A parallel evolutionary approach to solving systems of equations in polycyclic groups

Craven, Matthew J.; Robertz, Daniel

Published in:
Groups Complexity Cryptology

DOI:
10.1515/gcc-2016-0012

Publication date:
2016

Groups Complex. Cryptol. ? (????), 1–27
DOI 10.1515/gcc-????-???

A parallel evolutionary approach to solving systems of equations in polycyclic groups

Matthew J. Craven and Daniel Robertz

Abstract. The Anshel-Anshel-Goldfeld (AAG) key exchange protocol is based upon the multiple conjugacy problem for a finitely-presented group. The hardness in breaking this protocol relies on the supposed difficulty in solving the corresponding equations for the conjugating element in the group. Two such protocols based on polycyclic groups as a platform were recently proposed and were shown to be resistant to length-based attack. In this article we propose a parallel evolutionary approach which runs on multicore high-performance architectures. The approach is shown to be more efficient than previous attempts to break these protocols, and also more successful. Comprehensive data of experiments run with a GAP implementation are provided and compared to the results of earlier length-based attacks. These demonstrate that the proposed platform is not as secure as first thought and also show that existing measures of cryptographic complexity are not optimal. A more accurate alternative measure is suggested. Finally, a linear algebra attack for one of the protocols is introduced.

Keywords. Evolutionary algorithms, polycyclic groups, cryptography, Anshel-Anshel-Goldfeld key agreement protocol, high performance computing, parallel.

2010 Mathematics Subject Classification. 20P05, 68W30, 90C27, 94A60.

1 Introduction

Since the late nineties there have been many cryptosystems and key exchange protocols proposed based on group theory. Two of the most popular ones [2, 20] had base problems posed over a braid group platform. These protocols were thought to be secure by the complexity of algebraic and computational attacks. For example, the conjugacy problem in the braid group was originally solved by [14] in 1969 but the solution was of exponential complexity, whereas the word problem was eminently solvable. The Anshel-Anshel-Goldfeld (AAG) key exchange protocol [1] was based upon the (subgroup restricted) multiple conjugacy problem and that of [20] on the conjugacy problem. Over the next few years, both protocols were attacked through attempts to solve the base problem. For example, super summit set attacks [9] and length-based attacks (LBAs) (of which [18] was the first) were used. The LBA was strengthened subsequently by [12, 13, 22, 25]. The work [6]

also successfully proposed an attack on the multiple conjugacy problem over a homomorphic preimage of the braid group.

Since then there have been efforts to find platform groups that offer security. Polycyclic groups are but one platform [7], over which, again, the word problem is quickly solvable but the conjugacy problem has been said to be difficult [11]. Two key exchange protocols based on AAG over polycyclic groups have been proposed. The work of [11] proposed a protocol over polycyclic groups which are defined by a number field and applied several LBA algorithms against the protocol, with instances indexed by Hirsch length, finding that for a high Hirsch length LBA is unsuccessful. The work of [19] proposed a protocol over Heisenberg groups, a specific kind of polycyclic group defined in terms of matrices. These authors again applied LBA algorithms with similar results, concluding that polycyclic groups are indeed a viable platform group for cryptology.

In this article, a method for attacking the above two protocols of [11, 19] is proposed. The method is based on an evolutionary algorithm (EA), which overcomes common deficiencies of LBAs, showing that the protocols based on polycyclic groups are not as secure as suggested. This is shown through analysis of a large number of runs of the method. From this, it is also demonstrated that the Hirsch length is not an optimal measure of the complexity of a group (and so hardness of the problem). Moreover, an algebraic solution for the protocol over Heisenberg groups [19] is presented. (Note, subsequent to submission of this article, an independent solution for the generalised Heisenberg case was published [24].)

The approach is inherently parallelisable. One important effect of a parallel search of a solution is that, in comparison to LBAs, larger parts of the search space can be investigated in a much shorter time. The method was implemented in GAP [10] and applied to many examples in a high performance computing (HPC) environment. An extensive list of benchmarks is provided, the majority of which were adopted from [11, 19]. A detailed analysis of these runs is included, comparing the results to those of [11, 19] and providing additional statistical data.

The article is structured as follows. In Section 2 the proposed AAG-like protocols are summarised, before detailing a general scheme of the EA attack in Section 3. In Section 4 we give an implementation in ParGAP, enabling us to achieve a speedup in the attack through the use of multiple cores in a high performance computing environment. Section 5 gives the application of the EA to the two proposed protocols, directly comparing results with the work of [11, 19] before proposing a direct algebraic attack on [19]. We conclude the article in Section 6.

2 The Problem

The work of [11,19] presented AAG over two different types of polycyclic groups. The description below follows that of [11].

Suppose Alice and Bob are two individuals who wish to exchange cryptographic keys so that they may communicate in secret. They each have their own (private) keys and, via the key exchange protocol, wish to combine both keys into a shared key known to only both of them. Let $G = \langle\, g_1, \ldots, g_n \mid R \,\rangle$ be a finitely presented group. Positive integer-valued cryptographic parameters N, L, L_1 and L_2 are assumed to have been chosen.

The standard key exchange in the literature proceeds as follows.

(i) Alice chooses a public N-tuple $(a_i)_{i=1}^N = (a_1, \ldots, a_N)$ of words in the generators of G, each having length in the interval $[L_1, L_2]$ and publishes it;

(ii) Similarly, Bob chooses a public N-tuple $(b_i)_{i=1}^N = (b_1, \ldots, b_N)$ of words in the generators of G, each having length in $[L_1, L_2]$ and publishes it;

(iii) Alice chooses a private key $A = a_{\mu_1}^{\epsilon_1} a_{\mu_2}^{\epsilon_2} \ldots a_{\mu_L}^{\epsilon_L}$ where $\mu_i \in \{1, \ldots, N\}$, $\epsilon_i \in \{\pm 1\}$ for all $i = 1, \ldots, L$. Here L denotes the length of the key A in the subgroup $\langle\, a_i \mid i = 1, \ldots, N \,\rangle$;

(iv) Similarly, Bob chooses a private key $B = b_{\nu_1}^{\delta_1} b_{\nu_2}^{\delta_2} \ldots b_{\nu_L}^{\delta_L}$ with $\nu_i \in \{1, \ldots, N\}$, $\delta_i \in \{\pm 1\}$ for all $i = 1, \ldots, L$;

[Alice now holds A, $(a_i)_{i=1}^N$ and $(b_i)_{i=1}^N$, and Bob holds B, $(a_i)_{i=1}^N$ and $(b_i)_{i=1}^N$.]

(v) Alice computes the tuple of conjugates $A^{-1}\,(b_i)_{i=1}^N\,A$ and transmits it to Bob;

(vi) Similarly, Bob computes the tuple of conjugates $B^{-1}\,(a_i)_{i=1}^N\,B$ and transmits it to Alice.

Even without knowing Bob's private key B, Alice can use her private key A and $B^{-1}a_1B$, $B^{-1}a_2B$, …, $B^{-1}a_NB$ to compute

$$\begin{aligned} A^{-1}\left((B^{-1}a_{\mu_1}B)^{\epsilon_1} \ldots (B^{-1}a_{\mu_L}B)^{\epsilon_L}\right) &= A^{-1}\left(B^{-1}a_{\mu_1}^{\epsilon_1}B \ldots B^{-1}a_{\mu_L}^{\epsilon_L}B\right) \\ &= A^{-1}\left(B^{-1}a_{\mu_1}^{\epsilon_1} \ldots a_{\mu_L}^{\epsilon_L}B\right) \\ &= A^{-1}B^{-1}AB. \end{aligned}$$

Similarly, Bob can compute the inverse of the above key and then by inversion reach the identical key to Alice. Breaking the above protocol means solving a *subgroup restricted multiple conjugacy search problem (MCSP)*,

$$A^{-1}\,(b_i)_{i=1}^N\,A = (c_i)_{i=1}^N \tag{1}$$

of which the objective is to find the private key of either recipient.

The groups G to be dealt with here are assumed to be polycyclic. By definition (cf., e.g., [17]), there exists a subnormal series $G = G_1 \rhd G_2 \rhd \ldots \rhd G_{n+1} = \{1\}$ for some non-negative integer n such that each factor group G_i/G_{i+1} is cyclic, $i = 1, \ldots, n$. Elements $g_1, \ldots, g_n$ of G such that, for each i, the coset $g_i G_{i+1}$ generates G_i/G_{i+1} form a generating set for G. If $r_i \in \mathbb{N} \cup \{\infty\}$ is the order of $g_i G_{i+1}$ in G_i/G_{i+1}, then the number of infinite r_i's does not depend on the above choices and is referred to as the Hirsch length, $h(G)$, of G. Due to its polycyclic structure, G admits a presentation

$$\begin{aligned} G \cong \langle\, g_1, \ldots, g_n \quad | \quad & g_i^{r_i} = w_{i,i} \quad \text{for } i = 1, \ldots, n \text{ such that } r_i \neq \infty, \\ & g_k^{-1} g_j g_k = w_{j,k},\ g_k g_j g_k^{-1} = w_{k,j} \quad \text{for } 1 \leq k < j \leq n \,\rangle, \end{aligned}$$

where each $w_{i,j}$ is a word in the generators g_k of the form $g_{\min(i,j)}^{e_{\min(i,j)}} \cdots g_{n-1}^{e_{n-1}} g_n^{e_n}$ with $e_k \in \mathbb{Z}$ and $0 \leq e_k < r_k$. With respect to such a presentation every element of G has a unique normal form, which can be computed, so that the word problem for G has an effective solution.

The approach used to solving problem (1) will be to convert it into a combinatorial optimisation problem so it may be then solved via an optimisation algorithm. The combinatorial optimisation problem is then as follows:

Minimise

$$K(\alpha) := \sum_{i=1}^{N} \text{length}(\alpha^{-1} b_i \alpha c_i^{-1}) \tag{2}$$

where length(g) refers to the length of the normal form of g in G.

The next section introduces the rationale behind the approach.

3 Stochastic optimisation algorithms

The realm of stochastic optimisation algorithms houses many methods of using probabilities to find the optimum of a given function (or functions) over a continuous or discrete search space. Two such examples are given below.

3.1 Length attack algorithms

A length attack (or length-based attack, LBA) may also be known as a random mutation hillclimbing algorithm [16]. The essential idea is that a solution to a

problem may be built from "building blocks" (in this case, the generators of the group will be the building blocks from which the solution word is built):

$$g_{i_1}^{\epsilon_{i_1}}$$

$$g_{i_1}^{\epsilon_{i_1}} g_{i_2}^{\epsilon_{i_2}}$$

$$\dots$$

$$g_{i_1}^{\epsilon_{i_1}} g_{i_2}^{\epsilon_{i_2}} \dots g_{i_l}^{\epsilon_{i_l}}$$

In this case, $i_k \in \{1, \dots, n\}$ and $\epsilon_{i_k} \in \{-1, +1\}$ for $k = 1, \dots, l$. A simplistic example of an LBA begins with a word α, of which a generator is appended to the end of the word to make a new word, α'. Appending a new generator to the current word will usually not cause a cancellation in the new word. If $K(\alpha') < K(\alpha)$ then let $\alpha \leftarrow \alpha'$ and repeat. If the correct generator is chosen each time, then this process converges to a solution. Several applications of this method have been attempted in the domain of group-based cryptography [11–13, 18, 19, 22, 23, 25], involving memory and other mixed approaches.

Such algorithms have common deficiencies. The field of search of the algorithm is narrow, for example, because of a lack of parallel search. That is, the algorithm moves down one branch of the search tree. Along this branch local search may reveal no generators such that $K(\alpha') < K(\alpha)$ holds, giving only a local minimum in the cost landscape. This makes it extremely difficult, and sometimes impossible, to move to another point of decreased cost in the landscape. This is compounded by the realisation that an LBA technique depends on its initial guess for a building block, and so a good initial guess is often required. In addition, during the LBA process, suboptimal candidate solutions which may lead to branches of lesser cost are not examined, imposing further efficiency constraints on this type of algorithm.

On the other hand, such an algorithm may be simple to analyse (in terms of its time complexity or runtime). Various strategies exist to assist with the above issues (e.g., lookahead, where a number of generators are appended) but the basic flaw – a lack of parallel search – still remains. The next subsection gives a generalisation of LBAs to a larger class of algorithms that is believed to overcome the above deficiencies.

3.2 Evolutionary algorithms

Evolutionary algorithms form a class of iterative probabilistic population-based optimisation algorithms which take advantage of the principles of natural evolution. An EA begins with a random initial population of candidate solutions (initial guesses) to a given problem. Through simulation of Darwinian operations such as

natural selection (which is elitist), mutation and crossover, these solutions breed to create further generations. The above operations are performed relative to a best-fitness-first ranking of the population ("survival of the fittest"). In this way, evolution continues producing candidate solutions to the problem which are of equal or higher fitness to those found previously, terminating when some condition is invoked. In the case of this work, the chosen EA terminates when an exact solution to (1) is reached.

There are no restrictions upon operations, and such operations are typically performed at any position in a candidate solution. EAs are also inherently parallel; that is, a large proportion of the search space of all possible candidate solutions is capable of being explored simultaneously. This allows potential solutions which may have been off-limits to an LBA to be reached, and independently of the initial guesses. EAs are generally thought to provide efficiency gains over more restrictive algorithms such as an LBA, making it a more suitable class of algorithms for searching a search tree (or search space) that is very large.

Finally, operations which overcome local minima of the cost function by enabling jumps to be made in the cost landscape are often included. An example of such an operation is crossover, which enables large changes to be made to candidate solutions and the EA jumps to another branch of the search tree as a result. It is, however, not an easy task at all to detect whether the search is in a local minimum or not. In the interests of brevity, further discussion of EAs in general is omitted in this work. Further information may be found in other references (e.g., [16]). The next section gives details of the implementation.

4 An implementation in ParGAP

The EA developed in this paper was implemented in GAP [10]. The GAP package Polycyclic [8] was used for processing of polycyclic groups, and ParGAP [4] to implement parallelisation and make the attack more efficient.

4.1 Parallel approach

The EA described in this paper uses parallel processing in a high-performance computing environment.

ParGAP is executed on a given number of cores, one process being a master process and the others slave processes which are controlled by MPI (Message Passing Interface). The master maintains a list of candidate solutions (individuals of the population) to the MCSP instance. More precisely, G is given by a finite presentation, i.e., $G = F/[R]$, where F is a free group of rank n and $[R]$ is the smallest normal subgroup of F containing the given relators. The list of candidates, which

are elements in F, is sorted with respect to a cost function, where comparison of list entries is delegated to the slave processes. The cost function is assumed to evaluate to zero precisely those words in the generators $f_1, \ldots, f_n$ of F which map to solutions of the MCSP instance in G. The comparisons for sorting the list of candidates involve computing the normal forms as required by (2).

The algorithm allows flexible operation at any point in the word and has a mechanism to make local minima in the cost landscape less likely (see Section 4.4).

4.2 Representation and operations

The individuals (candidate solutions) are words w in the generators $f_1, \ldots, f_n$ of F, as are the chosen generators a_i of a subgroup of F. The length of w is denoted by $\ell(w)$, and $w[s \ldots t]$ is the subword of w from position s to t (which is empty if $s > t$). The operators of the EA fall into several categories. It is acknowledged that other operators could have been used. It is also worth stating that when generators are chosen from a group or subgroup, this includes inverses, and that when an action is performed at random the uniform distribution is assumed.

- Mutation: choose a word w from the top $m\%$ of the population by cost value (smallest cost first). Then perform one of the following types of mutation:

  (i) By insertion of subgroup generator: insert a generator a_i at the immediate left of a random position $r \in \{1, \ldots, \ell(w) + 1\}$ (so including the possibility of insertion at the left of position 1 or at the right of position $\ell(w)$). That is, assign

  $$w' = w[1 \ldots r-1]a_i w[r \ldots \ell(w)];$$

  (ii) By insertion of free group generator: identical to operation (i) but with a_i replaced by f_i;

  (iii) By deletion: delete a generator at a random position $r \in \{1, \ldots, \ell(w)\}$ of w. This gives the word

  $$w' = w[1 \ldots r-1]w[r+1 \ldots \ell(w)];$$

  (iv) By substitution: insert a generator f_i in place of the given one at a random position $r \in \{1, \ldots, \ell(w)\}$. The word

  $$w' = w[1 \ldots r-1]f_i w[r+1 \ldots \ell(w)]$$

  is obtained;

(v) By position conjugation: take a random position $r \in \{1, \ldots, \ell(w)\}$. Conjugate the generator at position r by a random generator f_i, i.e.

$$w' = w[1 \ldots r-1] f_i^{-1} w[r] f_i w[r+1 \ldots \ell(w)];$$

(vi) By subword conjugation: take a random position $r \in \{1, \ldots, \ell(w)\}$ and a positive integer s such that $r + s \leq \ell(w) + 1$. Conjugate the subword of length s starting at position r by a random generator f_i. That is, assign

$$w' = w[1 \ldots r-1] f_i^{-1} w[r \ldots r+s-1] f_i w[r+s \ldots \ell(w)].$$

- Crossover: choose two words w_1, w_2 from the top $m\%$ of the population by cost value. Choose two random numbers $r_1 \in \{1, \ldots, \ell(w_1)\}$, $r_2 \in \{1, \ldots, \ell(w_2)\}$. At random, output either of the two words

$$\begin{aligned} w' &= w_1[1 \ldots r_1] w_2[r_2 + 1 \ldots \ell(w_2)], \\ w' &= w_2[1 \ldots r_2] w_1[r_1 + 1 \ldots \ell(w_1)]. \end{aligned}$$

- Selection: elitist. Select the top word in the population by cost value and $n_s - 1$ words from the top $m\%$ of the population. Selection is performed in this way for simplicity of operation, although it is acknowledged there are other selection methods (cf., for example, [16]).

The new population is formed of the resulting words w'. The above operations are performed according to set EA parameters, which sum to the (fixed) population size. The implementation uses a population size of 100, $m = 20$ (to allow for population diversity) and the parameter vector $\mathcal{S} = (36, 1, 1, 35, 1, 1, 20, 5)$ for the number of operations given in the above order (hence, e.g., $n_s = 5$). Among various choices of parameter vectors in experiments for polycyclic groups, as discussed in Section 5.4, the above vector generally led to the minimal number of generations for many problem instances.

4.3 Cost function

Given a word $w = f_{i_1}^{e_{i_1}} f_{i_2}^{e_{i_2}} \ldots f_{i_r}^{e_{i_r}}$, where $e_j \in \mathbb{Z}$ are non-zero, f_1, f_2, …, f_n are the generators of F, and consecutive pairs of i_1, i_2, …, $i_r \in \{1, \ldots, n\}$ are distinct, the following length functions are used to evaluate costs:

$$\ell(w) = \sum_{k=1}^{r} |e_{i_k}|$$

$$\ell_{wt}(w) = \sum_{k=1}^{r} \omega_{i_k} |e_{i_k}|,$$

where the weight ω_j is defined as the sum of the lengths of the normal forms of the commutators $[g_j, g_k]$ in G for $k = 1, \ldots, n$.

The EA cost function is a metric of how close a candidate solution is to an exact solution. Of course, there may be many ways in measuring this. The cost function defined in the given implementation is a tuple composed of the following components (reached by experimentation), where $\lambda_i(\alpha) := \ell(\alpha^{-1} b_i \alpha c_i^{-1})$ and $\mu := \frac{1}{N} \sum_{i=1}^{N} \lambda_i(\alpha)$:

$$S_u(\alpha) = \sum_{i=1}^{N} \lambda_i(\alpha), \quad S_{wt}(\alpha) = \sum_{i=1}^{N} \ell_{wt}(\alpha^{-1} b_i \alpha c_i^{-1}), \quad W_u(\alpha) = \sum_{i\,:\,\lambda_i(\alpha) > \mu} \lambda_i(\alpha),$$

$$\max_u(\alpha) = \max_{i=1,\ldots,N} \lambda_i(\alpha), \quad \min_u(\alpha) = \min_{i=1,\ldots,N} \lambda_i(\alpha),$$

in a certain order, with $\ell(\alpha)$ as the last component. Cost tuples of individuals α are then compared lexicographically. An individual α is an exact solution of the MCSP if and only if any one of the first five components is zero (provided not all weights ω_j are zero, in which case G is abelian).

4.4 Algorithm layout

For a vector $\mathcal{S}$ of numbers, the notation $\mathcal{S}[i]$ is used to mean the number at position i of $\mathcal{S}$. Several features are identified before the algorithm pseudocode is given.

The first EA feature is that of detection of likely local minima of the cost function. One of the EA inputs is a "wait period", ε, such that if the minimal cost word α in each generation has identical cost C over ε subsequent generations, then the EA parameter vector $\mathcal{S}$ is perturbed. This perturbation has the effect of biasing the algorithm to operations which may be more likely to result in individuals of cost less than the previous minimal cost C. The perturbation proceeds as in Algorithm 1, rendering the EA adaptive. This perturbation repeats every ε generations if no cost reduction occurs. If there is a cost reduction then the current parameter vector is reverted to the initial parameter vector.

(A perturbation of parameter values as above is more likely to result in an escape of the local minimum if the operation whose frequency is changed is not 'selection', i.e., $p_1 \neq 8$; the choice of a rather small value for $\mathcal{S}[8]$ ensures this.)

Second on the list of EA features is that of cost and length control. In some cases, it may be true that a given candidate solution (word) may have a high cost in comparison to the word of current least cost in the population. This means that

Algorithm 1 Parameter Perturbation

Input: wait period ε, parameter vector $\mathcal{S}$
Output: perturbed parameter vector $\mathcal{S}$

$S_{\max} \leftarrow \max_i(\mathcal{S})$
$S_p \leftarrow \{j \in \{1, \ldots, 8\} : \mathcal{S}[j] = S_{\max}\}$
choose p_1 at random from the set S_p
choose p_2 at random from the set $\{\mathcal{S}[j] : j = 1, \ldots, 8\} \setminus S_p$
choose δ at random from the range $\mathcal{S}[p_2], \ldots, S_{\max}$
$\mathcal{S}[p_1] \leftarrow \mathcal{S}[p_1] - \delta$
$\mathcal{S}[p_2] \leftarrow \mathcal{S}[p_2] + \delta$
return $\mathcal{S}$. End.

not only will the former word likely be ranked lower in the population, but also that it takes a long time to process and evaluate. Thus the rule was instituted that if a word has cost greater than some given limit (which may be set to a high value) then the word is replaced by a random word of length one.

Similarly, word length of candidate solutions was found to be an issue for some instances. For example, in the case of polycyclic groups associated to number fields with defining polynomials of high degree (e.g., degree 9 or 11) the presentations contained relators of high length (see Section 5.6) which implies that normal form computations result in long strings of generators. This typically results in population drift towards very long words, costing a considerable amount of time. Thus, similarly to the above, the rule that if a word had length greater than a given limit then it was replaced by a word of length one, was instituted. It was concluded that this limit on length should be appropriate for the relator lengths of the group in question. Suitable cost and length limits were found by experimentation and are indicated where used.

The pseudocode for the EA is given by Algorithm 2 with the full code provided at the URL `https://github.com/MJCraven/PCyclic` .

In the next section, the application of the above EA to both proposed public key protocols mentioned in the introduction is given.

5 Applications

Both proposed public key protocols [11, 19] are based upon the AAG scheme [1]. As in the first two references above, the number of equations in each MCSP instance was set to $N = 20$ unless otherwise stated. The following six subsections detail applications of the EA, treating the Heisenberg groups case followed by the

Algorithm 2 Evolutionary Algorithm

Input: population size p, maximum number of generations M, MCSP instance, wait period ε, maximum permitted word length $\ell_{\max}$, parameter vector $\mathcal{S}$, target cost value $C_{\min}$ (usually zero), slave processors configuration
Output: solution of the MCSP instance or timeout

1: $g \leftarrow 1$
2: generate initial population $\mathcal{P}' = \{w_1, \ldots, w_p\}$ as a multiset of random words of length one from F
3: **while** $g \leq M$ **do**
4: $\mathcal{P} \leftarrow \mathcal{P}'$
5: compute the cost vector $C(w_i)$ for each w_i in $\mathcal{P}$, distributing the computation among all slave processors
6: sort $\mathcal{P}$ in increasing order with respect to the lexicographic order:

$$C(w_{i_1}) \leq_{\text{lex}} C(w_{i_2}) \leq_{\text{lex}} \ldots \leq_{\text{lex}} C(w_{i_p})$$

7: **if** $S_u(w_{i_1}) \leq C_{\min}$ **then** ▷ $S_u(w_{i_1})$ is the first component of $C(w_{i_1})$
8: **return** w_{i_1}. End. ▷ the word w_{i_1} is a solution to the instance
9: **end if**
10: check for local minima, executing Algorithm 1 if necessary
11: replace each w_{i_j} with $C(w_{i_j}) > C_{\max}$ or $\ell(w_{i_j}) > \ell_{\max}$ by a random word of length one from F
12: $\mathcal{P}' \leftarrow \emptyset$
13: execute operators according to the EA control parameter values in $\mathcal{S}$, each time adding to the multiset $\mathcal{P}'$ the output word(s)
14: $g \leftarrow g + 1$
15: **end while**
16: **return** timeout. End.

case of polycyclic groups defined by a number field. This order reflects the increasing difficulty of the problems. Note that the work of [11, 19] did not provide information about the distribution of runs or iterations. A key contribution of the present work is the provision of such information.

5.1 Heisenberg groups

An AAG key-exchange protocol based on the following family of polycyclic goups was proposed in [19]. For each positive integer n let H_n be the group of matrices

$$M(\mathbf{x}, \mathbf{y}, z) := \begin{pmatrix} 1 & x_1 & x_2 & \dots & x_n & z \\ 0 & 1 & 0 & \dots & 0 & y_1 \\ 0 & 0 & 1 & \dots & 0 & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & y_n \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix},$$

where $x_i, y_j, z \in \mathbb{Z}$, with matrix multiplication.

In this subsection we run a direct comparison between the EA approach and that of [19] on the above problem. In this case, it was found that the cost tuple $(S_u(\alpha), \max_u(\alpha), W_u(\alpha), \min_u(\alpha), \ell(\alpha))$ provided sufficient distinction between candidate solutions α.

Results

For each of the polycyclic groups according to the values of n shown, one hundred random instances were created and the EA was run once on each instance. The EA success rate was recorded and compared in Tables 1–2 with the success rate for [19] using LBA with a dynamic set. Each table records the following information. On the fourth column is recorded the mean number of EA generations processed ($\overline{g}$), followed by the mean time taken in seconds ($\bar{t}$) on the fifth column. Here the time spent on each problem instance is determined as the maximum of the times spent by each slave process in the high-performance computing environment. The sixth column gives a calculation for the mean time per generation (in seconds). In addition, to remove unrepresentative outliers, the 20% trimmed mean of the EA generations processed ($g_{.10}$) and time taken ($t_{.10}$) have been included (that is, the top and bottom 10% of experimental data were removed). It should also be mentioned that the means and trimmed means include the unsuccessful trials. The last column gives the standard deviation, s_g, of the number of generations. Finally, the Hirsch lengths for each of the groups are $h(H_5) = 11$, $h(H_6) = 13$, $h(H_7) = 15$ and $h(H_8) = 17$ (see the discussion in Section 5.3).

The experiments were run using five slave processors of an HPC cluster consisting of Intel Xeon E5620 CPUs running at 2.40GHz. The amount of memory used by the algorithm was typically around 150MB, and so relatively little memory was needed. The random number seed is based upon the computer clock. It was found

that (unless stated) for $5 \leq n \leq 7$, a suitable maximum number of generations to run was $M = 2500$ and for $n = 8$ a suitable maximum was $M = 5000$. A value of $\varepsilon = 10$ was used for the wait period to counteract likely local minima of the cost function.

n	[19]	EA	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$	$\overline{g}_{.10}$	$\overline{t}_{.10}$	s_g
$L_1 = 10, L_2 = 13, L = 10$								
5	29%	100%	42.8	29.8	0.70	40.6	27.0	18.4
6	69%	100%	64.9	74.1	1.14	56.4	60.8	51.0
7	51%	97%	171.6	223.1	1.30	85.8	117.5	419.9
8	62%	90%	795.5	932.8	1.17	364.1	454.1	1543.4
$L_1 = 20, L_2 = 23, L = 10$								
5	53%	100%	56.3	79.7	1.42	53.8	73.9	22.0
6	39%	100%	99.5	224.3	2.25	84.0	187.6	86.2
7	58%	93%	402.8	1360.1	3.38	226.5	875.6	647.3
8	67%	80%	1280.3	2812.2	2.20	966.6	1827.1	1899.8
$L_1 = 20, L_2 = 23, L = 20$								
5	11%	100%	67.8	223.0	3.29	60.4	200.8	42.4
6	7%	100%	115.3	422.2	3.66	96.5	351.1	93.9
7	5%	91%	514.5	2220.1	4.32	328.2	1512.1	718.1
8	9%	75%	1782.5	7667.8	4.30	1589.0	5882.8	2067.2
$L_1 = 20, L_2 = 23, L = 50$								
5	1%	100%	74.5	582.4	7.82	71.6	540.7	25.9
6	0%	100%	173.4	1211.5	6.99	130.2	888.4	200.4
7	1%	89%	588.8	3187.6	5.41	413.8	2204.6	746.5
8	7%	66%	2305.3	17373.3	7.54	2240.7	14540.3	2120.1

Table 1. Comparative success rates of the work of [19] and the EA with $N = 20$ and L_1, L_2 and L as shown.

5.2 Analysis

The following are observations of the data presented on Tables 1–2. The EA approach solves the problem in the range $5 \leq n \leq 8$ for $L_1 \leq 40$, $L_2 \leq 43$ and $L \leq 50$, finding a secret word $\alpha' = q\alpha$ where $q \in C_G(b_i)$ for $i = 1, \ldots, N$. By

n	[19]	EA	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$	$\overline{g}_{.10}$	$\overline{t}_{.10}$	s_g
$L_1 = 40, L_2 = 43, L = 50$								
5	-	100%	103.0	1471.1	14.28	99.4	1354.7	34.4
6	0%	99%	222.3	4002.6	18.01	170.3	3004.7	281.8
7	0%	94%	820.4	14365.7	17.51	587.7	10542.9	978.2
8	1%	72%	2611.1	28128.0	10.77	2610.8	27141.4	1966.9

Table 2. Comparative success rates of the work of [19] and the EA with $N = 20$ and L_1, L_2 and L as shown. Values of $M = 3750$ were used for the $n = 7$ case, and $M = 7500$ for $n = 8$. Due to the mean time taken, fifty instances were computed with length cap 500 for the case $n = 8$. The ratio $\overline{t}/\overline{g}$ in this case is lower than expected due to the length cap.

Tables 1–2, the approach demonstrably outperforms the LBA algorithm of [19] in terms of success rate in every case. The success rate seems to depend most on the value of n, especially in terms of time complexity. Analysing the mean number of seconds taken to compute a single generation, it is clear that when the parameters L_1, L_2 and L are fixed, increasing the value of n produces only a small increase in the mean number of seconds taken to compute a single generation. For example, the mean time per generation varies between 0.70 and 1.30 seconds when $L_1 = 10, L_2 = 13$ and $L = 10$, and between 5.41 and 7.82 seconds when $L_1 = 20, L_2 = 23$ and $L = 50$. For more information, see Section 5.6.

With respect to computation time, assuming a constant value of $n \neq 5$, increasing L_1, L_2 and L increases the total time taken, because the generators of the subgroup $\langle a_1, \ldots, a_N \rangle$ are longer. Increasing L_1 and L_2 dramatically affects the time taken and the success rate generally decreases. Finally in the cases with $L = 50$, comparing the success rates of the EA with that of [19] provide a particularly stark conclusion. For the larger matrix dimensions on the larger experiments (e.g., $n = 8$ for $L = 50$), the EA tends to be slower but the success rate demonstrably outstrips that of the authors.

Increasing the number of slave processors

This is primarily a test of scalability of the method and illustrates the impact of a parallelised architecture upon EA time complexity. The cryptographic parameters were fixed to $L_1 = 10$, $L_2 = 13$, $L = 10$ and the EA was rerun on 100 random instances with 10 and then 20 slave processors. Each block in Table 3 gives the mean number of generations, mean time and the mean time per generation for each trial. The block for 5 slave processors contains the applicable numbers from

Table 1. Finally, the cases of $n = 5, 6$ and 7 were run with $M = 2500$ and the case of $n = 8$ was run with $M = 5000$.

# / n	5			10			20		
	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$
5	42.8	29.8	0.70	66.3	31.7	0.48	41.3	16.2	0.39
6	64.9	74.1	1.14	78.7	61.7	0.78	115.3	44.5	0.39
7	171.6	223.1	1.30	240.5	148.2	0.62	312.8	127.9	0.41
8	795.5	932.8	1.17	1312.0	802.7	0.62	1157.2	429.8	0.37
Mean	-	315.0	1.08	-	261.1	0.63	-	154.6	0.39

Table 3. The effect on EA time taken by varying the number of slave processors used.

By Table 3, increasing the number of slave processors scales the computation time well, with the mean time across the given values of n decreasing from 315.0 seconds (for 5 slave processors) to 154.6 seconds (for 20 slave processors). This gives a factor 2.04 speedup from multiplying the number of slave processors by four. A more realistic metric, perhaps, is that of average time per generation, which decreases from 1.08 seconds per generation (for 5 slave processors) to 0.39 seconds per generation (for 20 slave processors). For this metric, there is a speedup of 2.77. The work of [19] gave a one-hour time limit for solving an instance via the LBA method. It is believed that this does not apply to the present work, as, being inherently parallel, the EA processes many more words per iteration (generation), providing increased success rates over that of the authors. It is assumed that the LBA processes iterations more rapidly, being a non-parallel method comparing only two individuals at a time, and so the number of iterations performed by an LBA is assumed to be greater than that performed by the EA. In any case, since the parallelised EA method is scaleable, it is believed that those instances not solved within the prescribed time limit would likely be solved if the number of slave processors were increased. Given the inexpensive nature of real-world computer hardware and additional cores, by Table 3, this is a feasible objective.

It is acknowledged that there exist other methods of breaking the proposed protocol. Indeed, the attack described in the following section is proposed, which makes use of the natural representation of the Heisenberg groups in terms of upper-triangular matrices, allowing solution of the MCSP with methods of linear algebra. While this article was under review an independent solution for generalised Heisenberg groups as a platform was published [24].

5.3 A linear algebra attack for the Heisenberg groups

Using the notation of Section 5.1, for all $\mathbf{x}$, $\mathbf{y}$, $\mathbf{x}'$, $\mathbf{y}' \in \mathbb{Z}^n$, z, $z' \in \mathbb{Z}$, we have

$$M(\mathbf{x},\mathbf{y},z)\,M(\mathbf{x}',\mathbf{y}',z') = M(\mathbf{x}+\mathbf{x}',\mathbf{y}+\mathbf{y}',\mathbf{x}\cdot\mathbf{y}'+z+z'),$$

where $\mathbf{x}\cdot\mathbf{y}'$ is the standard scalar product of $\mathbf{x}$ and $\mathbf{y}'$, hence

$$M(\mathbf{x},\mathbf{y},z)^{-1} = M(-\mathbf{x},-\mathbf{y},\mathbf{x}\cdot\mathbf{y}-z)$$

and

$$M(\mathbf{x}',\mathbf{y}',z')^{-1}\,M(\mathbf{x},\mathbf{y},z)\,M(\mathbf{x}',\mathbf{y}',z') = M(\mathbf{x},\mathbf{y},\mathbf{x}\cdot\mathbf{y}'-\mathbf{x}'\cdot\mathbf{y}+z).$$

Note also that $M(\mathbf{0},\mathbf{0},0) = I_n$ is the $n\times n$ identity matrix. It is easily verified that the centre $Z(H_n)$ of the group H_n consists of the matrices $M(\mathbf{0},\mathbf{0},z)$ with $z\in\mathbb{Z}$, and that $H_n/Z(H_n)$ is isomorphic to $\mathbb{Z}^n\oplus\mathbb{Z}^n = \mathbb{Z}^{2n}$. In particular, H_n is polycyclic with Hirsch length $h(H_n) = 2n+1$.

A polycyclic presentation of H_n is obtained as follows. For $i = 1,\ldots,n$ let $\mathbf{e_i}$ be the i-th standard basis vector of $\mathbb{Z}^n$. Then the matrices

$$\begin{aligned} M_i &:= M(\mathbf{e_i},\mathbf{0},0), && i=1,\ldots,n,\\ M_{n+i} &:= M(\mathbf{0},\mathbf{e_i},0), && i=1,\ldots,n,\\ M_{2n+1} &:= M(\mathbf{0},\mathbf{0},1). \end{aligned}$$

form a generating set for H_n. For $1\le i\le j\le 2n+1$ we have

$$[M_i,M_j] = \begin{cases} M_{2n+1}, & \text{if } i+n=j<2n+1,\\ I_n, & \text{otherwise.}\end{cases}$$

Hence, H_n has the polycyclic presentation

$$\begin{aligned} H_n \cong \langle\, g_1,\ldots,g_{2n+1} \;\mid\; & [g_i,g_{n+i}] = g_{2n+1},\ [g_i,g_j]=1,\\ & i=1,\ldots,n,\ j=1,\ldots,2n+1,\ j\neq n+i\,\rangle. \end{aligned}$$

If H_n is given by this presentation, then the simultaneous conjugacy problem

$$c_i = b_i^a, \qquad i=1,\ldots,N$$

in H_n can be attacked by representing the words b_i and c_j in g_1, g_2, …, g_{2n+1} as the corresponding words in the matrices M_1, M_2, …, M_{2n+1} using the isomorphism constructed above and solving a system of linear equations over the integers.

In fact, for every permutation σ of $\{1,\dots,n\}$, the map

$$\begin{cases} g_i \mapsto g_{\sigma(i)}, & i=1,\dots,n,\\ g_{n+i} \mapsto g_{n+\sigma(i)}, & i=1,\dots,n,\\ g_{2n+1} \mapsto g_{2n+1} \end{cases}$$

defines an automorphism of this finitely presented group, which shows that, even if the numbering of the given generators is disguised, it is sufficient to identify the pairs of letters $\{g_1, g_{n+1}\}$, $\{g_2, g_{n+2}\}$, …, $\{g_n, g_{2n}\}$ in the given relators to set up the matrix representation.

Let the unknown word a and the given words b_i and c_j be represented by the matrices $M(\mathbf{p},\mathbf{q},r)$, $M(\mathbf{x}_i,\mathbf{y}_i,z_i)$ and $M(\mathbf{u}_j,\mathbf{v}_j,w_j)$, respectively. Then the formula expressing conjugation in H_n at the beginning of this section shows that the simultaneous conjugacy problem is equivalent to the system of linear equations

$$\begin{cases} \mathbf{x}_i = \mathbf{u}_i, & i=1,\dots,N,\\ \mathbf{y}_i = \mathbf{v}_i, & i=1,\dots,N,\\ \mathbf{x}_i\cdot\mathbf{q}-\mathbf{p}\cdot\mathbf{y}_i+z_i = w_i, & i=1,\dots,N \end{cases}$$

for $\mathbf{p}=(p_1,\dots,p_n)$, $\mathbf{q}=(q_1,\dots,q_n)\in\mathbb{Z}^n$. The first set of $2N$ equations are necessary conditions for the simultaneous conjugacy of b_i and c_i, which can readily be checked. The description of the centre of H_n shows that the unknown r in $M(\mathbf{p},\mathbf{q},r)$ cannot be determined and every word a which is represented by $M(\mathbf{p},\mathbf{q},r)$ for a solution $\mathbf{p}$, $\mathbf{q}$ of the above equations is a solution of the simultaneous conjugacy problem.

5.4 Polycyclic groups defined by a number field

The additive group $\mathcal{O}$ of the ring of integers of a number field K is a finitely generated free abelian group and its group of units U is a finitely generated abelian group. By Dirichlet's unit theorem, the rank of U is $s+t-1$, where s is the number of embeddings of K into $\mathbb{R}$ and t the number of conjugate pairs of embeddings of K into $\mathbb{C}$. Moreover, the torsion subgroup of U is a cyclic group of even order (cf., e.g., [17, Theorem 8.27]).

Let $K=\mathbb{Q}[x]/(f)$ for some monic irreducible polynomial $f\in\mathbb{Z}[x]$ of degree n. Then the Hirsch lengths of $\mathcal{O}$ and of U are $h(\mathcal{O})=n$ and $h(U)=s+t-1$, respectively. In [7] the non-nilpotent infinite polycyclic group $\mathcal{O}\rtimes U$ was proposed as a platform group for cryptosystems. The semidirect product is defined by using the right multiplication action of U on $\mathcal{O}$, and we have $h(\mathcal{O}\rtimes U)=n+s+t-1$.

A polycyclic presentation for $\mathcal{O} \rtimes U$ is obtained from a basis $\{b_1, \ldots, b_n\}$ of $\mathcal{O}$ and a finite generating set $\{u_1, \ldots, u_{s+t}\}$ of U by determining the coefficients in the expressions of $b_i u_j$ and $b_i u_j^{-1}$ in terms of the basis $\{b_1, \ldots, b_n\}$. Given f, the GAP package Polycyclic [8] provides such a polycyclic presentation for $\mathcal{O} \rtimes U$.

Length-based attacks have been documented in [11]. In [21] the multiple conjugacy search problem for $\mathcal{O} \rtimes U$ was attacked by considering $\mathcal{O} \rtimes U$ as a subgroup of $K \rtimes K^*$, where K^* is the multiplicative group of K, by representing the elements of K as matrices with respect to a basis of K over $\mathbb{Q}$, and by solving the system of linear equations corresponding to the given problem.

Experimental results

In this subsection, a direct comparison between the EA approach and that of [11] on the above problem was run. In the cited work, only the success rates are presented; the present work introduces further details on the performance. For each experiment (line in the following tables), one hundred instances were run, except where stated, with one trial of each. Independent experiments revealed that a value of $\varepsilon = 20$ and the cost tuple $(S_u(\alpha), S_{wt}(\alpha), \max_u(\alpha), W_u(\alpha), \min_u(\alpha), \ell(\alpha))$ gave the best EA performance and thus these are used. On Tables 4–5, the Hirsch lengths of the polynomials in the given order were respectively 1, 3, 4, 7, 10, 14 and 16. To reduce the space taken by each of the following tables, the polynomial is replaced by its degree d in the first place. Hence the polynomials $x - 1$, $x^2 - x - 1$, $x^3 - x - 1$, $x^5 - x^3 - 1$, $x^7 - x^3 - 1$, $x^9 - 7x^3 - 1$ and $x^{11} - x^3 - 1$ on Tables 4 and 5 are denoted by their degrees. Table 6 concentrates on degrees 7 and 11. Apart from that, Tables 4–6 have the same format as the tables in Section 5.1. All statistical measurements include those instances in which the EA was unsuccessful.

5.5 Analysis

Some key observations of the results on Tables 4–6 are as follows. First, the EA approach outperforms the LBA algorithm of [11]. For degrees 1, 2 and 3 the EA performs generally equally as well as the work of [11] as recorded in Table 4 and the upper section of Table 5. But for higher degrees this is clearly not the case: in Table 4 the EA outperforms the LBA at more than twice the success rate for degree 5 and approximately seven times the success rate for degree 7. In all tables in this subsection, it is generally the case that increasing the degree of the polynomial used results in a decrease in EA success rate. In the lower section of Table 5, the LBA of [11] only shows success on the two classes of experiments for degree 2 and 11. The EA manages to solve a high percentage of instances for each degree.

d	[11]	EA	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$	$\overline{g}_{.10}$	$\overline{t}_{.10}$	s_g
$L_1 = 10, L_2 = 13, L = 5$								
1	98%	100%	5.3	1.0	0.19	5.0	0.9	2.4
2	100%	100%	32.6	34.3	1.05	23.9	19.1	37.8
3	100%	97%	102.1	52.5	0.51	21.6	15.1	426.6
5	35%	76%	1363.5	1291.0	0.95	1076.7	814.7	2092.8
7	8%	54%	2485.3	3970.1	1.60	2474.3	2874.6	2360.0
9	5%	30%	5683.9	38989.9	6.86	6149.8	27795.0	2996.6
11	5%	20%	6225.1	11728.4	1.88	6805.0	9797.4	2657.3

Table 4. Comparative success rates of the work of [11] and the EA with $N = 20$ and L_1, L_2 and L as shown. The maximum number of iterations $M = 2500$ was used for degrees 1, 2 and 3, $M = 5000$ for degrees 5 and 7, and $M = 7500$ for degrees 9 and 11. These values were found by experimentation. For degree 9, fifty iterations were performed with length cap 1000. For degree 11 the same length cap was imposed.

The mean number of generations taken by the EA to solve the average experiment generally increases as the cryptographic parameters are increased (e.g., from 898.0 to 2504.5 for degree 5 as L_1 and L_2 are increased), as does the average time per generation. This is, intuitively, to be expected. The EA success rate correspondingly decreases. However, the mean number of generations taken is still relatively low compared to what may be expected from an LBA. This may be seen as an efficiency gain over the LBA which comes from the population-based approach and its inherent parallelism. Even for high values of cryptographic parameters exhibited in Table 6, the EA reports successes which the LBA did not, particularly in the case of degree 7.

In the unsuccessful experiments, the EA generally times out for two reasons. First, the initial cost may be very high (e.g., with costs of the order of 10^8 to 10^{10} for degree 9 on the bottom section of Table 5) and, although the evolution progresses well to find large reductions in cost, the sum of these reductions is insufficient for the EA to find an exact solution to the instance. Second, the EA is trapped in a succession of local minima that the parameter perturbation strategy cannot overcome.

Commenting upon the mean time per generation for each degree, it is clear these vary somewhat. For example, Tables 4–5 relate that this measurement broadly increases with degree, with the pattern only being broken by experiments for degree 9. Further, the average problem for the case of degree 9 seems intuitively much

d	[11]	EA	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$	$\overline{g}_{.10}$	$\overline{t}_{.10}$	s_g
$L_1 = 5, L_2 = 8, L = 5$								
1	98%	100%	5.2	0.3	0.06	4.8	0.3	2.6
2	98%	100%	16.0	2.0	0.13	11.8	1.4	23.1
3	95%	94%	172.5	14.4	0.08	14.9	1.1	594.4
5		84%	898.0	176.4	0.20	496.4	77.8	1821.3
7		72%	1672.8	471.1	0.28	1464.8	323.1	2231.5
9		54%	3912.7	13503.0	3.45	3952.3	5544.2	3508.6
11	59%	61%	3140.6	1301.9	0.41	2987.2	696.6	3546.8
$L_1 = 15, L_2 = 18, L = 5$								
1		100%	5.6	0.8	0.14	4.8	0.7	4.2
2	96%	100%	58.1	21.4	0.37	34.9	7.9	92.6
3		90%	298.8	65.0	0.22	59.9	18.7	742.5
5		56%	2504.5	1770.5	0.71	2502.5	964.5	2327.5
7		44%	2971.6	3129.1	1.05	3086.2	2338.0	2325.2
9		32%	5465.2	16368.1	2.99	5884.3	11489.0	3099.7
11	53%	26%	5908.8	5046.9	0.85	6429.8	3616.2	2886.0

Table 5. Comparative success rates of the work of [11] and the EA with $N = 5$ and L_1, L_2 and L as shown. Blank spaces are shown where no success rates were provided by the authors. Values of M are as in Table 4. The length cap used was 1000 for $d = 9$ (upper and lower sections) and for $d = 5, 7$ and 11 in the lower section of the table. Fifty iterations were performed for $d = 9$ (upper and lower sections).

d	[11]	EA	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$	$\overline{g}_{.10}$	$\overline{t}_{.10}$	s_g
$L_1 = 10, L_2 = 13, L = 10$								
7	2%	40%	3236.9	7604.4	2.35	3408.5	4246.2	2222.1
11	0%	10%	9338.2	54034.2	5.79	10001.0	27417.5	2332.8

Table 6. Comparative success rates of the work of [11] and the EA with $N = 20$ and L_1, L_2 and L as shown. The values of M used are respectively 5000 for degree 7 and 10000 for degree 11. A length cap of 500 for degree 7 and 2000 for degree 11 was used, with fifty instances run on degree 11.

harder than that for degree 11 in terms of mean time taken, success rate and the mean number of generations.

Next, the number of slave processors used on the first four cases of Table 4 is increased, in order to quantify the time scalability of the algorithm.

Increasing the number of slave processors

# / d	5			10			20		
	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$	$\overline{g}$	$\overline{t}$	$\overline{t}/\overline{g}$
1	5.3	1.0	0.19	5.2	0.8	0.15	5.4	0.7	0.13
2	32.6	34.3	1.05	32.7	14.5	0.44	25.8	6.9	0.27
3	102.1	52.5	0.51	113.6	35.0	0.31	139.8	29.1	0.21
5	1363.5	1291.0	0.95	1006.3	485.4	0.48	1358.0	449.9	0.33
Mean	-	344.7	0.68	-	133.9	0.35	-	121.7	0.24

Table 7. Effect on time complexity of varying the number of slave processors used. The cryptographic parameters $N = 20$, $L_1 = 10$, $L_2 = 13$ and $L = 5$ were used.

The EA was rerun on one hundred random instances for the degrees shown with 10 and then 20 slave processors. Each block in Table 7 gives the mean number of generations, the mean time and the mean time per generation for each trial. The columns for 5 slave processors contain the applicable numbers from Table 4. Observe that increasing the number of slave processors again appropriately scales the computation time, with the average time across the given values of n decreasing from 344.7 seconds (for 5 slave processors) to 121.7 seconds (for 20 slave processors). This gives a factor 2.83 speedup from multiplying the number of slave processors by four, in comparison with the factor 2.04 speedup observed for the Heisenberg case, indicating that algorithm time complexity scales more with the number of slave processors used. Calculating the mean number of seconds per generation over all degrees d gives a decrease of 0.68 seconds (for 5 slave processors) to 0.24 seconds (for 20 slave processors), a factor 2.83 speedup (compared with 2.77 for the Heisenberg case). The next subsection analyses this further.

5.6 Algorithm time complexity

Some observations on general EA time complexity are given below, according to each given platform group.

Polycyclic groups defined by a number field

For the group defined by the polynomial $p = x^9 - 7x^3 - 1$ it was noted that, during the EA runs, the word lengths involved in the computation became disproportionately long compared to the solution length, resulting in a decrease in operation speed. Hence, the growth of words during normal form computation may represent a more reliable indicator of MCSP hardness over the given groups than Hirsch length as proposed in [11]. (A length cap is used in the implementation described in Section 4.4 to counteract this growth of words.)

For a polycyclic presentation of the form

$$G \cong \langle\, g_1, \ldots, g_n \;\mid\; g_i^{r_i} = w_{i,i} \quad \text{for } i = 1, \ldots, n \text{ such that } r_i \neq \infty,$$
$$g_k^{-1} g_j g_k = w_{j,k},\ g_k g_j g_k^{-1} = w_{k,j} \quad \text{for } 1 \leq k < j \leq n \,\rangle,$$

where each $w_{i,j}$ is a word in the generators g_k of the form $g_{\min(i,j)}^{e_{\min(i,j)}} \cdots g_{n-1}^{e_{n-1}} g_n^{e_n}$ with $e_k \in \mathbb{Z}$ and $0 \leq e_k < r_k$, we associate to the relator $g_i^{r_i} = w_{i,i}$ the length $r_i + \ell(w_{i,i})$, to the relator $g_k^{-1} g_j g_k = w_{j,k}$ the length $3 + \ell(w_{j,k})$ and to the relator $g_k g_j g_k^{-1} = w_{k,j}$ the length $3 + \ell(w_{k,j})$. The complexity of the given polycyclic presentation is then the sum of these lengths[1]. Table 8 gives the lengths of the relators in the presentation in each group and the complexities. Where applicable, subscript notation refers to a sequence of repeated numbers. For example, 0_5 should be read as the subsequence of five zeros $[0, 0, 0, 0, 0]$.

$h(G)$	Relator Length Vector	Complexity $K(G)$
1	[2,4]	6
3	[2,0,13_2]	28
4	[2,0,13,12,13]	40
7	[2,0_2,25,24,25,24_2]	124
10	[2,0_3,39,38,37,36_3,38]	262
14	[2,0_5,901,634,502,499,718,950,1115,1702,2241]	9264
16	[2,0_5,71,69_2,66,65_2,66_2,68,69,72]	748

Table 8. The lengths of the defining relators of each group, and the relator length sums (complexities).

As the Hirsch length $h(G)$ increases, the complexity $K(G)$ does not necessarily increase commensurately. It is argued that the complexities given above provide a

[1] Alternatively, the increase in word length in an elementary reduction step using the given relators could be measured as $\ell(w_{i,i}) - r_i$, $\ell(w_{j,k}) - 3$ and $\ell(w_{k,j}) - 3$, respectively.

more representative measure of average problem hardness than the Hirsch length for the MCSP in polycyclic groups defined by a number field. This is borne out by the results of Tables 4–6 which show that, by far, the problem for the group generated by the polynomial $p = x^9 - 7x^3 - 1$ is intuitively the hardest. In particular, for the settings $L_1 = 5$, $L_2 = 8$ in Table 5, the average number of seconds per generation is 3.45 for degree 9 and 0.41 for degree 11. This is also replicated in the remaining experiments shown. Practically it is observed that for polycyclic presentations of high complexity the EA operations produce words of large differences in cost.

Thus it is clear that algorithm time complexity is controlled by the complexity of the polycyclic presentation as well as the cryptographic parameters N, L, L_1, L_2 of the MCSP instance (see Section 5.5 for the dependence of time complexity on these). While some complexity estimates for normal form computations in polycyclic groups have been found (see, e.g., [15]), complexity analysis of EAs is still a challenge [26]. Therefore, providing a concrete expression for time complexity is expected to be an onerous task.

Heisenberg groups

Analogously, there were some findings in this case. Assuming a Heisenberg group H_{2n+1} for a given value of n, the relator length vector was $[0_n, 12_n, 0]$. This gives a relator sum complexity of $K(H_{2n+1}) = 12n$, exhibiting linear behaviour in n. Noting the work of [19] which relates the Hirsch lengths $h(H_{2n+1}) = 2n + 1$, which are also linear, it is clear that there is no advantage in this case using either Hirsch length or relator sum complexity to define the problem hardness.

6 Concluding remarks

To begin with, key contributions of the present work are recalled. First, the proposed EA approach has been shown to be successful across a range of cryptographic parameters, and particularly on the protocol for Heisenberg groups (Section 5.1). The EA approach demonstrably outperforms the LBA attacks suggested in [11, 19], and, to the best of the knowledge of the authors, marks the first work using this method to attack either of the proposed protocols.

It has also been shown that, in the case of polycyclic groups generated by a number field, Hirsch length may not be the best measure of problem hardness with respect to the group, suggesting that a more appropriate measure is the relator sum complexity $K(G)$. Conversely, there is a direct linear relationship between the relator sum complexity and Hirsch length for the Heisenberg groups. Finally, it was

shown that the EA attack on the Heisenberg group protocol inspires a deterministic linear algebra attack (Section 5.3).

The EA approach uses the now commonly-available multicore processor architectures to work in parallel. It also, being population-based, enables sampling and search across a large search space, in distinction to the LBA (which may require a very large number of words in the search space to be sampled in order to find a solution). This means that the EA reveals an increased amount of structural information in comparison to an LBA, enabling more detail about the core problem (the MSCP over the platform group) to be known with comparatively fewer runs.

It is believed that the EA approach is scalable (of which some evidence has been shown in Tables 3 and 7) and efficient (in terms of numbers of candidate solutions sampled), the conclusions in the present work strongly suggesting that this attack may be ported to GPU architectures resulting in large speedups whilst retaining the flexibility of the approach. The approach is also readily adaptable to other platform groups, and the approach features innovations such as adaptive parameter perturbation which effectively adapts the algorithm in real time. In addition, it is believed the statistics shown may be improved through a more judicious use of control parameters.

Despite the above, it is acknowledged that the EA approach is not perfect. The approach struggles where the initial guesses have very high cost (see Section 5.5). This situation is encountered in cases of high relator sum complexity (for example, for degrees 9 and 11 in the number field case). Solution of such instances often proves difficult because relator words with high length are rarely applicable for cost reduction. A complexity analysis of the EA is also more difficult than for an LBA, due to the algorithm behaviour more closely resembling a complex system. Tuning settings suitable for high performance is also non-trivial.

Further work will be performed on approaches to minimise the effects of high relator sum complexity groups, which may involve hybridising the EA with a local search algorithm in order to make high initial costs (at the beginning of a run) less likely. Through experiment it has been observed that high initial cost tends to result in a larger number of local cost minima encountered by the EA search during runs; some practical analysis of these local minima (or “peaks” as in [22]) would be fruitful. In addition, speed improvements are expected by the introduction of a mechanism which stores the normal forms of the words previously computed.

The above advantages and results of the EA approach show that pursuing evolutionary (and Monte-Carlo) methods in combinatorial group theory and group theoretic cryptology is a valid concept, and promises much in the way of fast, robust and effective stochastic methods for solving algebraic problems in general.

Acknowledgments. The Centre for Mathematical Sciences at Plymouth University is gratefully acknowledged for its generous research support and encouragement. The authors also gratefully acknowledge the kind comments of the anonymous referees.

Bibliography

[1] I. Anshel, M. Anshel and D. Goldfeld, An Algebraic Method for Public-Key Cryptography, *Math. Res. Lett.* 6 (1999), 287-291.

[2] I. Anshel, M. Anshel, B. Fisher and D. Goldfeld, New Key Agreement Protocols, in: D. Naccache (ed.), *Topics in Cryptology, CT-RSA 2001*, LNCS 2020, Springer (2001), 13-27.

[3] R. F. Booth, D. Yu. Bormotov and A. V. Borovik, Genetic Algorithms and Equations in Free Groups and Semigroups, *Contemp. Math.* 349 (2004), 63-80.

[4] G. Cooperman, ParGAP, Version 1.4.0, available from `http://www.gap-system.org/Packages/pargap.html` (2013).

[5] M. J. Craven, An Evolutionary Algorithm for the Solution of Two-Variable Word Equations in Partially Commutative Groups, *Studies in Comp. Intell.* 153, Springer (2008), 3-19.

[6] M. J. Craven and H. C. Jimbo, An Evolutionary Algorithm Solution of the Multiple Conjugacy Search Problem in Partially Commutative Groups with Applications, *Groups, Complexity and Cryptology* 4 (2012), 135-165.

[7] B. Eick and D. Kahrobaei, Polycyclic Groups: A New Platform for Cryptology?, Preprint, 2004, (`http://arxiv.org/abs/math/0411077`).

[8] B. Eick, W. Nickel and M. Horn, Polycyclic, Version 2.1.1, available from `http://www.gap-system.org/Packages/polycyclic.html` (2013).

[9] N. Franco and J. González-Meneses, Conjugacy Problem for Braid Groups and Garside groups, *J. Algebra* 266 (1) (2003), 112-132.

[10] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.7.7*; 2015, (`http://www.gap-system.org`).

[11] D. Garber, D. Kahrobaei and H. T. Lam, Length-Based Attacks in Polycyclic Groups, *J. Math. Crypt.* 9 (1) (2015), 33-43.

[12] D. Garber, S. Kaplan, M. Teicher, B. Tsaban and U. Vishne, Length-Based Conjugacy Search in the Braid Group, in: L. Gerritzen, D. Goldfeld, M. Kreuzer, G. Rosenberger, V. Shpilrain (eds.), *Algebraic Methods in Cryptography, Contemp. Math.* 418, American Mathematical Society, Providence (2006), 75-88.

[13] D. Garber, S. Kaplan, M. Teicher, B. Tsaban and U. Vishne, Probabilistic Solutions of Equations in the Braid Group, *Adv. Appl. Math.* 35 (2005), 323-334.

[14] F. A. Garside, The Braid Group and Other Groups, *Quart. J. Math. Oxford* 20 (78) (1969), 235-254.

[15] V. Gebhardt, Efficient Collection in Infinite Polycyclic Groups, *J. Symb. Comp.* 34 (2002), 213-228.

[16] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley (1989).

[17] D. F. Holt, B. Eick and E. A. O'Brien, Handbook of computational group theory, Chapman & Hall CRC (2005).

[18] J. Hughes and A. Tannenbaum, Length-Based Attacks for Certain Group Based Encryption Rewriting Systems, *Workshop SECI02* (2002), Tunis, Tunisia.

[19] D. Kahrobaei and H. T. Lam, Heisenberg Groups as Platform for the AAG Key-Exchange Protocol, *22nd IEEE Intern. Conf. on Network Protocols (ICNP)* (2014), 660-664.

[20] K. Ko, S. Lee, J. Cheon, J. Han, J. Kang and C. Park, New Public-Key Cryptosystem Using Braid Groups, in: M. Bellare (ed.), *CRYPTO 2000*, LNCS 1880, Springer (2000), 166-183.

[21] M. Kotov and A. Ushakov, Analysis of a Certain Polycyclic Group-Based Cryptosystem, *J. Math. Crypt.* 9 (2015), 161-167.

[22] A. D. Myasnikov and A. Ushakov, Length Based Attack and Braid Groups: Cryptanalysis of Anshel-Anshel-Goldfeld Key Exchange Protocol, in: T. Okamoto and X. Wang (eds.), *Public Key Cryptography*, LNCS 4450, Springer (2007), 76-88.

[23] A. G. Myasnikov and A. Ushakov, Random Subgroups and Analysis of the Length-Based and Quotient Attacks, *J. Math. Crypt* 2 (1) (2008), 29-61.

[24] A. Nikolaev and K. R. Blaney, A PTIME Solution to the Restricted Conjugacy Problem in Generalized Heisenberg Groups, *Gr. Complex. Crypt.* 8 (1) (2016), 69-74.

[25] D. Ruinskiy, A. Shamir and B. Tsaban, Length-Based Cryptanalysis: The Case of Thompson's Group, *J. Math. Crypt.* 1 (2007), 359-372.

[26] D. Sudholt, Parallel Evolutionary Algorithms, in: J. Kacprzyk and W. Pedrycz (eds.), *Springer Handbook of Computational Intelligence*, Springer (2015), 929-959.

Received **???**.

Author information

Matthew J. Craven, Centre for Mathematical Sciences, Plymouth University,
Drake Circus, Plymouth, PL4 8AA, United Kingdom.
E-mail: `matthew.craven@plymouth.ac.uk`

Daniel Robertz, Centre for Mathematical Sciences, Plymouth University,
Drake Circus, Plymouth, PL4 8AA, United Kingdom.
E-mail: daniel.robertz@plymouth.ac.uk