Interactive Proofs with Polynomial-Time Quantum Prover for Computing the Order of Solvable Groups

François Le Gall[1]
Graduate School of Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan

Tomoyuki Morimae[2]
Yukawa Institute for Theoretical Physics, Kyoto University
Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto 606-8502, Japan

Harumichi Nishimura[3]
Graduate School of Informatics, Nagoya University
Chikusa-ku, Nagoya, Aichi 464-8601, Japan

Yuki Takeuchi[4]
NTT Communication Science Laboratories, NTT Corporation
3-1 Morinosato-Wakamiya, Atsugi, Kanagawa 243-0198, Japan

Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan

Abstract

In this paper we consider what can be computed by a user interacting with a potentially malicious server, when the server performs polynomial-time quantum computation but the user can only perform polynomial-time classical (i.e., non-quantum) computation. Understanding the computational power of this model, which corresponds to polynomial-time quantum computation that can be efficiently verified classically, is a well-known open problem in quantum computing. Our result shows that computing the order of a solvable group, which is one of the most general problems for which quantum computing exhibits an exponential speed-up with respect to classical computing, can be realized in this model.

2012 ACM Subject Classification Theory of computation → Quantum computation theory

Keywords and phrases Quantum computing, interactive proofs, group-theoretic problems

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.26

1 Introduction

First-generation quantum computers will be implemented in the "cloud" style, since only few groups, such as governments or huge companies, will be able to possess such expensive and high-maintenance machines. In fact, IBM has recently opened their 16-qubit machine for a cloud service [34]. In a future when many companies provide their own quantum cloud computing services, a malicious company might emerge who is trying to palm a user off with

[1] Partially supported by the JSPS KAKENHI grants No. 15H01677, No. 16H01705 and No. 16H05853.
[2] Supported by JST PRESTO No. JPMJPR176A, and the Grant-in-Aid for Young Scientists (B) No. JP17K12637 of JSPS.
[3] Partially supported by the JSPS KAKENHI grants No. 26247016, No. 16H01705 and No. 16K00015.
[4] Supported by the Program for Leading Graduate Schools: Interactive Materials Science Cadet Program.

© François Le Gall, Tomoyuki Morimae, Harumichi Nishimura, and Yuki Takeuchi;
licensed under Creative Commons License CC-BY
43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018).
Editors: Igor Potapov, Paul Spirakis, and James Worrell; Article No. 26; pp. 26:1–26:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a wrong result forged from their fake quantum computer. In addition, even if a fortunate user is interacting with a honest server, some noises in the server's gate operations might change the result. How can a user verify the correctness of the server's quantum computation? If the user has his/her own quantum computer, the user can of course check the server's result, but in this case the user may not need the cloud service in the first place. If the solution of the problem is easily verifiable (e.g., integer factoring), the user can naturally verify the correctness of the server's result, but many problems considered in quantum computing are not believed to have this property. Verifying classically and efficiently a server's quantum computation is indeed in general highly nontrivial.

It is known that if at least two servers, who are entangled but not communicating with each other, are allowed, then any problem solvable in quantum polynomial time can be verified by a classical polynomial-time user who exchanges classical messages with the servers [20, 24, 27]. However, the assumption that servers are not communicating with each other is somehow unrealistic: how can the user guarantee that remote servers are not communicating with each other?

Whether the number of the servers can be reduced to one is a well-known open problem [4]. For certain computational problems solvable in quantum polynomial time, it is known that this can be done. Simon's problem [31] and factoring [30] are trivial examples, since the answer can be directly checked in classical polynomial time. It is known that recursive Fourier sampling [10], which was the first problem that separates efficient quantum and classical computing, can be verified by a polynomial number of message exchanges with a single quantum server [23]. Moreover, it was shown that certain promise problems related to quantum circuits in the second level of the Fourier hierarchy [29] are verifiable by a classical polynomial-time user interacting with a single quantum server who sends only a single message to the user [12, 26].

Our results. In this paper we consider the problem of computing the order, i.e., the number of elements, of a finite group given as a black-box group (the concept of black-box groups is defined in Section 2). This problem is central in computational group theory, especially since the ability of computing the order makes possible to decide membership in subgroups. This problem has also been the subject of several investigations in computational complexity [1, 6, 7, 9, 32, 33]. The seminal result by Babai [6], especially, which put this problem in the complexity class AM, has been one of the fundamental motivations behind the concept of interactive proofs. Note that this is clearly a hard problem for classical computation: it is easy to show that no polynomial-time classical algorithm exists in the black-box setting, even if the input is an abelian group [9].

Most of the known quantum algorithms that achieve exponential speedups with respect to the best known classical algorithms are for group-theoretic problems, and especially problems over abelian groups. Shor's algorithm for factoring [30], for instance, actually computes the order of a cyclic black-box group. Watrous has shown that the group order problem can be solved in quantum polynomial time when the input group is solvable [33]. Since the class of solvable groups, defined in Section 2, is a large[5] class of finite groups that includes all

[5] It is known (see for instance [11]) that

$$\lim_{m\to\infty} \frac{\log \mathcal{G}_s(m)}{\log \mathcal{G}(m)} = 1,$$

where $\mathcal{G}(m)$ denotes the number of finite groups of order at most m and $\mathcal{G}_s(m)$ denotes the number of finite solvable groups of order at most m. It is even conjectured that the quotient $\mathcal{G}_s(m)/\mathcal{G}(m)$ goes to 1 when m goes to infinity, i.e., most finite groups are solvable.

abelian groups, this result significantly generalized Shor's algorithm. Watrous' algorithm can actually be seen as one of the most general results achieving an exponential speedup with respect to classical computation.

In this paper we show that the group order problem over solvable groups is also verifiable with a single server. More formally, in Section 2, where we introduce the relevant model of interactive protocols, we will introduce the notation $\mathrm{IP}[k, \mathrm{qpoly}]$ to denote the class of computational problems that are verifiable by a classical polynomial-time user interacting in k messages with a server who works in quantum polynomial time. Our main result is as follows.

▶ **Theorem 1.** *The solvable group order problem is in the complexity class* $\mathrm{IP}[3, \mathrm{qpoly}]$*. Moreover, if the set of prime factors of the order is also given as input, then the solvable group order problem is in* $\mathrm{IP}[2, \mathrm{qpoly}]$*.*

This result shows that for this important computational problem, the number of servers can be reduced to one as well, using a small number of messages. Note that assuming, in the second part of Theorem 1, that the set of prime factors of the order is known corresponds to several practical situations. An important example is computing the order of p-groups[6] with p known, which cannot be done in polynomial time in the classical setting [9]. The main open question is whether the number of messages can also be reduced to 2 without any assumption on the prime factors.

Other related works. In addition to the introduction of multiple servers mentioned above, there are other approaches considered in the literature for constructing verification systems for quantum computation.

First, if the user is allowed to be "slightly quantum", any problem solvable in quantum polynomial time can be efficiently verified with a single quantum server. For example, Refs. [2, 14] assume that the user can generate randomly-rotated single-qubit states, and Refs. [13, 16, 25] assume that the user can measure single-qubit states.

Second, since the class BQP (the class of decision problems that can be solved in quantum polynomial-time) is trivially in PSPACE and PSPACE = IP [21, 28], any problem in BQP can be classically verified using generic interactive proof protocols for PSPACE. In such protocols, however, the server has unbounded computational power. A tempting approach is to try to specialize these generic protocols to the class BQP, with the hope that the server's necessary computational power may be reduced. Ref. [3] made an significant first step in this direction.

Finally, it has been shown very recently that assuming that the learning with errors problem is intractable for polynomial-time quantum computation, any problem solvable in quantum polynomial time can be efficiently verified with a single quantum server and a single classical user [22].

2 Preliminaries

In this paper we assume that the reader is familiar with the standard notions of group theory (we refer to, e.g., [18] for a good introduction). All the groups considered will be finite. Given

[6] A (finite) p-group, where p is a prime, is a group of order p^r for some integer $r \geq 1$. A basic result from group theory shows that any p-group is solvable.

a group G, we use $|G|$ to denote its order (i.e., the number of elements in G), and use e to denote its identity element. Given elements $g_1, \ldots, g_r \in G$, we denote $\langle g_1, \ldots, g_r \rangle$ the subgroup of G generated by $g_1, \ldots, g_r$.

Black-box groups. We now describe the model of black-box groups. This concept, in which each group element is represented by a string and each group operation is implemented using an oracle, was first introduced by Babai and Szemerédi [9] to describe group-theoretic algorithms in the most general way, without having to concretely specify how the elements are represented and how groups operations are implemented. Indeed, any efficient algorithm in the black-box group model gives rise to an efficient concrete algorithm whenever the oracle operations can be replaced by efficient procedures. Especially, performing group operations can be done directly on the elements in polynomial time for many natural groups, including permutation groups and matrix groups where the group elements are represented by permutations and matrices, respectively. In the quantum setting, black-box groups have first been considered by Ivanyos et al. [19] and Watrous [32, 33].

A black-box group is a representation of a group G where each element of G is uniquely encoded by a binary string of a fixed length n, which is called the encoding length. The encoding length n is known. In order to be able to express the complexity of black-box group algorithms in terms of the group order $|G|$, and not in terms of the encoding length, we make the standard assumption that $n = O(\log |G|)$. Oracles are available to perform group operations. More precisely, two oracles are available. A first oracle performs the group product: given two strings representing two group elements g and h, the oracle outputs the string representing gh. The second oracle performs inversion: given a string representing an element $g \in G$, the oracle outputs the string representing the element g^{-1}. Note that the two oracles may behave arbitrarily on strings not corresponding to elements in G; this is not a problem since our protocols will never use the oracles on such strings. We say that a group G is input as a black-box if a set of strings representing generators $\{g_1, \ldots, g_s\}$ of G with $s = O(\log |G|)$ is given as input and queries to the oracles can be done at cost 1.[7] The input length is thus $sn = \mathrm{poly}(\log |G|)$.

To be able to take advantage of the power of quantum computation when dealing with black-box groups, the oracles performing the group operations have to be able to deal with quantum superpositions. Concretely, this is done as follows (see [19, 32, 33]). Let $s\colon G \to \{0,1\}^n$ denote the encoding of elements as binary strings. We assume that a quantum oracle V_G is available, such that $V_G(|s(g)\rangle|s(h)\rangle) = |s(g)\rangle|s(gh)\rangle$ for any two elements $g, h \in G$, and behaving in an arbitrary way on other inputs (i.e., strings not in $s(G)$). Another quantum oracle V'_G is also available, such that $V'_G(|s(g)\rangle|s(h)\rangle) = |s(g)\rangle|s(g^{-1}h)\rangle$ for any $g, h \in G$ and again behaving in an arbitrary way on other inputs.

Approximate sampling in black-box groups. Babai [5] proved the following result for general groups, which shows that elements of a black-box group can be efficiently sampled nearly uniformly.

▶ **Theorem 2.** *([5]) Let G be a black-box group. For any $\varepsilon > 0$, there exists a classical randomized algorithm running in time polynomial in $\log(|G|)$ and $\log(1/\varepsilon)$ that outputs an element of G such that each $g \in G$ is output with probability in range $(1/|G| - \varepsilon, 1/|G| + \varepsilon)$.*

[7] The assumption $s = O(\log |G|)$ is standard. Indeed, every group G has a generating set of size $O(\log |G|)$. Additionally, a set of generators of any size can be converted efficiently into a set of generators of size $O(\log |G|)$ by taking random products of elements [5].

Solvable groups. Before discussing solvable groups, let us introduce the following concept of polycyclic generating sequences (see [17] for details).

▶ **Definition 3.** Let G be a group. A polycyclic generating sequence of G is a sequence $(h_1, \ldots, h_t)$ of t elements from G, for some integer t, such that:
1. $\langle h_1, \ldots, h_t \rangle = G$;
2. for each $1 < j \leq t$, the subgroup $\langle h_1, \ldots, h_{j-1} \rangle$ is normal in $\langle h_1, \ldots, h_j \rangle$.

There are many equivalent definitions of solvable groups in the literature (see, e.g., [17] for a thorough discussion). In this paper we will use the following characterization: a finite group is solvable if and only if it has a polycyclic generating sequence. This characterization, which was already used by Watrous [33], is the most convenient for our purpose. As discussed in [33], for any finite solvable group G given as a black box, a polycyclic generating sequence $(h_1, \ldots, h_t)$ with $t = O(\log |G|)$ can be computed classically in polynomial time with high probability using for instance the randomized algorithm by Babai et al. [8].

Watrous showed that the order of a solvable black-box group can be computed in polynomial time in the quantum setting. We state this result in the following theorem.

▶ **Theorem 4.** *([33]) Let G be a solvable group given as a black-box group. There exists a quantum algorithm running in time* $\mathrm{poly}(\log |G|)$ *that outputs $|G|$ with probability at least* $1 - 1/\mathrm{poly}(|G|)$.

Let G be a solvable group and $(h_1, \ldots, h_t)$ be a polycyclic generating sequence of G. In the following we will write $H_j = \langle h_1, \ldots, h_j \rangle$ for each $j \in \{1, \ldots, t\}$, and for convenience write $H_0 = \{e\}$. Since H_j is obtained from H_{j-1} by adding one generator, the factor group H_j/H_{j-1} is cyclic. Let us write its order m_j. Note that the order of G is thus the product $m_1 m_2 \cdots m_t$. A fundamental (and easy to show) property of polycyclic generating sequences is the following: For any $j \in \{1, \ldots, t\}$, any element $h \in H_j$ can be written, in a unique way, as $h = h_1^{a_1} h_2^{a_2} \cdots h_j^{a_j}$ with integers $a_i \in \{0, 1, \ldots, m_i - 1\}$ for $i \in \{1, \ldots, j\}$. We call this sequence $(a_1, \ldots, a_j)$ the decomposition of h over H_j. Watrous [33] showed that the decomposition of any element can be computed efficiently in the quantum setting, which immediately leads to an efficient algorithm for membership testing in the subgroups H_j. We state these two results, separately, in the following theorem.

▶ **Theorem 5.** *([33]) Let G be a solvable group given as a black-box group and let $(h_1, \ldots, h_t)$ be a polycyclic generating sequence of G with $t = O(\log |G|)$. There exist two quantum algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ running in time polynomial in* $\log |G|$ *as follows.*
- *Algorithm $\mathcal{A}_1$ receives an integer $j \in \{1, \ldots t\}$ and an element $h \in H_j$, and outputs with probability at least* $1 - 1/\mathrm{poly}(|G|)$ *the decomposition of h over H_j.*
- *Algorithm $\mathcal{A}_2$ receives an integer $j \in \{1, \ldots t\}$ and an element $h \in G$, and decides whether $h \in H_j$ or not. The decision is correct with probability at least* $1 - 1/\mathrm{poly}(|G|)$.

Interactive proofs with efficient quantum prover. Interactive proof systems are typically described as protocols for decision problems. In this paper it will be more convenient to consider interactive proofs for computing functions, since we are interested in computing the order of the input group.[8] The definition we give below is inspired by [15].

[8] In order to be completely rigorous, we should actually define this concept for functional problems where the input is represented using oracles (since we are dealing with black-box groups where the group operation is represented by oracles). We nevertheless omit this purely technical point in the exposition.

Let $f : X \to \{0,1\}^*$ be a function, where X is a finite set. We consider protocols between a prover and a verifier, who both receives as input an element $x \in X$ and can exchange classical messages of polynomial length. At the end of the protocol, the verifier outputs either some $y \in \{0,1\}^*$ or one special element $\perp$. We say that the function f has a *k-message polynomial-time interactive proof* if there exists a k-message protocol in which the verifier works in classical polynomial time, such that the following properties hold:

1. (completeness) there is a prover P such that the verifier's output y satisfies $y = f(x)$ with probability at least $2/3$ when interacting with P;
2. (soundness) for any prover P', the verifier's output y satisfies $y \in \{f(x), \perp\}$ with probability at least $2/3$ when interacting with P'.

The prover P in the completeness condition is called the *honest prover*.

The above definition makes no assumption on the computational powers of the provers. Our main definition is obtained by restricting the computational power of the *honest* prover, i.e., the prover P in the completeness condition.

▶ **Definition 6.** A function f is in the class $\mathrm{IP}[k, \mathrm{qpoly}]$ if it has a k-message polynomial-time interactive proof where the honest prover P works in quantum polynomial time.

The notation $\mathrm{IP}[k, \mathrm{qpoly}]$ comes from its definition as a k-message interactive protocol with a prover working in quantum polynomial time (when honest). We stress that in Definition 6 there is no assumption on the computational power of P' for the soundness.

3 2-Message Protocol with Known Prime Factors

In this section we assume that the prime factors of the order of the black-box group G are known. We present a 2-message protocol in this case, which proves the second part of Theorem 1.

3.1 Preliminaries

We will need the following result in our protocol. This result essentially shows how to reduce the computation of the order of a solvable group G to the problem of deciding if its factor groups H_i/H_{i-1} have order 1 or not.

▶ **Theorem 7.** *Let G be a solvable group given as a black-box group. Let $p_1, \ldots, p_\ell$ denote the prime factors of $|G|$ and assume that the set $S = \{p_1, \ldots, p_\ell\}$ is also given as input. There exists a classical algorithm running in time polynomial in $\log |G|$ that outputs elements $h_1, \ldots, h_t \in G$, with $t = \mathrm{poly}(\log |G|)$, and t prime numbers $r_1, \ldots, r_t \in S$ such that, with probability at least $1 - 1/\mathrm{poly}(|G|)$, the following conditions hold:*

- *$(h_1, \ldots, h_t)$ is a polycyclic generating sequence of G;*
- *the order of H_i/H_{i-1} is either 1 or r_i for each $1 \le i \le t$, where we denote $H_i = \langle h_1, \ldots, h_i \rangle$ for $1 \le i \le t$ and $H_0 = \{e\}$.*

Before proving Theorem 7, let us discuss the main idea of the algorithm in this theorem. The approach is to start with an arbitrary polycyclic generating sequence and refine it by replacing each element by decreasing powers of it. Consider for instance the cyclic group of order 12, for which we have $\ell = 2$, $p_1 = 2$, $p_2 = 3$ and $|G| = 12$. Assume that we start with the polycyclic generating sequence (k_1) consisting of a unique element k_1 of order 12. We refine this sequence as (h_1, h_2, h_3) with $h_1 = k_1^{|G|/p_1} = k_1^6$, $h_2 = k_1^{|G|/p_1^2} = k_1^3$ and $h_3 = k_1^{|G|/(p_1^2 p_2)} = k_1$. This is a polycyclic generating sequence with $|H_1/H_0| = 2$, $|H_2/H_1| = 2$ and $|H_3/H_2| = 3$. The difficulty is that naturally we do not know the order $|G|$. Remember nevertheless that

we know the encoding length n of the black-box group, which is an upper bound on $\log_2 |G|$. This means that the quantity $m = p_1^n \times \ldots \times p_\ell^n$ is a multiple of the order $|G|$, and thus we can use the same approach, working with m instead of $|G|$ when refining the original polycyclic generating sequence.

Proof of Theorem 7. Let us consider the function $\lambda\colon \{1,\ldots,\ell\} \times \{1,\ldots,n\} \to \mathbb{Z}$ such that

$$\lambda(i,a) = p_i^{n-a} \times p_{i+1}^n \times \cdots \times p_\ell^n$$

for any $(i,a) \in \{1,\ldots,\ell\} \times \{1,\ldots,n\}$. Now consider the sequence

$$(\lambda(1,1),\ldots,\lambda(1,n),\lambda(2,1),\ldots,\lambda(2,n),\ldots,\lambda(\ell,1),\ldots,\lambda(\ell,n)) \tag{1}$$

consisting of ℓn integers (the integers in the sequence are strictly decreasing). Define the function $\mu\colon \{1,\ldots,\ell n\} \to \mathbb{Z}$ such that $\mu(j)$ is the j-th integer in Sequence (1). Note that $\mu(j-1)/\mu(j) \in S$ for any $j \in \{2,\ldots,\ell n\}$.

We now describe our algorithm that computes the claimed generating sequence.

We first compute a polycyclic generating sequence $(k_1,\ldots,k_{t'})$ of G with $t' = O(\log|G|)$ using the randomized polynomial-time algorithm from [8], already mentioned in Section 2, which succeeds with probability at least $1 - 1/\mathrm{poly}(|G|)$. Let us write $K_{i'} = \langle k_1,\ldots,k_{i'}\rangle$ for each $1 \le i' \le t'$, and $K_0 = \{e\}$.

We now show how to refine the polycyclic generating sequence. For each $i' \in \{1,\ldots,t'\}$, we replace $k_{i'}$ by the sequence of ℓn elements $(k_{i'}^{\mu(1)},\ldots k_{i'}^{\mu(\ell n)})$, which gives a new sequence

$$\left(k_1^{\mu(1)},\ldots,k_1^{\mu(\ell n)},k_2^{\mu(1)},\ldots,k_2^{\mu(\ell n)},\ldots,k_{t'}^{\mu(1)},\ldots,k_{t'}^{\mu(\ell n)}\right), \tag{2}$$

of $\ell n t'$ elements. Sequence (2) is a polycyclic generating sequence of G since $(k_1,\ldots,k_{t'})$ is a polycyclic generating sequence of G and $\mu(\ell n) = 1$. For any $i' \in \{1,\ldots,t'\}$, observe that

$$\left|\langle k_1^{\mu(1)},\ldots,k_{i'}^{\mu(j)}\rangle / \langle k_1^{\mu(1)},\ldots,k_{i'}^{\mu(j-1)}\rangle\right| \in \{1,\mu(j-1)/\mu(j)\} \tag{3}$$

for any $j \in \{2,\ldots \ell n\}$. Similarly for any $i' \in \{2,\ldots t'\}$ we have

$$\left|\langle k_1^{\mu(1)},\ldots,k_{i'}^{\mu(1)}\rangle / \langle k_1^{\mu(1)},\ldots,k_{i'-1}^{\mu(\ell n)}\rangle\right| \in \{1,p_1\}. \tag{4}$$

Let us rename the elements of Sequence (2) as $h_1,\ldots,h_t$, with $t = \ell n t'$. Note that $t = O(\ell(\log|G|)^2) = O((\log|G|)^3)$. Let us write $H_i = \langle h_1,\ldots,h_i\rangle$ for $1 \le i \le t$ and $K_0 = \{e\}$. For each $1 \le i \le t$, the order of H_i/H_{i-1} is either 1 or r_i, where r_i can be determined from Equations (3) and (4). More concretely, r_i is of the form $\mu(j-1)/\mu(j)$ for some j (which can be immediately computed from i) when H_i/H_{i-1} corresponds to the case of Equation (3), and $r_i = p_1$ when H_i/H_{i-1} corresponds to the case of Equation (4). Note that in both cases we have $r_i \in S$, from the property $\mu(j-1)/\mu(j) \in S$ mentioned before. ◀

3.2 The protocol

Let $S = \{p_1,\ldots,p_\ell\}$ denote the set of prime factors of $|G|$, which is given as an additional input. The protocol is given in Figure 1. The main idea is that the verifier can, using Theorem 7, compute by itself a polycyclic generating sequence $(h_1,\ldots,h_t)$ and prime numbers $r_1,\ldots,r_t$ such that $|H_i/H_{i-1}| \in \{1,r_i\}$ for each $1 \le i \le t$. This is done at Step 1 of the protocol. Note that $|G| = \prod_{i=1}^t |H_i/H_{i-1}|$. The purpose of Steps 2-5 is to decide

Input: • a black-box solvable group G with generators $\{g_1, \ldots, g_s\}$
• the set $S = \{p_1, \ldots, p_\ell\}$ of prime factors of $|G|$

1. The verifier uses the algorithm of Theorem 7 to compute elements $h_1, \ldots, h_t$ and prime numbers $r_1, \ldots, r_t \in S$. Let us write $H_i = \langle h_1, \ldots, h_i \rangle$ for each $1 \leq i \leq t$ and $H_0 = \{e\}$.
2. For each $i \in \{1, \ldots, t\}$, the verifier takes a bit $s_i \in \{0, 1\}$ uniformly at random and takes a random element $x_i \in H_{i-1}$ using the algorithm of Theorem 2 with $\varepsilon = 1/2^{2n}$ (where n represents the encoding length of the black-box group).
3. The verifier sends to the prover the elements $h_1, \ldots, h_t$ and, for each $i \in \{1, \ldots, t\}$, the element $h_i^{s_i} x_i$.
4. The prover sends to the verifier bits $b_1, \ldots, b_t$ and integers $a_{i,j}$ for $i \in \{1, \ldots t\}$ and $j \in \{1, \ldots, i-1\}$.
5. For each $i \in \{1, \ldots, t\}$ the verifier does the following:
   5.1 If $h_i = h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$ then set $\ell_i = 1$;
   5.2 If $h_i \neq h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$ and $b_i = s_i$ then set $\ell_i = r_i$;
   5.3 If neither of these two conditions holds, then abort the protocol and output $\perp$.
6. The verifier outputs the product of the ℓ_i's.

Figure 1 Our 2-message protocol computing the order of a solvable group when the prime factors of the order are known.

whether $|H_i/H_{i-1}| = 1$ or $|H_i/H_{i-1}| = r_i$, for each $i \in \{1, \ldots, t\}$, by interacting with the prover. More precisely, the verifier interacts with the prover to test, for each i, whether $h_i \in H_{i-1}$ or $h_i \notin H_{i-1}$. This requires testing non-membership in a solvable group with a polynomial-time quantum prover, which is achieved by sending (at Step 3) to the prover the element $h_i^{s_i} x_i$ for a random bit s_i and a random element x_i, and asking the prover to find the chosen bit s_i. These tests enable the verifier to decide which of the two cases holds (at Steps 5.1 and 5.2), and then to compute $|G|$ at Step 6, or to detect cheating (at Step 5.3).

3.3 Analysis of the protocol

We now analyze the protocol of Figure 1. Let $h_1, \ldots, h_t$ be the group elements and $r_1, \ldots, r_t \in S$ be the prime numbers computed at Step 1. The analysis below is done under the assumption that $(h_1, \ldots, h_t)$ is a polycyclic generating sequence of G and $|H_i/H_{i-1}| \in \{1, r_i\}$ for all $i \in \{1, \ldots, t\}$, which is true with probability at least $1 - 1/\mathrm{poly}(|G|)$ from Theorem 7.

Let us first consider the correctness, i.e., showing that there exists a prover (working in quantum polynomial time) who enables the verifier to compute $|G|$ with high probability. This prover acts as follows. For each $i \in \{1, \ldots, t\}$, the prover checks if the element $h_i^{s_i} x_i$ received at Step 3 is in the subgroup H_{i-1}, using Algorithm $\mathcal{A}_2$ of Theorem 5. If the prover learns that this element is in H_{i-1} then the prover applies Algorithm $\mathcal{A}_1$ of Theorem 5 to obtain a decomposition $(a_{i,1}, \ldots, a_{i,i-1})$ of h_i over H_{i-1}, and sends to the verifier the bit $b_i = 0$ and these values $a_{i,1}, \ldots, a_{i,i-1}$. If the prover learns that this element is not in H_{i-1}, then the prover sends to the verifier the bit $b_i = 1$ and arbitrary values $a_{i,1}, \ldots, a_{i,i-1}$.

Let us analyze the verifier's output when interacting with the above prover. If $|H_i/H_{i-1}| = 1$ then we have $h_i \in H_{i-1}$ and thus $h_i^{s_i} x_i \in H_{i-1}$ whatever the value of s_i is. With probability at least $1 - 1/\mathrm{poly}(|G|)$, the prover's message is thus $b_i = 0$ and $a_{i,1}, \ldots, a_{i,i-1}$ corresponding

to the decomposition of h_i over H_{i-1}, and then the verifier sets $\ell_i = 1$. If $|H_i/H_{i-1}| = r_i$ then we have $h_i \notin H_{i-1}$ and thus $h_i^{s_i} x_i \in H_{i-1}$ if and only if $s_i = 0$. With probability at least $1 - 1/\mathrm{poly}(|G|)$, the bit b_i sent by the prover satisfies $b_i = s_i$, and thus the verifier sets $\ell_i = r_i$ (since the second part of the message $a_{i,1}, \ldots, a_{i,i-1}$ cannot correspond to the decomposition of h_i over H_{i-1}). In conclusion, with probability at least $1 - 1/\mathrm{poly}(|G|)$ the output at Step 6 is

$$\prod_{i=1}^{t} \ell_i = \prod_{i=1}^{t} |H_i/H_{i-1}| = |G|.$$

Let us now consider the soundness, i.e., showing that for any prover the verifier outputs either $|G|$ or $\perp$ with high probability. It is clear that if $|H_i/H_{i-1}| = r_i$, then the prover cannot convince the verifier to set $\ell_i = 1$, since there is no set of integers $a_{i,1}, \ldots, a_{i,i-1}$ such that $h_i = h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$. On the other hand, if $|H_i/H_{i-1}| = 1$ then the prover cannot convince the verifier to set $\ell_i = r_i$ unless the prover is able to decide whether $s_i = 0$ or $s_i = 1$ from the element $h_i^{s_i} x_i$ received, which cannot be done with probability larger than $\frac{1}{2} + \frac{1}{2}\delta$, where

$$\delta = \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr_{x_i}[x_i = h] - \Pr_{x_i}[h_i x_i = h] \right|$$

represents the variational distance between the two probability distributions x_i and $h_i x_i$ (seen as distributions over H_{i-1}). We have

$$\begin{aligned}
\delta &\leq \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr[x_i = h] - \frac{1}{|H_{i-1}|} \right| + \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr[h_i x_i = h] - \frac{1}{|H_{i-1}|} \right| \\
&= \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr[x_i = h] - \frac{1}{|H_{i-1}|} \right| + \frac{1}{2} \sum_{h \in H_{i-1}} \left| \Pr[x_i = h_i^{-1} h] - \frac{1}{|H_{i-1}|} \right| \\
&= \sum_{h \in H_{i-1}} \left| \Pr[x_i = h] - \frac{1}{|H_{i-1}|} \right| \\
&\leq |H_{i-1}| \varepsilon \\
&\leq 1/2^n,
\end{aligned}$$

where the second inequality follows from Theorem 2 and the third inequality follows from our choice of ε and the upper bound $|G| \leq 2^n$. Thus, for any fixed i such that $|H_i/H_{i-1}| = 1$, the prover cannot convince the verifier to set $\ell_i = r_i$ with probability greater than $\frac{1}{2} + \frac{1}{2^{n+1}} = 1/2 + 1/\mathrm{poly}(|G|)$. Let us now bound the probability that the verifier's output is either $|G|$ or $\perp$. This corresponds to the probability that the verifier does not output an integer different from the order of G. Note that the verifier can output an integer not equal to the order only if the prover forces the verifier to set $\ell_i \neq |H_i/H_{i-1}|$ for at least one index i. From the above analysis, we know that this can happen with probability at most $1/2 + 1/\mathrm{poly}(|G|)$, i.e., such a cheating is detected by the verifier at Step 5.3 with probability at least $1/2 - 1/\mathrm{poly}(|G|)$, in which case the verifier immediately aborts the protocol and outputs $\perp$. Thus the overall probability that the verifier's output is either $|G|$ or $\perp$ is at least $1/2 - 1/\mathrm{poly}(|G|)$. Note finally that this probability can be amplified to reach the soundness threshold of $2/3$ used in Definition 6 by repeating the protocol of Figure 1 a constant number of times in parallel and deciding the output based on a standard threshold argument.

Input: a black-box solvable group G with generators $\{g_1, \ldots, g_s\}$

0. The prover sends to the verifier the following:
- **a.** a list of t elements $h_1, \ldots, h_t \in G$, for some $t = \text{poly}(\log |G|)$;[9]
- **b.** a list of t prime numbers $r_1, \ldots, r_t$;
- **c.** a list of integers $\alpha_{i,j}$, for $i \in \{1, \ldots, s\}$ and $j \in \{1, \ldots, t\}$;
- **d.** a list of integers $\beta_{i,j}$, for $i \in \{2, \ldots, t\}$ and $j \in \{1, \ldots, i-1\}$;
- **e.** a list of integers $\gamma_{i,j,\ell}$, for $i \in \{2, \ldots, t\}$ and $j, \ell \in \{1, \ldots, i-1\}$.

Let us write $H_i = \langle h_1, \ldots, h_i \rangle$ for each $1 \leq i \leq t$ and $H_0 = \{e\}$.

1. The verifier checks that the following equalities hold:
- **a.** $g_i = h_1^{\alpha_{i,1}} \cdots h_t^{\alpha_{i,t}}$ for all $i \in \{1, \ldots, s\}$;
- **b.** $h_i^{r_i} = h_1^{\beta_{i,1}} \cdots h_{i-1}^{\beta_{i,i-1}}$ for all $i \in \{2, \ldots, t\}$ and $h_1^{r_1} = e$;
- **c.** $h_i h_\ell h_i^{-1} = h_1^{\gamma_{i,1,\ell}} \cdots h_{i-1}^{\gamma_{i,i-1,\ell}}$ for all $i \in \{2, \ldots, t\}$ and all $\ell \in \{1, \ldots, i-1\}$.

If any of these equalities fails, then the verifier aborts the protocol and outputs $\perp$.

2. For each $i \in \{1, \ldots, t\}$, the verifier takes a bit $s_i \in \{0, 1\}$ uniformly at random and takes a random element $x_i \in H_{i-1}$ using the algorithm of Theorem 2 with $\varepsilon = 1/2^{2n}$.

3. The verifier sends, for each $i \in \{1, \ldots, t\}$, the element $h_i^{s_i} x_i$.

4. The prover sends to the verifier bits $b_1, \ldots, b_t$ and integers $a_{i,j}$ for $i \in \{1, \ldots t\}$ and $j \in \{1, \ldots, i-1\}$.

5. For each $i \in \{1, \ldots, t\}$ the verifier does the following:
- 5.1 If $h_i = h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$ then set $\ell_i = 1$;
- 5.2 If $h_i \neq h_1^{a_{i,1}} \cdots h_{i-1}^{a_{i,i-1}}$ and $b_i = s_i$ then set $\ell_i = r_i$;
- 5.3 If neither of these two conditions holds, then abort the protocol and output $\perp$.

6. The verifier outputs the product of the ℓ_i's.

Figure 2 Our 3-message protocol computing the order of a solvable group.

4 General 3-Message Protocol

In this section we show that when the prime factors of the order of G are not known, we can design a 3-message protocol, which proves the first part of Theorem 1.

4.1 The protocol

Our 3-message protocol, described in Figure 2, is obtained by modifying the protocol of the previous section. More precisely, Step 1 in the protocol of the previous section is replaced by two steps (Steps 0 and 1 in Figure 2): instead of having the verifier compute a polycyclic generating sequence $(h_1, \ldots, h_t)$ using Theorem 7, which requires the knowledge of the set of factors of $|G|$, in the new protocol the prover computes by itself this sequence and sends it at Step 0 to the verifier, who then checks that the sequence is really correct at Step 1. All the other steps 2-6 are exactly the same as for the protocol in Figure 1 (one small exception is Step 3, which is slightly rewritten since the polycyclic generating sequence does not need to be sent to the prover anymore).

4.2 Analysis of the protocol

Let us consider the correctness. In that case the prover first uses the algorithm of Theorem 4 to compute the order $|G|$, then factorizes it using Shor's algorithm [30] and collects the prime factors in a set S. The prover then uses the algorithm of Theorem 7 using the set S as input to obtain group elements $h_1, \ldots, h_t$ and a list of integers $r_1, \ldots, r_t \in S$ such that with probability at least $1 - 1/\text{poly}(|G|)$ the following two conditions hold:

(i) $(h_1, \ldots, h_t)$ is a polycyclic generating sequence of G, with $t = \text{poly}(\log |G|)$,

(ii) the order of H_i/H_{i-1} is either 1 or r_i for each $1 \leq i \leq t$,

where as usual we use the notation $H_i = \langle h_1, \ldots, h_i \rangle$ for any $i \in \{1, \ldots, t\}$ and the convention $H_0 = \{e\}$. These two conditions are equivalent to the following:

(a) $H_t = G$, i.e., $g_i \in H_t$ for each $i \in \{1, \ldots, s\}$;

(b) $h_i^{r_i} \in H_{i-1}$ for each $i \in \{1, \ldots, t\}$;

(c) H_{i-1} is normal in H_i for any $i \in \{2, \ldots, t\}$, i.e., $h_i h_\ell h_i^{-1} \in H_{i-1}$ for any $\ell \in \{1, \ldots, i-1\}$.

Thus, with probability at least $1 - 1/\text{poly}(|G|)$, the prover can compute the following decompositions in quantum polynomial time using Algorithm $\mathcal{A}_1$ of Theorem 5:

- a decomposition $(\alpha_{i,1}, \ldots, \alpha_{i,t})$ of g_i over H_t, for each $i \in \{1, \ldots, s\}$;
- a decomposition $(\beta_{i,1}, \ldots, \beta_{i,i-1})$ of $h_i^{r_i}$ over H_{i-1}, for each $i \in \{2, \ldots, t\}$;
- a decomposition $(\gamma_{i,1,\ell}, \ldots, \gamma_{i,i-1,\ell})$ of $h_i h_\ell h_i^{-1}$ over H_{i-1}, for each $i \in \{2, \ldots, t\}$ and each $\ell \in \{1, \ldots, i-1\}$.

At Step 0, the prover sends all these integers, along with the elements $h_1, \ldots, h_t$ and the primes $r_1, \ldots, r_t$. All the tests performed by the verifier at Step 1 then pass. The analysis of the second part of the protocol (Steps 2-6) is then exactly the same as the analysis of the protocol of Section 3.

The soundness follows by observing that passing the tests performed by the verifier at Step 1 guarantees that Conditions (a)-(c) of the previous paragraph hold. This guarantees that Conditions (i)-(ii) hold as well, and thus the soundness analysis for the second part of the protocol (Steps 2-6) is exactly the same as the analysis of the protocol of Section 3.

References

1. Scott Aaronson and Greg Kuperberg. Quantum versus classical proofs and advice. *Theory of Computing*, 3:129–157, 2007. doi:10.4086/toc.2007.v003a007.
2. Dorit Aharonov, Michael Ben-Or, Elad Eban, and Urmila Mahadev. Interactive proofs for quantum computations. *arXiv:1704.04487*, 2017.
3. Dorit Aharonov and Ayal Green. A quantum inspired proof of $P^{\#P} \subseteq IP$. *arXiv:1710.09078*, 2017.
4. Dorit Aharonov and Umesh Vazirani. Is quantum mechanics falsifiable? A computational perspective on the foundations of quantum mechanics. *arXiv:1206.3686*, 2012.
5. László Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 164–174, 1991. doi:10.1145/103418.103440.

[9] Naturally, this is binary strings corresponding to the elements $h_1, \ldots, h_t$ (i.e., the oracle representations of these elements) that are actually sent, not the elements themselves. Note also that, to simplify the exposition, we are assuming that these strings do correspond to elements of G. To deal with a cheating prover that may send strings not corresponding to group elements, we can simply ask the prover to send a certificate of membership in G for each string (such a certificate can be computed in quantum polynomial time using the algorithms of Theorem 5).

6 László Babai. Bounded round interactive proofs in finite groups. *SIAM Journal on Discrete Mathematics*, 5(1):88–111, 1992. doi:10.1137/0405008.

7 László Babai, Robert Beals, and Ákos Seress. Polynomial-time theory of matrix groups. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 55–64, 2009. doi:10.1145/1536414.1536425.

8 László Babai, Gene Cooperman, Larry Finkelstein, Eugene M. Luks, and Ákos Seress. Fast monte carlo algorithms for permutation groups. *Journal of Computer and System Sciences*, 50(2):296–308, 1995. doi:10.1006/jcss.1995.1024.

9 László Babai and Endre Szemerédi. On the complexity of matrix group problems I. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 229–240, 1984. doi:10.1109/SFCS.1984.715919.

10 Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26:1411–1473, 1997. doi:10.1137/S0097539796300921.

11 Simon R. Blackburn, Peter M. Neumann, and Geetha Venkataraman. *Enumeration of Finite Groups*. Cambridge University Press, 2017. doi:10.1017/CBO9780511542756.

12 Tommaso F. Demarie, Yungkai Ouyang, and Joseph F. Fitzsimons. Classical verification of quantum circuits containing few basis changes. *arXiv:1612.04914*, 2016.

13 Joseph F. Fitzsimons, Michael Hajdušek, and Tomoyuki Morimae. Post hoc verification of quantum computation. *Physical Review Letters*, 120:040501, 2018. doi:10.1103/PhysRevLett.120.040501.

14 Joseph F. Fitzsimons and Elham Kashefi. Unconditionally verifiable blind computation. *Physical Review A*, 96:012303, 2017. doi:10.1103/PhysRevA.96.012303.

15 Shafi Goldwasser, Ofer Grossman, and Dhiraj Holden. Pseudo-deterministic proofs. In *Proceedings of the 9th Innovations in Theoretical Computer Science Conference*, pages 17:1–17:18, 2018. doi:10.4230/LIPIcs.ITCS.2018.17.

16 Masahito Hayashi and Tomoyuki Morimae. Verifiable measurement-only blind quantum computing with stabilizer testing. *Physical Review Letters*, 115:220502, 2015. doi:10.1103/PhysRevLett.115.220502.

17 Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of computational group theory*. Chapman & Hall/CRC, 2005. doi:10.1201/9781420035216.

18 I. Martin Isaacs. *Finite group theory*. American Mathematical Society, 2008.

19 Gábor Ivanyos, Frédéric Magniez, and Miklos Santha. Efficient quantum algorithms for some instances of the non-abelian hidden subgroup problem. *International Journal of Foundations of Computer Science*, 14(5):723–740, 2003. doi:10.1142/S0129054103001996.

20 Zhengfeng Ji. Classical verification of quantum proofs. In *Proceedings of the 48th Annual ACM symposium on Theory of Computing*, pages 885–898, 2016. doi:10.1145/2897518.2897634.

21 Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. doi:10.1145/146585.146605.

22 Urmila Mahadev. Classical verification of quantum computations. *arXiv:1804.01082*, 2018.

23 Mathew McKague. Interactive proofs with efficient quantum prover for recursive fourier sampling. *Chicago Journal of Theoretical Computer Science*, 2012(6), 2012. doi:10.4086/cjtcs.2012.006.

24 Mathew McKague. Interactive proofs for BQP via self-tested graph states. *Theory of Computing*, 12(3):1–42, 2016. doi:10.4086/toc.2016.v012a003.

25 Tomoyuki Morimae, Daniel Nagaj, and Norbert Schuch. Quantum proofs can be verified using only single-qubit measurements. *Physical Review A*, 93:022326, 2016. doi:10.1103/PhysRevA.93.022326.

26 Tomoyuki Morimae, Yuki Takeuchi, and Harumichi Nishimura. Merlin-Arthur with efficient quantum Merlin and quantum supremacy for the second level of the fourier hierarchy. *arXiv:1711.10605*, 2017.

27 Ben W. Reichardt, Falk Unger, and Umesh Vazirani. Classical command of quantum systems. *Nature*, 496:456–460, 2013. doi:10.1038/nature12035.

28 Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992. doi:10.1145/146585.146609.

29 Yaoyun Shi. Quantum and classical tradeoffs. *Theoretical Computer Science*, 344:335–343, 2005. doi:10.1016/j.tcs.2005.03.053.

30 Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. doi:10.1137/S0097539795293172.

31 Daniel R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. doi:10.1137/S0097539796298637.

32 John Watrous. Succinct quantum proofs for properties of finite groups. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 537–546, 2000. doi:10.1109/SFCS.2000.892141.

33 John Watrous. Quantum algorithms for solvable groups. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 60–67, 2001. doi:10.1145/380752.380759.

34 https://www-03.ibm.com/press/us/en/pressrelease/52403.wss.