17] A. D. Myasnikov and A. Ushakov, Length-based attack and braid groups: Cryptanalysis of Anshel–Anshel–Goldfeld key-exchange protocol, in: *Public Key Cryptography* (PKC 2007), Lecture Notes Comp. Sci. 4450, Springer, Heidelberg (2007), 76–88.

[18] A. G. Myasnikov and A. Ushakov, Random subgroups and analysis of the length-based and quotient attacks, *J. Math. Cryptol.* **2** (2008), no. 1, 29–61.

[19] D. Ruinskiy, A. Shamir and B. Tsaban, Length-based cryptanalysis: The case of Thompson's group, *J. Math. Cryptol.* **1** (2007), 359–372.

[20] I. Stewart and D. O. Tall, *Algebraic Number Theory and Fermat's Last Theorem*, A.K. Peters, Natick, 2002.

[21] B. Tsaban, The conjugacy problem: Cryptoanalytic approaches to Dehn's problem, slides of a minicourse given in GAGTA-6 Conference (Düsseldorf 2012), http://reh.math.uni-duesseldorf.de/~gcgta/slides/Tsaban_minicourses.pdf.

Received January 27, 2014; revised September 26, 2014; accepted November 4, 2014.