Quantum Information Processing (2021) 20:205
https://doi.org/10.1007/s11128-021-03069-1

On completely factoring any integer efficiently in a single run of an order-finding algorithm

Martin Ekerå[1,2]

Received: 6 December 2020 / Accepted: 15 March 2021 / Published online: 9 June 2021

Abstract
We show that given the order of a single element selected uniformly at random from $\mathbb{Z}_N^*$, we can with very high probability, and for any integer N, efficiently find the complete factorization of N in polynomial time. This implies that a single run of the quantum part of Shor's factoring algorithm is usually sufficient. All prime factors of N can then be recovered with negligible computational cost in a classical post-processing step. The classical algorithm required for this step is essentially due to Miller.

Keywords Factoring · Order finding · Shor's algorithms

Mathematics Subject Classification 11A51 · 68Q12 · 81P68

1 Introduction

In what follows, let

$$N = \prod_{i=1}^{n} p_i^{e_i}$$

be an m bit integer, with $n \geq 2$ distinct prime factors p_i, for e_i some positive integer exponents. Let an algorithm be said to *factor* N if it computes a non-trivial factor of N, and to *completely* factor N if it computes the set $\{p_1, \ldots, p_n\}$.

Let ϕ be Euler's totient function, let λ be the Carmichael function, and let $\lambda'(N) = \text{lcm}(p_1 - 1, \ldots, p_n - 1)$. Furthermore, let $\mathbb{Z}_N^*$ denote the multiplicative group of $\mathbb{Z}_N$, the ring of integers modulo N, and let ln and log be the natural and base-two logarithms, respectively. Denote by $[a, b]$ the range of integers from a up to and including b.

✉ Martin Ekerå
ekera@kth.se

1 KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden

2 Swedish NCSA, Swedish Armed Forces, SE-107 85 Stockholm, Sweden

Throughout this paper, we shall assume N to be odd for proof-technical reasons. This does not imply a loss of generality: It is easy to fulfill this requirement by using trial division. Indeed, one would in general always remove small prime factors before calling upon more elaborate factoring algorithms. Note furthermore that order finding may be performed prior to trial division being applied to N if desired, see Sect. 3.2.2 for further details.

There exist efficient probabilistic primality tests, such as Miller–Rabin [15,18], and efficient algorithms for reducing perfect powers $z = q^e$ to q: A simple option is to test if $z^{1/d}$ is an integer for some $d \in [2, \lfloor \log z \rfloor]$. For more advanced options, see e.g. Bernstein et al. [1].

2 Earlier works

Shor [21,22] proposed to factor N by repeatedly selecting a random $g \in \mathbb{Z}_N^*$, computing its order r via quantum order finding, and executing a classical procedure inspired by Miller [15]. Specifically, Shor proposed to use that if r is even, and $g^{r/2} \not\equiv -1 \pmod{N}$, it must be that

$$(g^{r/2} - 1)(g^{r/2} + 1) = g^r - 1 \equiv 0 \pmod{N}$$

where $\gcd((g^{r/2} \pm 1) \bmod N, N)$ yields non-trivial factors of N. Note that it holds that $g^{r/2} \not\equiv 1 \pmod{N}$ by definition, as r is otherwise not the order of g.

Shor proved that the probability of the above two requirements being met is at least $1/2$. If both requirements are not met, the algorithm may be re-run for a new g, in which case the probability is again at least $1/2$ of succeeding.

This implies that Shor's algorithm will eventually succeed in splitting N into two non-trivial factors. However, as re-running the quantum order-finding part of the algorithm is expensive, it is natural to consider improved strategies.

To completely factor N, recursive calls to Shor's factoring algorithm, and hence to the quantum order-finding part, would naïvely be required, albeit with consecutively smaller factors, until the factors are prime, perfect powers, or sufficiently small to factor using classical algorithms. Again, it is natural to consider improved strategies to avoid re-runs in this setting.

2.1 On the success probability of quantum order finding

Shor's factoring algorithm as originally described can fail either because the order r of g is not amenable to factoring N, in the sense that r is odd or $g^{r/2} \equiv -1 \pmod{N}$, or because the order-finding part of the algorithm fails to return r given g.

The probability of the algorithm failing for the latter reason is negligible, however, if the quantum part is correctly parameterized and post-processed, and if it is executed as per its mathematical description by the quantum computer, see e.g. Appendix A to [4] or [2] for analyses.

In what follows, we therefore primarily focus our attention on classically recovering non-trivial factors of N given r. When referring to factoring in a single run of an order-finding algorithm, we assume the order-finding algorithm to yield r given g.

2.2 Tradeoffs in quantum order finding

Before proceeding, we note that Seifert [20] has proposed to modify the order-finding part of Shor's algorithm to enable tradeoffs between the number of runs that need to be performed on the quantum computer and the complexity of each run.

In essence, Seifert's idea is to compute only partial information on the order in each run, thereby reducing the number of operations that need to be performed by the computer in each run without uncorrectable errors arising. Given the outputs from a sufficiently large number of such partial runs, the order may then be re-constructed efficiently classically, yielding a complete order-finding algorithm that returns r with high probability given g. In the context of making tradeoffs, whenever we refer to a single run of an order-finding algorithm in this work, it should be understood to refer to a single run of a complete algorithm that yields r given g.

Making tradeoffs may prove advantageous in the early days of quantum computing when the capabilities of the computers available are limited. In a future with large-scale quantum computers, the fact that the partial runs are independent, and may be executed in parallel, may furthermore prove useful.

On a side note, Knill [9] has proposed a different kind of tradeoffs, where the goal is not to perform fewer operations in each run, but rather to obtain improved lower bounds on the success probability of the order being returned.

2.3 Improvements for odd orders

Several improvements to Shor's original classical post-processing approach have been proposed, including in particular ways of recovering factors of N from odd orders [6,8,10,14]. Grosshans et al. [6] point out that if a small prime factor q divides r, then $\gcd((g^{r/q} - 1) \bmod N, N)$ is likely to yield non-trivial factors of N. Johnston [8] later made similar observations.

In the context of Shor's algorithm, the observation that odd r may yield non-trivial factors of N seems to first have been made by Martín-López et al. [14] in an actual experimental implementation. This is reported in a work by Lawson [10] and later by Grosshans et al. [6].

We may efficiently find all small prime factors of r. This often gives us several attempts at recovering non-trivial factors of N, leading to an increase in the probability of factoring N. Furthermore, we can try all combinations of these prime factors, with multiplicity when applicable, to increase the number of non-trivial divisors.

2.4 Improvements for special form integers

Ekerå and Håstad [3,5] have introduced a specialized algorithm for factoring RSA integers that is more efficient than Shor's general factoring algorithm. The problem of factoring RSA integers merits special consideration because it underpins the security of the widely deployed RSA cryptosystem [19].

The algorithm of Ekerå and Håstad classically reduces the RSA integer factoring problem to a short discrete logarithm problem in a cyclic group of unknown order, using ideas from [7], and solves this problem quantumly. It is more efficient primarily because the quantum step is less costly compared to traditional quantum order finding, both when not making tradeoffs and comparing to Shor, and when making tradeoffs and comparing to Seifert. It furthermore allows for the two factors of the RSA integer to be recovered deterministically, once the short discrete logarithm has been computed. This implies that there is little point in optimizing the post-processing in Shor's original algorithm if the goal is to factor RSA integers.

On the topic of factoring special form integers, Grosshans et al. [6] have furthermore shown how so-called safe semi-primes may be factored deterministically after a single run of Shor's original order-finding algorithm. Xu et al. [23] have presented similar ideas. Leander [11] has shown how the lower bound of $1/2$ in Shor's original analysis may be improved to $3/4$ for semi-primes.

2.5 Other related works on factoring via order finding

There is a considerable body of literature on factoring. The specific problem of factoring via number theoretical oracles has been widely explored, in the scope of various contexts. Many of the results have a lineage that can be traced back to the seminal works of Miller [15].

More recently, Morain et al. [16] have investigated *deterministic* algorithms for factoring via oracles that yield $\phi(N)$, $\lambda(N)$ or the order r of an element $g \in \mathbb{Z}_N^*$. They find that given $\phi(N)$, it is possible to factor N unconditionally and deterministically in polynomial time, provided that certain conditions on the prime factors of N are met: It is required that N be square-free and that N has a prime factor $p > \sqrt{N}$. Their approach leverages the Lenstra–Lenstra–Lovász (LLL) [12] lattice basis reduction algorithm.

Morain et al. furthermore explicitly note that their work is connected to Shor's factoring algorithm, and that efficient *randomized* factoring algorithms are produced by all three oracles (see Sects. 2.3 and 2.5 in [16]). They recall the method of Miller [15], its adapted use in Shor's algorithm for factoring via an oracle that yields the order r of $g \in \mathbb{Z}_N^*$, and the fact that it may be necessary to consider multiple g to find one with even order suitable for factoring N. This implies that multiple oracle calls may be required to find non-trivial factors.

The authors furthermore state that if one has access to an oracle that yields e.g. $\phi(N)$ or $\lambda(N)$, it is possible to do better: It is then possible to find a $g \not\equiv \pm 1 \pmod{N}$ such that $g^2 \equiv 1 \pmod{N}$. In particular, one may use that the orders of all elements in $\mathbb{Z}_N^*$ divide $\lambda(N) = 2^t o$, for some positive t and odd o, to efficiently find such g. The

same statement holds for $\phi(N)$. This is closely related to the observations made in this paper. The original algorithm is from Miller [15].

2.6 On the relation to our contribution

Given the abundance of literature on the topic of factoring, it is admittedly hard to make new original contributions, or even to survey the existing literature in its entirety. We are, however, not aware of anyone previously demonstrating, within the context of Shor's algorithm, that a single call to the order-finding algorithm is in general sufficient to completely factor any composite integer with high probability.

On the contrary, it is sometimes said or implied that Shor's original post-processing algorithm should be used in practice, potentially requiring several order-finding calls to find even a non-trivial factor, let alone the complete factorization.

3 Our contribution

We give an efficient classical probabilistic polynomial-time algorithm, that is essentially due to Miller [15], for completely factoring N given the order r of a single element g selected uniformly at random from $\mathbb{Z}_N^*$. We furthermore analyze the runtime and success probability of the algorithm: In particular, we give a lower bound on its success probability.

3.1 Notes on our original intuition for this work

Given the order r of g, we can in general correctly guess the orders of a large fraction of the other elements of $\mathbb{Z}_N^*$ with high probability.

To see why this is, note that g is likely to have an order such that $\lambda(N)/r$ is a moderate size product of small prime factors. Hence, by multiplying on or dividing off small prime factors to r, we can guess $\lambda(N)$, and by extension the orders of other elements in the group. This observation served as our original intuition for pursuing this line of work. In this paper, we do, however, take matters a few steps further:

In particular, instead of guessing the orders of individual elements in $\mathbb{Z}_N^*$, we instead guess some multiple of $\lambda'(N)$. Furthermore, we show that even if we only manage to guess some multiple of a divisor of $\lambda'(N)$, we are still often successful in recovering the complete factorization of N.

3.2 The algorithm

In what follows, we describe a classical algorithm, essentially due to Miller [15] (see the algorithm in Lemma 5 ERH), for completely factoring N given a multiple of $\lambda'(N)$. We have slightly modified it, however, by adding a step in which we attempt to guess such a multiple, denoted r' below, given the order r of g.

Furthermore, we select k group elements x_j uniformly at random from $\mathbb{Z}_N^*$, for

$k \geq 1$ some small parameter that may be freely selected, whereas Miller iterates over all elements up to some bound.

With these modifications, we shall prove that the resulting probabilistic algorithm runs in polynomial time, with the possible exception of the call to an order-finding algorithm in the first step, and analyze its success probability.

To be specific, the resulting algorithm first executes the below procedure once to find non-trivial factors of N:

1 Select g uniformly at random from $\mathbb{Z}_N^*$.
Compute the order r of g via an order-finding algorithm.

2 Let $\mathcal{P}(B)$ be the set of primes $\leq B$.
Let $\eta(q, B)$ be the largest integer such that $q^{\eta(q,B)} \leq B$.
Let $m' = cm$ for some constant $c \geq 1$ that may be freely selected. Recall from the introduction that m is the bit length of N.
Compute $r' = r \prod_{q \in \mathcal{P}(m')} q^{\eta(q,m')}$.

3 Let $r' = 2^t o$ where o is odd.

4 For $j = 1, 2, \ldots, k$ for some $k \geq 1$ that may be freely selected do:

4.1 Select x_j uniformly at random from $\mathbb{Z}_N^*$.

4.2 For $i = 0, 1, \ldots, t$ do:

4.2.1 Compute $d_{i,j} = \gcd(x_j^{2^i o} - 1, N)$.
If $1 < d_{i,j} < N$ report $d_{i,j}$ as a non-trivial factor of N.

We then obtain the complete factorization from the $d_{i,j}$ reported as follows:

A set is initialized and N added to it before executing the above algorithm. For each non-trivial factor reported, the factor is added to the set. The set is kept reduced, so that it contains only non-trivial pairwise coprime factors. It is furthermore checked for each factor in the set, if it is a perfect power q^e, in which case q is reported as a non-trivial factor. The algorithm succeeds if the set contains all distinct prime factors of N when the algorithm stops.

Recall from the introduction that there are efficient methods for reducing q^e to q, and methods for testing primality in probabilistic polynomial time.

3.2.1 Notes on efficient implementation

Note that the algorithm as described in Sect. 3.2 is not optimized: Rather, it is presented for ease of comprehension and analysis.

In an actual implementation it would for example be beneficial to perform arithmetic modulo N' throughout step 4 of the algorithm, for N' a composite divisor of N that is void of prime factors that have already been found.

The algorithm would of course also stop incrementing j as soon as the factorization is complete, rather than after k iterations, and stop incrementing i as soon as $x_j^{2^i o} \equiv 1 \pmod{N'}$ rather than continue up to t. It would select x_j and g from $\mathbb{Z}_N^* \backslash \{1\}$ rather than from $\mathbb{Z}_N^*$. In step 4.2.1, it would compute $d_{i,j} = \gcd(u_{i,j} - 1, N')$, where

$u_{0,j} = x_j^o \bmod N'$, and $u_{i,j} = u_{i-1,j}^2 \bmod N'$ for $i \in [1, t]$, to avoid raising x_j to o repeatedly.

Ideas for potential optimizations: To further speed up the exponentiations, instead of raising each x_j to a pre-computed guess r' for a multiple of $\lambda'(N)$, a smaller exponent that is a multiple of the order of $x_j \bmod N'$ may conceivably be speculatively computed and used in place of r'. To obtain more non-trivial factors from each x_j, combinations of small divisors of the exponent may conceivably be exhausted; not only the powers of two that divide the exponent.

Missing factors: If an x_j is such that $w_j = x_j^{N'r'} \not\equiv 1 \pmod{N'}$, a factor q equal to the order of $w_j \bmod N'$ is missing in the guess r' for a multiple of $\lambda'(N')$. Should this lead the algorithm to fail to completely factor N', it may be worthwhile to attempt to compute the missing factor:

The options available include searching for q by exponentiating to all primes up to some bound, which is essentially analogous to increasing c, or using some form of cycle-finding algorithm that does not require a multiple of q to be known in advance.

In short, there are a number of optimizations that may be applied, but doing so above would obscure the workings of the algorithm, and the analysis that we are about to present. It is furthermore not necessary, since the algorithm as described is already very efficient.

3.2.2 Notes on performing order finding for a multiple of *N*

Note that the order-finding call in step 1 may be performed for a multiple of N if desired. This can only cause r to grow by some multiple, which in turn can only serve to increase the success probability, in the same way that growing r to r' in step 2 serves to increase the success probability, see Sect. 3.3. In turn, this explains why we can replace N with N' as described in the previous section, and why a restriction to odd N does not imply a loss of generality.

3.2.3 Notes on analogies with Miller's, Rabin's and Long's works

Miller's original version of the algorithm in Sect. 3.2 is deterministic, and proven to work only assuming the validity of the extended Riemann hypothesis (ERH), as is Miller's primality test in the same thesis [15].

Rabin [18] later converted Miller's primality test into a probabilistic polynomial-time algorithm that is highly efficient in practice. At about the same time, Long [13], acknowledging ideas from Flajolet, converted Miller's factoring algorithm into a probabilistic polynomial-time algorithm. This in a technical report that seemingly went unpublished.[1] More specifically, Long lower-bounds the probability of randomly selecting an element $g \in \mathbb{Z}_N^*$ of order r a multiple of $\lambda'(N)$. He then gives a randomized version of Miller's algorithm for splitting N into two non-trivial factors given this multiple of $\lambda'(N)$.

The above is closely related to our work. We take matters a step further, however, by converting Miller's factoring algorithm into an efficient probabilistic polynomial-time

[1] The report became available to us only as we were preparing to publish this paper.

algorithm for recovering the complete factorization of N given the order r of a single element g selected uniformly at random from $\mathbb{Z}_N^*$. We lower-bound the probability of the algorithm succeeding in recovering all factors of N given r. We show this probability to be very high, by carefully considering cases where r is a multiple of a divisor of $\lambda'(N)$.

3.2.4 Notes on analogies with Pollard's works

Miller's algorithm may be regarded as a generalization of Pollard's $p-1$ algorithm [17]: Miller essentially runs Pollard's algorithm for all prime factors p_i in parallel by using that a multiple of $\lambda'(N) = \operatorname{lcm}(p_1-1, \ldots, p_n-1)$ is known. Pollard, assuming no prior knowledge, uses a product of small prime powers up to some smoothness bound B in place of $\lambda'(N)$. This factors out p_i from N if $p_i - 1$ is B-smooth, giving Pollard's algorithm its name.

Since we only know r, a multiple of some divisor of $\lambda'(N)$, we grow r to r' by multiplying on a product of small prime powers. This is in analogy with Pollard's approach.

3.3 Analysis of the algorithm

A key difference between our modified algorithm in Sect. 3.2 and the original algorithm in Miller's thesis [15] is that we compute the order of a random g and then add a guessing step: We guess an r' in step 2 that we hope will be a multiple of $p_i - 1$ for all $i \in [1, n]$, and if not all, then at least for all but one index on this interval, in which case the algorithm will still be successful in completely factoring N.

This is shown in the below analysis. Specifically, we lower-bound the success probability and demonstrate the polynomial runtime of the algorithm. Throughout the analysis, and the remainder of this work, we use the notation implicitly introduced in the pseudocode for the algorithm in Sect. 3.2.

Furthermore, we use that for g selected uniformly at random from

$$\mathbb{Z}_N^* \simeq \mathbb{Z}_{p_1^{e_1}}^* \times \ldots \times \mathbb{Z}_{p_n^{e_n}}^*,$$

we have that $g \bmod p_i$ for $i \in [1, n]$ are selected independently and uniformly at random from the cyclic groups $\mathbb{Z}_{p_i}^*$. The same holds with x_j in place of g.

Definition 1 The prime p_i is unlucky if r' is not a multiple of $p_i - 1$.

Lemma 1 *The probability that p_i is unlucky is at most* $\log p_i / (m' \log m')$.

Proof For p_i to be unlucky, there has to exist a prime power q^e such that

(i) $q^e > m'$, as q^e otherwise divides r',
(ii) q^e divides $p_i - 1$, and
(iii) g is a q^e-power mod p_i, to reduce the order of g mod p_i by a factor q^e.

The number of such prime powers q^e that divide $p_i - 1$ is at most $\log p_i / \log m'$ by a simple size comparison, as $q^e > m'$ and as the product of the prime powers in question cannot exceed $p_i - 1$. For each such prime power, the probability that g is a q^e-power mod p_i is at most $1/q^e \leq 1/m'$. The lemma follows by taking the product of these two expressions. □

Lemma 2 *If at most one p_i is unlucky, then except with probability at most*

$$2^{-k} \cdot \binom{n}{2}$$

all n prime factors of N will be recovered by the algorithm after k iterations.

Proof For the algorithm not to find all prime factors, there must exist two distinct prime factors q_1 and q_2 that both divide N, such that for all combinations of $i \in [0, t]$ and $j \in [1, k]$, either both factors divide $x_j^{2^i o} - 1$, or none of them divide $x_j^{2^i o} - 1$.

To see why this is, note that the two factors will otherwise be split apart for some combination of i and j in step 4.2.1 of the algorithm in Sect. 3.2, and if this occurs pairwise for all factors, the algorithm will recover all factors.

There are $\binom{n}{2}$ ways to select two distinct primes from the n distinct primes that divide N. For each such pair, at most one of q_1 and q_2 is unlucky, by the formulation of the lemma.

(i) If either q_1 or q_2 is unlucky:
Without loss of generality, say that q_1 is lucky and q_2 is unlucky.

– The lucky prime q_1 then divides $x_j^{2^t o} - 1$. To see why this is, recall that $x_j \in \mathbb{Z}_N^*$ and that $q_1 - 1$ divides r' since q_1 is lucky, so

$$x_j^{2^t o} = x_j^{r'} \equiv 1 \pmod{q_1}.$$

– The unlucky prime q_2 divides $x_j^{2^t o} - 1$ iff $x_j^{2^t o} \equiv 1 \pmod{q_2}$.
For x_j selected uniformly at random from $\mathbb{Z}_N^*$, and odd q_2, where we recall that we assumed N and hence q_2 to be odd in the introduction, this event occurs with probability at most $1/2$.
To see why this is, note that since q_2 is unlucky, only an element x_j with an order modulo q_2 that is reduced from the maximum order $q_2 - 1$ by some factor dividing $q_2 - 1$ can fulfill the condition. The reduction factor must be at least two. It follows that at most $1/2$ of the elements in $\mathbb{Z}_N^*$ can fulfill the condition.

For each iteration $j \in [1, k]$, the failure probability is hence at most $1/2$.
Since there are k iterations, the total failure probability is at most 2^{-k}.

(ii) If both q_1 and q_2 are lucky:
In this case, both q_1 and q_2 divide $x_j^{2^t o} - 1$, since $x_j \in \mathbb{Z}_N^*$, and since $r' = 2^t o$

where $q_1 - 1$ and $q_2 - 1$ both divide r', so

$$x_j^{2^t o} = x_j^{r'} \equiv 1 \pmod{q} \quad \text{for } q \in \{q_1, q_2\}.$$

The algorithm fails iff x_j^o has the same order modulo both q_1 and q_2.
To see why this is, note that

$$d_{i,j} = \gcd(x_j^{2^i o} - 1, N)$$

is computed in step 4.2.1 of the algorithm, for $i \in [0, t]$, and that the prime $q \in \{q_1, q_2\}$ divides $d_{i,j}$ iff $x_j^{2^i o} \equiv 1 \pmod{q}$. It is only if this occurs for the same i for both q_1 and q_2 that q_1 and q_2 will not be split apart, i.e., if x_j^o has the same order modulo both q_1 and q_2.
To analyze the probability of x_j^o having the same order modulo q_1 and q_2, we let 2^{t_1} and 2^{t_2} be the greatest powers of two to divide $q_1 - 1$ and $q_2 - 1$, respectively. Recall furthermore that we assumed N, and hence q_1 and q_2, to be odd in the introduction. This implies that we may assume that $t \geq t_1 \geq t_2 \geq 1$ without loss of generality.

Consider $x_j^{2^{t_1-1} o}$:

- If $t_1 = t_2$, the probability that $x_j^{2^{t_1-1} o} - 1$ is divisible by q_1 but not by q_2 is $1/4$, and vice versa for q_2 and q_1. Hence, the probability is at most $1/2$ that x_j^o has the same order modulo both q_1 and q_2.
- If $t_1 > t_2$, the probability that $x^{2^{t_1-1} o} - 1$ is divisible by q_1 is $1/2$, whereas the same always holds for q_2. Hence, the probability is again at most $1/2$ that x_j^o has the same order modulo q_1 and q_2.

For each iteration $j \in [1, k]$, the probability is hence again at most $1/2$.
Since there are k iterations, the total failure probability is at most 2^{-k}.

The lemma follows from the above argument, as there are $\binom{n}{2}$ combinations with probability at most 2^{-k} each. □

By definition q is said to divide u iff $u \equiv 0 \pmod{q}$. Note that this implies that all $q \neq 0$ divide $u = 0$. This situation arises in the above proof of Lemma 2.

Lemma 3 *At least two primes are unlucky with probability at most*

$$\frac{1}{2c^2 \log^2 cm}.$$

Proof The three conditions used to establish the upper bound in the proof of Lemma 1 are independent for any two distinct primes p_i. Hence, by the proof of Lemma 1, we

have that the probability of at least two primes being unlucky is upper-bounded by

$$\sum_{(i_1,i_2)\in\mathcal{S}} \frac{\log p_{i_1}}{m' \log m'} \cdot \frac{\log p_{i_2}}{m' \log m'} \leq \frac{1}{2(m' \log m')^2} \left(\sum_{i=1}^{n} \log p_i \right)^2 \leq \frac{1}{2c^2 \log^2 cm}$$

where we used that $\sum_{i=1}^{n} \log p_i \leq \log N \leq m$ and $m' = cm$, and where $\mathcal{S}$ is the set of all pairs $(i_1, i_2) \in [1, n]^2$ such that the product $p_{i_1} \cdot p_{i_2}$ is distinct, and so the lemma follows. □

3.3.1 Runtime analysis

Claim 1 *It holds that* $\log r' = O(m)$.

Proof By the prime number theorem, there are $O(m'/\ln m')$ primes less than m'. As $r < N$ we have $\log r < m$. Furthermore, as each prime power q^e in r'/r is less than m', we have

$$\log r' \leq \log r + O(m'/\ln m') \cdot \log m' = O(m)$$

as $m' = cm$ for some constant $c \geq 1$, and so the claim follows. □

3.3.2 Main theorem

Theorem 1 *The factoring algorithm, with the possible exception of the single order-finding call, completely factors N in polynomial time, except with probability at most*

$$2^{-k} \cdot \binom{n}{2} + \frac{1}{2c^2 \log^2 cm}$$

where n is the number of distinct prime factors of N, m is the bit length of N, $c \geq 1$ is a constant that may be freely selected, and k is the number of iterations performed in the classical post-processing.

Proof It is easy to see that the non-order-finding part of the algorithm runs in polynomial time in m, as all integers are of length $O(m)$, including in particular r' by Claim 1. The theorem then follows from the analysis in Sect. 3.3, by summing the upper bound on the probability of a failure occurring when at most one prime is unlucky in Lemma 2, and on the probability of at least two primes being unlucky in Lemma 3. □

By the above main theorem, the algorithm will be successful in completely factoring N, if the constant c is selected so that $1/(2c^2 \log^2 cm)$ is sufficiently small, and if 2^{-k} for k the number of iterations is sufficiently small in relation to $\binom{n}{2}$ for n the number of distinct prime factors in N. The latter requirement is easy to meet: Pick $k \geq 2 \log n - 1 + \tau$ for some positive τ. Then $2^{-k} \cdot \binom{n}{2} \leq 2^{-\tau}$.

The time complexity of the algorithm is dominated by k exponentiations of an integer modulo N to an exponent of length $O(m)$ bits, and by the need to test the factors identified for primality. This is indeed very efficient.

Note furthermore that our analysis of the success probability of the algorithm is a worst-case analysis. In practice, the actual success probability of the algorithm is higher. Also, nothing in our arguments strictly requires c to be a constant: We can make c a function of m to further increase the success probability at the expense of working with $O(c(m)\,m)$ bit exponents.

4 Summary and conclusion

When factoring an integer N via order finding, as in Shor's factoring algorithm, computing the order of a single element selected uniformly at random from $\mathbb{Z}_N^*$ suffices to completely factor N, with very high probability, depending on how c and k are selected in relation to the number of factors n and the bit length m of N, for N any integer.

Acknowledgements I am grateful to Johan Håstad for valuable comments and advice. Funding and support for this work was provided by the Swedish NCSA that is a part of the Swedish Armed Forces. I thank the participants of the Schloss Dagstuhl quantum cryptanalysis seminar, and in particular Daniel J. Bernstein, for asking questions eventually leading me to consider more general factoring problems.

Funding Open access funding was provided by the Royal Institute of Technology.

A Supplementary simulations

We have implemented the algorithm in Sage to test it in practice. This is possible for any problem instance for which the factorization of $N = p_1^{e_1} \cdot \ldots \cdot p_n^{e_n}$ is known:

Order finding can be implemented exactly classically if the factorization of $p_i - 1$ is known for all $i \in [1, n]$. If only the p_i are known, order finding can be simulated heuristically classically: The correct order of g is then returned with very high probability. If the correct order is not returned, some multiple of the correct order is returned, see Sect. A.2. Both approaches are efficient.

A.1 Selecting problem instances

To setup problem instances, we select $n \geq 2$ distinct primes $\{p_1, \ldots, p_n\}$ uniformly at random from the set of all odd ℓ bit primes, exponents $\{e_1, \ldots, e_n\}$ uniformly at random from the integers on $[1, e_{\max}]$, and compute $N = p_1^{e_1} \cdot \ldots \cdot p_n^{e_n}$.

This yields N with two or more factors of size approximately ℓ bits, that may or may not occur with multiplicity depending on how $e_{\max}$ is selected, providing a suitable range of problem instances for testing the algorithm. Note that N on the above form are hard to factor classically when ℓ is large. However, we know the factorization by virtue of how N is selected, enabling us to simulate order finding heuristically when executing our tests.

A.2 Simulating order finding

To simulate order finding heuristically, we use for efficiency that selecting g uniformly at random from $\mathbb{Z}_N^*$ is equivalent to selecting $(g_1, \ldots, g_n)$ uniformly at random from

$$\mathbb{Z}^*_{p_1^{e_1}} \times \ldots \times \mathbb{Z}^*_{p_n^{e_n}} \simeq \mathbb{Z}_N^*.$$

To approximate the order r_i of each g_i thus selected, we let $\lambda(p_i^{e_i})$ be an initial guess for r_i. For all $f \in \mathcal{P}(B_s)$ for some bound B_s, we then let $r_i \leftarrow r_i/f$ for as long as f divides r_i and it holds that

$$g_i^{r_i/f} \equiv 1 \pmod{p_i^{e_i}},$$

where we recall that $\mathcal{P}(B_s)$ is the set of all primes $\leq B_s$.

We then construct g from g_i and p_i, e_i via the Chinese remainder theorem, by requiring that $g \equiv g_i \pmod{p_i^{e_i}}$ for all $i \in [1, n]$, and take $r = \operatorname{lcm}(r_1, \ldots, r_n)$ as the approximate order r of g. This is a good approximation of r, in the sense that it is equal to r with very high probability, provided that B_s is selected sufficiently large.

There is of course still a tiny risk that the approximation of r will be incorrect, in which case it will be equal to $c_r \cdot r$, for $c_r > B_s$ a factor that divides $\lambda(N)/r$. This implies that the factoring algorithm will perform slightly better under simulated order finding than under exact order finding, as the order is never reduced by factors greater than B_s. The difference is, however, negligible for sufficiently large B_s.

A.3 Results

We have executed tests, in accordance with the above, for all combinations of

$$\ell \in \{256, 512, 1024\} \quad n \in \{2, 5, 10, 25\} \quad e_{\max} \in \{1, 2, 3\}$$

with $c = 1$, unbounded k, and $B_s = 10^6$ in the order-finding simulator. As expected, the algorithm recovered all factors efficiently from r and N in all cases considered.

For these choices of parameters, the runtime typically varies from seconds up to minutes when the Sage script is executed on a regular laptop computer.

References

1. Bernstein, D.J., Lenstra, H.W., Pila, J.: Detecting perfect powers by factoring into coprimes. Math. Comput. **76**(257), 385–388 (2007)
2. Bourdon, P.S., Williams, H.T.: Sharp probability estimates for Shor's order finding algorithm. Quantum Inf. Comput. **7**(5), 522–550 (2007)
3. Ekerå, M.: On post-processing in the quantum algorithm for computing short discrete logarithms. Des. Codes Cryptogr. **88**(11), 2313–2335 (2020)
4. Ekerå, M.: Quantum algorithms for computing general discrete logarithms and orders with tradeoffs. J. Math. Cryptol. **15**(1), 359–407 (2021)
5. Ekerå, M., Håstad, J.: Quantum algorithms for computing short discrete logarithms and factoring RSA integers. In: PQCrypto, Lecture Notes in Comput. Sci. (LNCS), vol. 10346, Springer, Cham, pp. 347–363 (2017)
6. Grosshans, F., Lawson, T., Morain, F., Smith, B.: Factoring safe semiprimes with a single quantum query. ArXiv:1511.04385v3 (2015)
7. Håstad, J., Schrift, A.W., Shamir, A.: The discrete logarithm modulo a composite hides $O(n)$ bits. J. Comput. Syst. Sci. **47**(3), 376–404 (1993)
8. Johnston, A.M.: Shor's algorithm and factoring: don't throw away the odd orders. IACR ePrint Archive 2017/083 (2017)
9. Knill, E.: On Shor's quantum factor finding algorithm: increasing the probability of success and tradeoffs involving the Fourier transform modulus. Tech. Rep. LAUR-95-3350, Los Alamos National Laboratory (1995)
10. Lawson, T.: Odd orders in Shor's factoring algorithm. Quantum Inf. Process. **14**(3), 831–838 (2015)
11. Leander, G.: Improving the success probability for Shor's factoring algorithm. ArXiv:quant-ph/0208183 (2002)
12. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. **261**(4), 515–534 (1982)
13. Long, D.L.: Random equivalence of factorization and computation of orders. Technical report. Princeton University
14. Martín-López, E., Laing, A., Lawson, T., Alvarez, R., Zhou, X.-Q., O'Brien, J.L.: Experimental realization of Shor's quantum factoring algorithm using qubit recycling. Nat. Photonics **6**(11), 773–776 (2012)
15. Miller, G.L.: Riemann's hypothesis and tests for primality. J. Comput. Syst. Sci. **13**(3), 300–317 (1976)
16. Morain, F., Renault, G., Smith, B.: Deterministic factoring with oracles. ArXiv:1802.08444 (2018)
17. Pollard, J.M.: Theorems of factorization and primality testing. Math. Proc. Camb. Philos. Soc. **76**(3), 521–528 (1974)
18. Rabin, M.O.: Probabilistic algorithm for testing primality. J. Number Theory **12**(1), 128–138 (1980)
19. Rivest, R.L., Shamir, A., Adleman, L.: A Method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
20. Seifert, J.P.: Using fewer qubits in Shor's factorization algorithm via simultaneous Diophantine approximation. In: CT-RSA, Lecture Notes in Comput. Sci. (LNCS), vol. 2020, Springer, Berlin Heidelberg, pp. 319–327 (2001)
21. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: SFCS, proceedings of the 35th annual symposium on foundations of computer science, IEEE Computer Society, Washington, DC, pp. 124–134 (1994)
22. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997)
23. Xu, G., Qiu, D., Zou, X., Gruska, J.: Improving the success probability for Shor's factorization algorithm. In: reversibility and universality. Emergence, complexity and computation, vol. 30, Springer, Cham, pp. 447–462 (2018)