Electronic Colloquium on Computational Complexity, Report No. 212 (2026)

A Computational Perspective on Carmichael Numbers

Nikhil Gupta* Alan Sikarov† Ilya Volkovich‡

Abstract

We consider the problem of deterministically factoring integers provided with oracle access to important number-theoretic functions such as Euler's Totient function - $\phi(\cdot)$ and Carmichael's Lambda function - $\lambda(\cdot)$. We focus on *Carmichael* numbers - also known as *Fermat pseudoprimes*. In particular, we obtain the following results:

- Let N be a 'simple' Carmichael number with three prime factors (also known as simple C_3-numbers). Then, given oracle access to $\lambda(\cdot)$, we can completely factor N in deterministic polynomial time.
- There exists a deterministic polynomial-time algorithm that given oracle access to $\phi(\cdot)$, completely factors simple C_3-numbers, satisfying some 'size' bounds. Although in this case our methods do not provide a *theoretical* guarantee for all such numbers due to the required size bounds, we show *experimentally* that our algorithm can factor **more than** 99% of **all** simple C_3-numbers up to 10^{13}.

Our techniques extend the work of Morain, Renault, and Smith (*Applicable Algebra in Engineering, Communication, and Computation*, 2023), at the core of which sits the Coppersmith's method that provides an efficient way to find *bounded roots* of a bivariate polynomial over the integers. We combine these techniques with a new upper bound on $\gcd(N-1, \phi(N))$ for C_3-numbers, which could be of an independent interest.

AI Usage Statement

This paper was written without the use of ChatGPT, large language models (LLMs), or other generative AI tools.

1 Introduction

One of the most fundamental and important problems in computational number theory is *integer factoring*, whose objective is to find a non-trivial factor of an integer N. Since this problem is trivial for prime numbers, we assume that the input integer N is a composite number. The straightforward approach for this problem is to test if there exists an integer p between 2 and $\sqrt{N}$ that divides N and this yields an exponential-time algorithm for integer factoring [i]. The best-known algorithm for integer factoring given in [BLP93] has sub-exponential heuristic running time, provided that the input integer is sufficiently large. The inability to solve integer factoring efficiently strengthens

*Computer Science Department, Boston College, Chestnut Hill, MA. Email: nikhil.gupta.3@bc.edu
†Computer Science Department, Boston College, Chestnut Hill, MA. Email: asikarov@gmail.com
‡Computer Science Department, Boston College, Chestnut Hill, MA. Email: ilya.volkovich@bc.edu
[i]This approach is popularly known as *trial division* and was given by Fibonacci.

ISSN 1433-8092

the belief of many researchers that this is a computationally 'hard' problem. On the other hand, as the decision version of integer factoring lies in $\mathsf{NP} \cap \mathsf{co\text{-}NP}$ [ii], the decision version is *unlikely* to be complete for NP or $\mathsf{co\text{-}NP}$ [iii]. However, many researchers suspect that this is NP-intermediate [iv]. This (supposed) hardness of integer factoring is the main reason behind the success of the RSA cryptosystem [RSA78], which is an important component of modern-day cryptography. An efficient algorithm to factor a product of two prime numbers would break the security of RSA.

There have been many attempts towards resolving the integer factoring problem. Based on Fermat's factorization method, an algorithm given in [Leh74] factors an integer N in $O(N^{\frac{1}{3}})$ time. Strassen [Str77] gave a deterministic algorithm that factors N in time $O(N^{\frac{1}{4}+o(1)})$, and this runtime was improved to $\widetilde{O}(N^{\frac{1}{4}})$ in [BGS07, CH14, Har21]. A deterministic algorithm given in [Hit21] factors N in $O(N^{\frac{2}{9}+o(1)})$ time and [Har21] gave a deterministic algorithm that factors N in $O(N^{\frac{1}{5}+o(1)})$ time. Pollard's rho method [Pol75] is a randomized way to factor an integer N in time $O(\sqrt{p})$, where p is the smallest factor of N. Apart from the general scenario, there are also algorithms that factor integers having prime factors with some special properties. Pollard's $p-1$ algorithm [Pol74] efficiently factors every integer N such that for a prime factor p of N, $p-1$ has 'small' prime factors. William's $p+1$ algorithm given in [Wil82] gives an efficient way to factor every integer N, where N has a prime factor p such that $p+1$ has 'small' factors. Several algorithms with subexponential heuristic running time are known for this problem such as: the continued fraction method [MB75], the elliptic curve method [Len87], the quadratic sieve method [Pom85], and the general number field sieve method [BLP93], which is the fastest in this list. For more details, we direct readers to [Pom98, Len00, Fil12].

Many works efficiently solve special cases of integer factoring. Fermat's algorithm [dFTH94] efficiently factors odd integers of the kind $N = a^2 - b^2$, where N is a product of two prime numbers and each of them is close to $\sqrt{N}$. [BDHG99] showed that if $N = p^r q$, where p, q are distinct primes and $r \approx \log p$ then N can be factored in polynomial time. The numbers of the kind $p^r q$ have also been used in cryptography [OU98, Tak98]. The result of [BDHG99] was improved in [CZ18], where a polynomial-time factoring algorithm was given when $N = p^r q^s$ satisfying $r = \Omega(\log p)$. A polynomial-time factoring algorithm for integers of the kind $N = pq$, where $p \neq q$ are primes with small 'algebraic independence' was given in [ASS16].

Another important line of work on the integer factoring problem pertains to efficient algorithms having access to some number-theoretic oracles [Mil76, BMS86, Wol87, Lan88, HP20, KVZ20, DV21, RV23, MRS23]. The purpose of these oracles is to reveal 'useful information' about the integer N given as input to the integer factoring algorithm. On the practical side, such oracles can model extra information on N obtained by means of side-channel attacks. The following oracles have been widely used in the integer factoring algorithms:

1. The Euler's Totient oracle Φ: On input $N \in \mathbb{Z}$, it returns in unit time $\phi(N)$, *Euler's Totient function's value* of N (see Definition 2.1).

2. The Carmichael Lambda oracle Λ: This oracle takes input $N \in \mathbb{Z}$, and returns in unit time $\lambda(N)$, *Carmichael's Lambda function's value* of N (see Definition 2.3).

[ii] The decision version of integer factoring is the following problem: given $n, m \in \mathbb{N}$, decide if there exists a prime number $2 \leq p \leq m$ such that p divides n.

[iii] If the decision version of integer factoring turns out to be either NP-complete or $\mathsf{co\text{-}NP}$-complete then $\mathsf{NP} = \mathsf{co\text{-}NP}$, which is widely believed to be false.

[iv] Assuming $\mathsf{P} \neq \mathsf{NP}$, a decision problem in NP is said to be NP-intermediate if it is neither in P, nor NP-complete. Richard E. Ladner showed the existence of NP-intermediate problems [Lad75].

3. The order oracle $\mathcal{O}$: It takes input two co-prime integers N and a, and returns in unit time the minimum $d \in \mathbb{N}, d > 0$ such that $a^d \equiv 1 \mod N$.

4. The sum of divisors oracle Σ: On input $N \in \mathbb{Z}$, it returns in unit time $\sigma(N)$, the sum of all positive divisors of N.

It is a folklore result that given oracle access to Φ, we can factor any product of two distinct primes in polynomial time, deterministically (see Fact 2.2). The same result holds if we replace Φ with the Λ oracle (see Fact 2.6). Miller [Mil76] showed that assuming Extended Riemann Hypothesis (ERH), the problems of integer factoring, computation of $\phi(\cdot)$, and computation of $\lambda(\cdot)$ are Turing-reducible to each other in deterministic polynomial time (see Theorem 3 of [Mil76]). Subsequently, Long [Lon81] and Rabin [Rab80] observed that these three problems are **unconditionally** reducible to each other in *randomized* polynomial time (see Section 5 of [Lon81]). The main idea behind these results can be extracted as follows: let p be a prime factor of a given integer N and suppose that in addition to N we are also given an integer M such that $p - 1 \mid M$. Then there is a polynomial-time algorithm that outputs a non-trivial factor of N. The algorithm can be made deterministic assuming the ERH [Mil76] or randomized - unconditionally [Lon81, Rab80]. Definitions 2.1 and 2.3 imply that for every N, the values of $\phi(N)$ and $\lambda(N)$ are natural candidates for M. This idea is also useful for our results.

More broadly, Long also showed in [Lon81] that problems of integer factoring and the computation of order modulo N are reducible to each other in randomized polynomial time. The problem of computation of order of an element in a finite group has been widely studied in the classical [Sha71, Pol75, Sut07] and in quantum setting [Sho97]. To the best of our knowledge, the best classical algorithm to compute the order of an element a, denoted $|a|$, in a finite group runs in time $o(|a|^{\frac{1}{2}})$ [Sut07], whereas there exists a polynomial-time quantum algorithm to compute $|a|$ [Sho97]. This algorithm yields a polynomial-time quantum algorithm for factoring integers. It was shown in [BMS86] that the same holds true the problems of computing $\sigma(\cdot)$ (the sum of all positive divisors of N) and integer factoring. Thus in summary, the following five problems are reducible to each other in randomized polynomial time: a) integer factoring, b) computation of $\phi(\cdot)$, c) computation of $\lambda(\cdot)$, d) computing $\sigma(\cdot)$, and e) computation of order modulo N. This implies that given oracle access to any of the four oracles mentioned above, we can factor any integer in randomized polynomial time. The work of [Źr10] gave an explicit set of integers that can be efficiently factored with oracle access to Φ and also showed that using logarithmic many queries to Φ, we can design a deterministic sub-exponential algorithm for integer factoring. In [MRS23], Morain, Renault and Smith devised a deterministic polynomial-time algorithm that given oracle access to Φ or Λ or Σ, and an integer N having a large prime factor $p > \sqrt{N}$, finds p.

Can we deterministically factor a product of (merely) three primes with oracle access to one of the four oracles described above? This will extend the folklore result of deterministically factoring product of two primes with oracle access to Λ or Φ, and also take us one step forward towards derandomizing the results of [Lon81, BMS86]. In this direction, Morain et.al. gave a deterministic algorithm with oracle access to either Φ or Σ for factoring an $N = pqr$, where $p < q < r$ are primes and they satisfy some 'size' bounds (see [MRS23]).

Our work extends their result slightly for general integers (see Theorem 2), and mainly for an important sub-class of integers called *Carmichael numbers*. These are the composite numbers which satisfy Fermat's little theorem [v]. An equivalent description of a Carmichael number is given in

[v] A composite number n is Carmichael if for every $1 < a < N$ with $\gcd(a, n) = 1$, we have $a^n \equiv a \mod n$. Fermat's

Definition 2.7. These numbers are widely studied in mathematics for more than a century [Car10, Che39, Erd56, Leh76, AGP94, Pom92, Lar22]. There are infinitely many Carmichael numbers [AGP94]. Carmichael numbers are also important from the viewpoint of cryptography as they play the role of false-positives in many primality testing algorithms (see [Kar17]). It is natural to study computational questions related to them like determining if a given number is Carmichael, efficient factoring of a Carmichael number, etc.

A *randomized* polynomial-time algorithm for factoring Carmichael numbers was given in 1988 [BBC+88] (see also [Sam24]). This question has recently attracted renewed attention. In particular, a recent post on the Computational Complexity Blog [For25] noted, among other things, that the Miller–Rabin primality test (based on the aforementioned foundational works of [Mil76, Rab80]) can, in fact, be used to factor Carmichael numbers, and not merely determine that they are composite, in *randomized* polynomial time. Similar observations have previously appeared in the literature (see e.g. [Sta12]). It was further mentioned that "Gemini 3 has been released and introduced new capabilities" including the ability to produce a factoring approach for Carmichael numbers. Nonetheless, despite these recent developments, a *deterministic* polynomial-time algorithm for factoring Carmichael numbers remained elusive for nearly four decades. In fact, it is only recently [SW26] that a deterministic algorithm for **recognizing** Carmichael numbers has been developed. More precisely, the algorithm determines whether a given integer is a Carmichael number, although its running time is conditioned on the ERH. We note that an **unconditional** polynomial-time deterministic algorithm for this task remains unknown.

In our work, we consider the problem of factoring Carmichael numbers which are products of three distinct primes [vi]. Our results can be seen as a step forward towards obtaining a deterministic factoring algorithm for the whole set of Carmichael numbers. We also give a deterministic algorithm for factoring general products of three distinct primes satisfying some 'size-bounds', following and slightly improving the result of [MRS23] (see Theorem 2). We believe that this result is interesting in its own right.

1.1 Our results

Let C_3 be the set of all Carmichael numbers with exactly 3 prime factors [vii]. $C_3 \neq \emptyset$ as $561 \in C_3$. Yet, whether C_3 is finite remains open. It is conjectured that for every $m \geq 3$, there are infinitely many Carmichael numbers having exactly m prime factors (or C_m-numbers). Chernick showed in [Che39] that if *Dickson's k-tuple conjecture* is true then C_m is infinite for every $m \geq 3$ and Wright showed that the same result holds if a weaker version of Dickson's k-tuple conjecture is true [WRI16, Wri19]. C_3 is a well-studied set [Che39, BN97, Jam11].

It was shown by Carmichael in 1910 [Car10] that N is a Carmichael number if and only if $N \equiv 1 \mod \lambda(N)$, where $\lambda(\cdot)$ is the Carmichael's Lambda function. This immediately implies that there exists a polynomial-time algorithm that decides if the input integer is Carmichael, given oracle access to Λ. There also exists a randomized polynomial-time algorithm for testing whether a number is Carmichael or not, which follows from an efficient primality testing algorithm [Rab80, AB03, AKS04] and Fermat's little theorem (see [Kar17] in this regard). Our work is related to the deterministic factoring of C_3-numbers with oracle access to Λ or Φ.

---

little theorem is stated for prime numbers. Yet, as Carmichael numbers also satisfy this theorem, they fool primality tests based on Fermat's little theorem and hence are known as Fermat's pseudoprimes (see [Kar17]).

[vi] Every Carmichael number has at least three prime factors (See Fact 2.9).

[vii] We have borrowed the notation C_3-numbers from [Jam11].

Before listing our contributions, we give some useful notations. Let $N \in C_3$ be such that $N = pqr$, where $p < q < r$ are prime numbers. Let $g = \gcd(p-1, q-1, r-1)$. Then, there exist $a, b, c \in \mathbb{N}$ such that $p = ag + 1, q = bg + 1$, and $r = cg + 1$. We say that N is *simple* iff $a = 1$. Alternatively, N is simple iff $p - 1 \mid q - 1$ and $p - 1 \mid r - 1$. Some useful properties of simple C_3-numbers are given in [Jam11]. Chernick gave the following interesting classification of a subclass of C_3-numbers in [Che39] - if $N = (6k+1)(12k+1)(18k+1)$, where $6k+1, 12k+1, 18k+1$ are primes then N is Carmichael. Observe that each number of this kind is a simple C_3-number. We observed that in the list of first $10,000$ Carmichael numbers given in [Slo], there are 1166 C_3-numbers and 786 simple C_3-numbers, which constitutes more than 67% of all C_3-numbers in this list. Thus, it is natural to study factoring of simple C_3-numbers. In the first result, we obtain deterministic polynomial-time factoring of simple C_3-numbers given oracle access to Λ. We focus on simple C_3-numbers as it is not clear to us how oracle access to Λ yields a deterministic algorithm to factor all C_3-numbers.

Lemma 1.1 (Deterministic factoring of simple C_3-numbers with oracle access to Λ)**.** *There exists a deterministic algorithm that given a simple C_3-number N and oracle access to Λ completely factors N, in time* $\mathrm{poly}(\log N)$.

This lemma is proved in Section 3. As noted before, we know that the problems of factoring an integer, computing the Euler's Totient function, and computing the Carmichael's Lambda function are reducible to each other in *randomized* polynomial time. However, we do not know if this reduction can be made deterministic. In the absence of such a result, it is not clear if we get a deterministic algorithm to factor simple C_3-numbers on replacing Λ with Φ in Theorem 1.1. In the following theorem, we give a deterministic algorithm that factors simple C_3-numbers satisfying certain 'size' bounds. This is similar to Theorem 2 of [MRS23], which gives a deterministic polynomial-time algorithm to factor general integers obeying some 'size' bounds. Our results do not follow from [MRS23]. We give a comparison of our results with theirs in Section 1.4.

Theorem 1 (Factoring algorithm with oracle access to Φ for simple C_3-numbers)**.** *Let $N = pqr$ be a simple C_3-number, where $p < q < r$ are primes. Let $\alpha_1 := \log_N r$ and $\alpha_2 := \log_N q$. Suppose N satisfies one of the following bounds. Then, there exists a deterministic algorithm that takes oracle access to Φ and completely factors N in* $\mathrm{poly}(\log N)$ *time.*

1. $\sqrt{\alpha_2(1-\alpha_1)} < \alpha_1$.
2. $\sqrt{(1-\alpha_1)(3-3\alpha_1-4\alpha_2)} < \alpha_1$.
3. $\alpha_1 + \alpha_2 < \frac{3}{4}$.
4. $\sqrt{(1-\alpha_2)(1-\alpha_1-\alpha_2)} < \alpha_2$.

It is proved in Section 4. In the following result, we improve Theorem 2 of [MRS23] by appending the 'size' bound given in Point 3. The proof follows from their result and Lemma 4.11. We highlight that the following result holds for a general product of three distinct prime numbers and not just C_3-numbers.

Theorem 2 (Improved factorization for general squarefree product of three primes)**.** *Let $N = pqr$ be a squarefree number where $p < q < r$ are primes. Let $\alpha_1 := \log_N r, \alpha_2 := \log_N q$. Suppose N satisfies one of the following bounds. Then, there exists a deterministic algorithm that takes oracle access to Φ and completely factors N in* $\mathrm{poly}(\log N)$ *time.*

1. $\alpha_1 > 0.5$
2. $2\alpha_1 + 3\alpha_2 \geq 2$
3. $\sqrt{(1-\alpha_2)(1-\alpha_1-\alpha_2)} < \alpha_2$.

We provide a comparison of our result with the result from [MRS23] in Section 1.4.

1.2 Discussion of the results

While it may seem that "trading" randomness for oracle access to Φ or Λ is not a favorable exchange, since these functions are themselves computationally hard, the traditional "hardness versus randomness" paradigm [NW94, IW97] in fact leverages the output of *provably* hard functions to derandomize randomized algorithms! In this light, we make the following two remarks.

1. Our result achieves derandomization using functions that are *suspected* to be hard, but not yet *provably* so. Consequently, our result can be viewed as additional evidence supporting the computational hardness of these functions.
2. While, as discussed earlier, there exist *randomized* algorithms that use the power of Φ or Λ to factor integers, the core difficulty addressed in this work is the lack of a known *deterministic* methods to "extract" this computational power.

1.3 Techniques and proof overview

Coppersmith's method. In [Cop96], Coppersmith gave a deterministic polynomial time algorithm to find *bounded* roots of a bivariate polynomial over $\mathbb{Z}$, provided some technical conditions are satisfied. This result is broadly known in the literature as the *Coppersmith's method* (see Lemma 2.13 for the formal statement). As was observed by Morain et.al. in [MRS23], Coppersmith's method gives rise to the following conditional deterministic polynomial-time algorithm for integer factoring: Suppose we are given $N, M \in \mathbb{N}$ such that there exists a D satisfying $D \mid N$ and $D-1 \mid M$ [viii]. If there exists a $t > 0$ such that $N^t M^{\frac{1}{t}} < D^{2+t+\frac{1}{t}}$ then we can compute a non-trivial factor of N in polynomial time (see Corollary 2.15). The deterministic polynomial-time conditional integer factoring algorithms in [MRS23] were obtained by using the special case when $t = 1$. In our work we are using the root finding algorithm in its full generality.

Proof overview. We now discuss the main ideas used to obtain our results. First, we prove Lemma 2.16 which is a generalization of Corollary 2.15 and plays an important role in the proof of Theorem 1. Formally, let $N, M \in \mathbb{Z}$ be such that for some $D \in \mathbb{N}$ and $\ell \geq 1$ we have that $D \mid N$ and $(D-1)^\ell \mid M$. Suppose $M \approx N^\theta$ for some $\theta > 0$. Then, using the Coppersmith's method, we show that if $N^{\frac{\theta}{\theta+\ell}} < D$ then we can compute a non-trivial factor of N in polynomial time. Thus, the whole story revolves around obtaining a suitable M and it is at this point when the two functions - the Euler's Totient function and the Carmichael's Lambda function - play their roles.

We are dealing with simple C_3-numbers (Definition 2.12). Let N be the input simple C_3-number. Then, $N = pqr$, where $p < q < r$ are primes. Let $g = \gcd(p-1, q-1, r-1)$ and $b, c \in \mathbb{N}$ such that

[viii] this framework had been previously outlined in the works of Miller and Long [Mil76, Lon81].

$q = bg + 1$ and $r = cg + 1$. Observe that $N = \Theta(bcg^3)$. For the rest of this section, we ignore the constants and take $N = bcg^3$.

Let us first talk about Lemma 1.1. Definition 2.3, $\lambda(N) = abcg$. We compute the ratio $\frac{N-1}{\lambda(N)}$. By Lemma 2.11 this is an integer number sandwiched between two consecutive perfect squares: g^2 and $(g+1)^2$. This allows to recover g. Once we have g the problem reduces to the case of two prime factors.

Now let us turn to Theorem 1. As mentioned in Section 1.1, it is currently unknown whether the computations of Euler's Totient function (see Definition 2.1) and Carmichael's Lambda function are reducible to each other in deterministic polynomial time. Consequently, Theorem 1.1 does not immediately yield a deterministic polynomial-time algorithm with oracle access to Φ for factoring a simple C_3 numbers. In particular, given that $\phi(N) = abcg^3$, it is not clear how to efficiently "extract" the expression $abcg$ from it.

As a result, we obtain a deterministic algorithm for factoring simple C_3 numbers satisfying certain 'size' bounds specified in Theorem 1. While the previous approach cannot be directly applied, since the ratio $\frac{N-1}{\phi(N)}$ lies between 1 and $(1 + 1/g)^2$, we observe that $\lambda(N)$ is a common factor of both $N - 1$ and $\phi(N)$. Hence, by definition, it divides their gcd. This motivates the consideration of the parameter $M = \gcd(N - 1, \phi(N))$ (as mentioned above). We then have that $M = \gcd(N - 1, \phi(N)) = abcg \cdot \gcd\left(\frac{N-1}{abcg}, g^2\right)$. The main challenge is to derive an effective and useful upper bound for the latter expression.

Using the generalization of Corollary 2.15 mentioned earlier, we obtain one of the size bounds stated in Theorem 1. Specifically, if the input integer N is a simple C_3 number satisfying these size conditions, we can compute a nontrivial factor of N in polynomial time. We then apply Fact 2.2 to completely factor N in deterministic polynomial time. Although our algorithm does not give a theoretical guarantee to factor all simple C_3-numbers, we show experimentally in Section B that this algorithm completely factors more than 99% of the simple 787 C_3-numbers up to 1.7×10^{13}. Indeed, as it turns out that there are only four simple C_3-numbers in this list which are not factored by our algorithm.

1.4 Comparison with [MRS23]

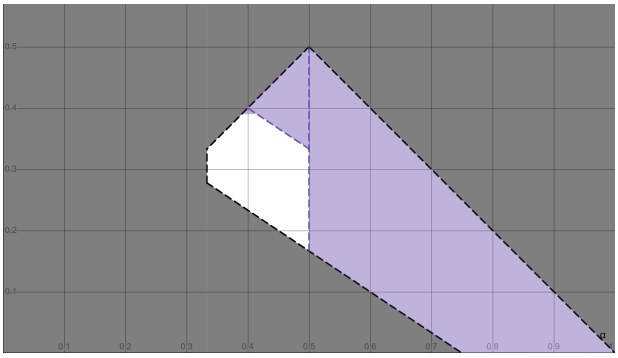
In this section we compare our result with the previous result of [MRS23]. We provide further comparison and experimentation in Section B. Let $N = pqr$ be a product of three primes $p < q < r$ and define $\alpha_1 := \log_N r$ and $\alpha_2 := \log_N q$. We restate the main result of [MRS23]:

Lemma 1.2 ([MRS23])**.** *There exists a deterministic algorithm that given N and oracle access to Φ completely factors N, in time* $\mathrm{poly}(\log N)$*, if at least one of the following conditions hold:*

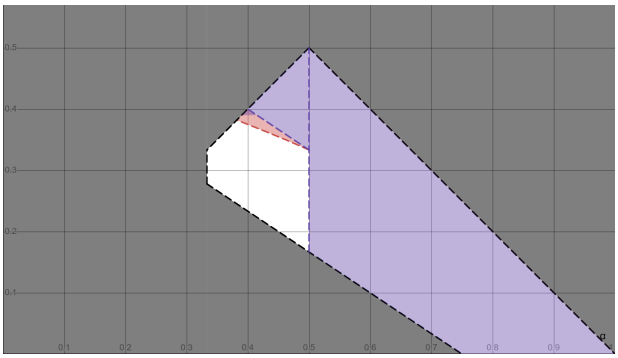
1. $\alpha_1 > 0.5$
2. $2\alpha + 3\alpha_2 \geq 2$
3. $\alpha_2 > (-1 + \sqrt{17})/8$

Given the additional conditions that α_i-s must satisfy (e.g. $\alpha_1 > \alpha_2$, $\alpha_1 > 1/3$, etc), the authors presented a detailed region of (α_1, α_2) which corresponds to the cases covered by this result. That is, their result cover the area in purple in Figure 1a, however, do not (provably) apply to the region in white. In our work, Theorem 2 allows to slightly extend the coverage area of

[MRS23]. In particular, the extension is given by the red area in Figure 1b and our theorem also captures the result of [MRS23] colored in purple.



(a) The pictorial representation of the result of [MRS23]



(b) The pictorial representation of Theorem 2

Figure 1: Comparison between Theorem 2 and the result of [MRS23]

2 Preliminaries

2.1 Euler's Totient function and Carmichael's Lambda function

Definition 2.1. (Euler's Totient function): *Let $n \in \mathbb{N}$. Then, the Euler's Totient function of n, denoted $\phi(n)$, is defined as $\phi(n) = |\{m : m \in \mathbb{N}, m \leq n, \gcd(m,n) = 1\}|$. Suppose $n = p_1^{e_1} \cdots p_k^{e_k}$, where $p_1, \ldots, p_k$ are distinct prime numbers and $e_1, \ldots, e_k \in \mathbb{N}$. Then, $\phi(n) = p_1^{e_1-1} \cdots p_k^{e_k-1} \times (p_1 - 1) \cdots (p_k - 1)$.*

The following fact shows that if we have oracle access to Φ (which computes $\phi(n)$ for any n in a unit time) then we can efficiently factor a product of two primes. Although a folklore result, we give a proof of this for the sake of completeness in Section A of the appendix.

Fact 2.2. (Folklore). *Let $N = pq$, where $p \neq q$ are primes. Then, there exists a deterministic algorithm that given N and oracle access to Φ, outputs p, q, in time* $\mathrm{poly}(\log N)$.

Definition 2.3. (Carmichael's Lambda function). *Let $N \in \mathbb{N}$. Then, the value of Carmichael's Lambda function, denoted $\lambda(N)$, is recursively defined as follows:*

$$\lambda(N) = \begin{cases} \phi(N) & \text{if } N \in \{1, 2, 4\}, \\ \phi(N) & \text{if } N = p^e, \text{ where } p \text{ is an odd prime}, \\ \frac{1}{2}\phi(N) & \text{if } N = 2^e \text{ and } e > 2 \\ \mathrm{lcm}(\lambda(p_i^{e_i}))_{i=1}^k & \text{if } N = \prod_{i=1}^k p_i^{e_i}, \ p_i\text{'s are distinct primes.} \end{cases}$$

Observation 2.4. *Let $n \in \mathbb{N}$. Then, $\lambda(n)$ divides $\phi(n)$.*

This observation follows from Definitions 2.1 and 2.3. The following fact gives a clear relationship between $\phi(n)$ and $\lambda(n)$ when n is a product of two distinct primes.

Fact 2.5. ([MRS23]). *Let $n \in \mathbb{N}$ be such that $n = pq$, where $p < q$ are primes. Then, $\phi(n) = \lambda(n) \cdot \gcd(n-1, \lambda(n))$.*

Recall the oracle Λ from Section 1. Then Facts 2.5 and 2.2 imply the following.

Fact 2.6. *Let $N = pq$, where p, q are distinct prime numbers. Then, there exists a deterministic algorithm that given N and oracle access to Λ, outputs p, q, in time* $\mathrm{poly}(\log N)$.

2.2 Carmichael Numbers

Prime numbers are the building blocks of integers and are one of the most important objects of study in mathematics. One of the many interesting properties that prime numbers satisfy is the so-called *Fermat's little theorem*, which is as follows: Let p be a prime number. Then, for every integer a we have that $a^p \equiv a \pmod{p}$. However, there are also composite numbers which satisfy a similar property and such numbers are called *Carmichael numbers* or *Fermat pseudoprimes*. It is a composite number n that satisfies $a^n \equiv a \pmod{n}$ for every integer a satisfying $\gcd(a, n) = 1$. There is an equivalent definition of Carmichael numbers due to Korselt and we consider this definition in our work.

Definition 2.7. (Korselt's criterion): *A composite number n is Carmichael if and only if n is squarefree and for every prime factor p of n, $p-1$ divides $n-1$.*

Although this criterion was given by Alwin Korselt in 1899, concrete examples of such numbers were first discovered by Robert Carmichael only in 1910 [ix]! The smallest Carmichael number is 561 as $561 = 3 \cdot 11 \cdot 17$. Carmichael showed that every Carmichael number is odd and has at least three factors. Subsequently, in [AGP94], Alford, Granville and Pomerance have shown that there are infinitely many Carmichael numbers. One can also express Korselt's criterion using Carmichael's Lambda function:

Fact 2.8. *An integer $n \in \mathbb{N}$ is a Carmichael number if and only if $\lambda(n) \mid n-1$.*

The following well-known fact is a corollary of Korselt's criterion. For the sake of completeness, we provide a proof in Section A of the appendix.

Fact 2.9. *Every Carmichael number n is odd, has at least three distinct prime factors, and every prime factor p of n satisfies $p < \sqrt{n}$.*

2.2.1 Carmichael Numbers with 3 Prime Factors

Recall that C_3 is the set of all Carmichael numbers with three prime factors. As noted above, $561 \in C_3$, which implies $C_3 \neq \emptyset$. Yet, whether C_3 is finite remains an open question. We direct interested readers to an excellent survey [Jam11] on C_3 numbers.

Let $N \in C_3$. Then, $N = pqr$, where p, q, r are primes and $p < q < r$. Let $g = \gcd(p-1, q-1, r-1)$. Then, there exist $a, b, c \in \mathbb{N}$ such that $p = ag + 1, q = bg + 1, r = cg + 1$. Henceforth, we fix these notations.

[ix] Actually, the first Carmichael numbers were discovered already in 1885 by Václav Šimerka, however this discovery remained unnoticed.

Fact 2.10. ([Jam11]). *Let $N \in C_3$ and $N = pqr$, where $p = ag + 1, q = bg + 1, r = cg + 1$, and $g = \gcd(p - 1, q - 1, r - 1)$. Then, $\gcd(a, b) = \gcd(b, c) = \gcd(a, c) = 1$ and $1 \leq a < b < c$.*

Observe that $\lambda(N) = abcg$. Furthermore, we obtain the following relation between N and $\lambda(N)$:

Lemma 2.11. *Let $N \in C_3$. Then $\frac{N-1}{\lambda(N)}$ is an integer number satisfying: $g^2 < \frac{N-1}{\lambda(N)} \leq (g + 1)^2$.*

Proof. It follows from Fact 2.8 that (any) N is Carmichael iff the ratio $\frac{N-1}{\lambda(N)}$ is an integer number. Furthermore,

$$\frac{N-1}{\lambda(N)} = \frac{(ag+1)(bg+1)(cg+1)}{abcg} = \frac{1}{g} \cdot \left(g + \frac{1}{a}\right)\left(g + \frac{1}{b}\right)\left(g + \frac{1}{c}\right)$$

$a, b, c > 0$ we obtain the lower bound. However, in fact, since $c > b > a \geq 1$ we have that $c \geq 3, b \geq 2$. Therefore, the expression is maximized when $c = 3, b = 2, a = 1$ which results in

$$\frac{1}{g} \cdot (g+1)\left(g + \frac{1}{2}\right)\left(g + \frac{1}{3}\right) = (g+1)\left(g + \frac{5}{6} + \frac{1}{6g}\right) \leq (g+1)^2$$

□

Definition 2.12. (Simple C_3-numbers) [Jam11] *Let $N = pqr \in C_3$. Let $g = \gcd(p-1, q-1, r-1)$. Then, N is simple if there exist $b, c \in \mathbb{N}$ such that $p = g + 1, q = bg + 1$, and $r = cg + 1$.*

In other words, $p - 1 \mid q - 1$ and $p - 1 \mid r - 1$.

2.3 Coppersmith's Method and Its Applications

The following gives a deterministic algorithm to compute *bounded* roots of a bivariate polynomial over $\mathbb{Z}$.

Lemma 2.13 ([Cop96]). *Let $f = \sum_{i,j} f_{i,j} x^i y^j$ be an irreducible polynomial over $\mathbb{Z}$ such that $\deg_x(f) = d$ and $\deg_y(f) = e$. Let X, Y be bounds on the desired $|x_0|$ and $|y_0|$ such that $f(x_0, y_0) = 0$. Let $W := \max_{i,j} |f_{i,j}| X^i Y^j$. Suppose there exists a $t > 0$ such that*

$$X^{d + \frac{et}{2}} \cdot Y^{e + \frac{d}{2t}} < W \cdot 2^{-3(d^2+e^2)-2}. \tag{1}$$

Then, there exists an algorithm that takes input f and outputs all $(x_0, y_0) \in \mathbb{Z}^2$ such that $f(x_0, y_0) = 0$ and $|x_0| < X, |y_0| < Y$. The running time of the algorithm is $\mathrm{poly}(d, e, \log W)$.

We remark that the algorithm **does not** need to know the value of t; mere existence is sufficient. The following lemma shows that we can scale W with arbitrary natural number in the R.H.S. of the inequality (1) by paying an appropriate price in the running time of the algorithm of Lemma 2.13. The proof of Lemma 2.14 implicitly follows from a collection of works [Cop96, BM05, May09, CKT13].

Lemma 2.14. *Let f, d, e, X, Y, and W be as given in Lemma 2.13. Let $u > 0$. Suppose there exists a $t > 0$ such that $X^{d + \frac{et}{2}} \cdot Y^{e + \frac{d}{2t}} < W \cdot u$. Then, there exists an algorithm that takes input f, u and outputs all $(x_0, y_0) \in \mathbb{Z}^2$ such that $f(x_0, y_0) = 0$ and $|x_0| \leq X, |y_0| \leq Y$. The running time of the algorithm is $\mathrm{poly}(d, e, u, 2^{O(d^2+e^2)}, \log W)$.*

Next, we note some useful consequences of this result. A variant (when $t = 1$ and $u = 1$) of the following corollary has already been proven in [MRS23]. Here we state the general case.

Corollary 2.15. *Let $N, M, D \in \mathbb{N}$ such that N is composite, $D \mid N$ and $D - 1 \mid M$. Let $u > 0$. Suppose there exists a constant $t > 0$ s.t. $N^t \cdot M^{1/t} < u \cdot (D-1)^{2+t+\frac{1}{t}}$. Then, there is a deterministic algorithm that given N, M, and u, finds a non-trivial factor of N in time* $\text{poly}(\log N, \log M, u)$.

Proof. Let x, y be variables and $f := yN - yx - Mx$. Then, it is easy to verify that f is irreducible over $\mathbb{Z}$. Let $x_0 := \frac{N}{D}, y_0 := \frac{M}{D-1}$, $X := \frac{N}{D-1}$ and $Y := \frac{M}{D-1}$. Then, observe that $x_0 \leq X, y_0 \leq Y$, and $f(x_0, y_0) = 0$. Let $d = \deg_x(f)$ and $e = \deg_y(f)$. Then, $d = e = 1$. Let W be the parameter mentioned in Lemma 2.13. Then, note that $W = \frac{NM}{D-1}$. Suppose there exists a $t > 0$ such that $N^t \cdot M^{1/t} < u \cdot (D-1)^{2+t+\frac{1}{t}}$ which happens if and only if $\left(\frac{N}{D-1}\right)^{1+\frac{t}{2}} \left(\frac{M}{D-1}\right)^{1+\frac{1}{2t}} < \sqrt{u} \cdot \frac{NM}{D-1}$. Thus, $X^{1+\frac{t}{2}} \cdot Y^{1+\frac{1}{2t}} < \sqrt{u} \cdot W$. Lemma 2.14 implies that we can find a non-trivial factor of N in time $\text{poly}(\log N, \log M, u)$.

□

Lemma 2.16. *Let $N, M \in \mathbb{N}$, N be composite and $M = \alpha \cdot N^\theta$, where $\alpha = \text{poly}(\log N)$. Suppose there exist $D, \ell \in \mathbb{N}$ such that ℓ is a constant, $D \mid N$ and $(D-1)^\ell \mid M$. Suppose $N^{\frac{\theta}{\theta+\ell}} < D - 1$. Then, there exists an algorithm that takes $M, N, \alpha, \theta, \ell$ as input and finds a non-trivial factor of N in time* $\text{poly}(\log N, \log M, \alpha^{\frac{\ell}{2\theta}})$.

Proof. Let x, y be variables and $f = y(N - x)^\ell - Mx^\ell$. We claim that f is irreducible over $\mathbb{Z}$. Suppose not. Then, $f = g \cdot h$, where $g, h \in \mathbb{Z}[x, y]$ are non-constant polynomials. As $\deg_y(f) = 1$, there exists exactly one polynomial amongst g and h which contains y. Without loss of generality, suppose y is present in g. Then, $\deg_y(g) = 1$, which means that we can write g as $g = g_1 y + g_0$, where $g_1, g_0 \in \mathbb{Z}[x]$. Then, $g_1 y h + g_0 h = y(N - x)^\ell - Mx^\ell$. This immediately implies that $g_1 h = (N - x)^\ell$ and $g_0 h \equiv -Mx^\ell$. As h is non-constant, clearly $h = (N - x)^r$ for some $1 \leq r \leq \ell$. Then, $g_0(N - x)^r \equiv -Mx^\ell$, which implies $\deg_x(g_0) = \ell - r$. Observe that $g_0(N - x)^r$ contains a non-zero x-monomial having degree less than ℓ. This implies that $g_0 h \not\equiv -Mx^\ell$ and f is not reducible.

We are given that $D \mid N$ and $D - 1 \mid M$. Let $d := \deg_x(f), e := \deg_y(f)$. Then, $d = \ell$ and $e = 1$. Let $x_0 := \frac{N}{D}, y_0 := \frac{M}{(D-1)^\ell}, X := \frac{N}{D-1}$, and $Y := \frac{M}{(D-1)^\ell}$. It is not difficult to verify that $f(x_0, y_0) = 0$. Note that $x_0 \leq X$ and $y_0 \leq Y$. Let W be the parameter defined in Lemma 2.13. Then, by definition of W and on expanding f, we get $W = \max\left\{YN^\ell, \binom{\ell}{1} YN^{\ell-1}X, \ldots, \binom{\ell}{i} YN^{\ell-i}X^i, \ldots, YX^\ell, MX^\ell\right\}$. On substituting the values of X and Y, we get

$$W = \max\left\{\frac{MN^\ell}{(D-1)^\ell}, \ldots, \binom{\ell}{i}\frac{MN^\ell}{(D-1)^{\ell+1}}, \ldots, \frac{MN^\ell}{(D-1)^{2\ell}}, \frac{MN^\ell}{(D-1)^\ell}\right\}.$$

Let $W' = \frac{MN^\ell}{(D-1)^\ell}$. Observe that $W' \leq W$. Suppose $N^{\frac{\theta}{\theta+\ell}} < D - 1$. This implies $N^{\frac{\theta+\ell}{2}} < (D-1)^{\frac{(\theta+\ell)^2}{2\theta}}$. Thus, $N^{\frac{\theta}{2}+\frac{\ell}{2}} < (D-1)^{\ell+\frac{\theta}{2}+\frac{\ell^2}{2\theta}}$. On multiplying by $\alpha^{\frac{\ell}{2\theta}}$, we get $N^{\frac{\theta}{2}}(\alpha N^\theta)^{\frac{\ell}{2\theta}} < (D-1)^{\ell+\frac{\theta}{2}+\frac{\ell^2}{2\theta}} \cdot \alpha^{\frac{\ell}{2\theta}}$. On multiplying both sides by MN^ℓ and using $M = \alpha N^\theta$, we get $\left(\frac{N}{D-1}\right)^{\ell+\frac{\theta}{2}} \left(\frac{M}{(D-1)^\ell}\right)^{1+\frac{\ell}{2\theta}} < \frac{MN^\ell}{(D-1)^\ell}$. On plugging $X = \frac{N}{D-1}, Y = \frac{M}{(D-1)^\ell}$, and $W = \frac{MN^\ell}{(D-1)^\ell}$, we get $X^{\ell+\frac{\theta}{2}} Y^{1+\frac{\ell}{2\theta}} < W' \cdot \alpha^{\frac{\ell}{2\theta}}$. As $W' \leq W$, $X^{\ell+\frac{\theta}{2}} Y^{1+\frac{\ell}{2\theta}} < W \cdot \alpha^{\frac{\ell}{2\theta}}$. Now, it follows from Lemma 2.14 that we can find a non-trivial factor of N in time $\text{poly}(\log N, \log M, \alpha^{\frac{\ell}{2\theta}})$.

□

The proof of the following observation is given in Section A of the appendix.

Observation 2.17. *Let $f(t) := At + \frac{B}{t}$, where A, B are non-negative constants. Then, f gets minimized when $t = \sqrt{\frac{B}{A}}$ and the minimum value f for any substitution of t is $2\sqrt{AB}$.*

3 Proof of Lemma 1.1

Let $N = pqr$ be a simple C_3-number, where $p < q < r$. Let $g := \gcd(p-1, q-1, r-1), b, c \in \mathbb{N}$, such that $p = g+1, q = bg+1$, and $r = cg+1$. We want to show that given N and oracle access to Λ, we can factor N in $\text{poly}(\log N)$ time. We consider $S = \frac{N-1}{\lambda(N)}$ and compute $\hat{g}$ as the integer part of $\sqrt{S}$. This can be carried out in time $\text{poly}(\log N)$ using a binary search. By Lemma 2.11, $\hat{g} = g$. Once we recovered g, we can "remove" p from N by computing $N' = N/(g+1)$. At this point we are given a number which is a product of two primes and the claim follows from Fact 2.6.

4 Factoring simple C_3-numbers with oracle access to Φ

We study factoring of simple C_3-numbers with oracle access to Φ, which on input N returns $\phi(N)$ in a unit time. We first present some results required for proving Theorem 1.

4.1 Useful results

We first talk about general C_3-numbers. Let $N \in C_3$. Then, $N = pqr$, where $p < q < r$ are prime numbers. Let $p = ag+1, q = bg+1$, and $r = cg+1$, where $g := \gcd(p-1, q-1, r-1)$. Then, by definition, $N = \Theta(abcg^3)$. Suppose we could somehow efficiently compute an $M \in \mathbb{N}$, where $M = N^\theta$ and $(r-1)^\ell \mid M$ for some $\ell \in \mathbb{N}$ such that $\frac{\theta}{\theta+\ell} \leq \frac{1}{3}$. Then, as r is the largest prime factor of $N, N^{\frac{1}{3}} < r$. Now, it follows from Lemma 2.16 that we can get hold of a factor D of N and by applying Fact 2.2 on $\frac{N}{D}$, we get the complete factorization of N. So, we hunt for such an M. In search of a number that is divisible by $r-1$, the first option that comes to the mind is $M := N-1$. As N is Carmichael, by definition $r-1 \mid M$. Then, clearly $M = \alpha \cdot N$, for some $\alpha < 1$. Recall the parameters θ and ℓ mentioned in Lemma 2.16. Then, in this case, $\theta = \ell = 1$. Then, $\frac{\theta}{\theta+\ell} = \frac{1}{2}$. If $r = \Theta(N^{\frac{1}{2}})$ then we can factorize N. But this does not happen as every prime factor of a Carmichael number N is less than $\sqrt{N}$ (see Fact 2.9). The second candidate for M is $\phi(N)$. Definition 2.1 implies

$$\phi(N) = abcg^3. \tag{2}$$

Then, $M = \alpha \cdot N$ for some constant α. Here also $\theta = \ell = 1$, where θ and ℓ are the parameters of Lemma 2.16 and we get the same result as in the previous case when $M = N-1$. Our third candidate is $M := \gcd(N-1, \phi(N))$. It follows from the above discussion that $r-1 \mid M$. The definition of N implies

$$N - 1 = abcg^3 + g^2(ab+bc+ca) + g(a+b+c). \tag{3}$$

Fact 4.1 ([Jam11])**.** *$\exists k \in \mathbb{N}$ s.t. $kabc = g(ab+bc+ca) + a + b + c$.*

Equations (2) and (3), and the fact above implies the following.

Corollary 4.2. *Let $k \in \mathbb{N}$ be the parameter mentioned in Fact 4.1 and $M = \gcd(N-1, \phi(N))$. Then, $N - 1 = abcg(g^2 + k)$ and $M = abcgk_0$, where $k_0 := \gcd(g^2, k)$.*

We now prove some useful upper bounds on k_0. The observation below follows from the definition of k_0.

Observation 4.3. ([Jam11]). *$k_0 \leq 3g$.*

Fact 4.4. *For any $a, b \geq 0$ we have that $\min\{a, 2b\} \leq 2\min\{a, b\}$.*

Observation 4.5. *Let $r, s \in \mathbb{N}$. Then, $\gcd(r, s^2) \mid \gcd(r, s)^2$.*

Proof. Let $p_1, \ldots, p_m$ be the prime numbers such that each p_i divides either r or s. Then, $r = p_1^{d_1} \cdots p_m^{d_m}$ and $s = p_1^{e_1} \cdots p_m^{e_m}$, where for every $i \in [m], d_i, e_i \geq 0$. Then, $\gcd(r, s^2) = \prod_{i \in [m]} p_i^{\min\{d_i, 2e_i\}}$ and by Fact 4.4, we get $\gcd(r, s^2) \mid \gcd(r, s)^2$. □

Proposition 4.6. *Let a, b, k_0 be as mentioned above. Then, $k_0 = O\left(\frac{a^2 g^2}{b^2}\right)$.*

Proof. Recall that $k_0 = \gcd(k, g^2)$. We know from Observation 4.5 that $\gcd(k, g^2) \mid \gcd(k, g)^2$. Note that $\gcd(k, g) \leq \gcd(ka, g)$. Fact 4.1 implies

$$\gcd(k, g) \leq \gcd(ka, g) = \gcd\left(g\left(1 + \frac{a}{b} + \frac{a}{c}\right) + \frac{a+b+c}{bc}, g\right).$$

As $b < c$, $\frac{ga}{c} < \frac{ga}{b}$. Thus, $\gcd(k, g) \leq \gcd\left(\frac{ga}{b} + \frac{ga}{c} + 3, g\right) \leq \frac{2ga}{b} + 3$. This immediately implies $k_0 = O\left(\frac{a^2 g^2}{b^2}\right)$.

□

Now we restrict the discussion to simple C_3-numbers (Definition 2.12). For the following results, let $N = pqr$ be a simple C_3-number, and $p = g + 1, q = bg + 1, r = cg + 1$ be primes, where $g = \gcd(p-1, q-1, r-1)$ and $b, c \in \mathbb{N}, b < c$. Note that if $g \leq \log N$ then we can trivially compute p in $O(\log N)$ time and then using Fact 2.2, we can completely factorize N. So we assume that $g > \log N$. Let $\varepsilon_b := \log_g b, \varepsilon_c := \log_g c$, and $\gamma := \log_g k_0$. As $k_0 \leq 3g, \gamma \leq 1 + \log_g 3$.

Lemma 4.7. *Let $\zeta \geq 0$ and suppose $\sqrt{(\varepsilon_b + 2)(\varepsilon_b + \gamma)} < \varepsilon_c + 1 + \zeta$. Then, there exists an algorithm that takes input N and ζ, oracle access to Φ and outputs all the factors of N in time $\mathrm{poly}(\log N, g^\zeta)$.*

Proof. Let $M := \gcd(N-1, \phi(N))$. Corollary 4.2 implies $M = bcgk_0$, where $k_0 = \gcd(g^2, k)$. Let $D := cg + 1$. We claim that there exists a $t > 0$ such that $N^t M^{\frac{1}{t}} < 8^t \cdot 4^\zeta \cdot (D-1)^{2+t+\frac{1}{t}}$. As $N \leq 8bcg^3$, this inequality holds if $(8bcg^3)^t (bcgk_0)^{\frac{1}{t}} < 8^t \cdot g^{2\zeta} \cdot (cg)^{2+t+\frac{1}{t}}$. This happens if and only if $b^{t+\frac{1}{t}} g^{2t} k_0^{\frac{1}{t}} < g^{2\zeta} \cdot (cg)^2$. On taking log base g on both sides, we get $\left(t + \frac{1}{t}\right)\varepsilon_b + 2t + \frac{\gamma}{t} < 2(\varepsilon_c + 1) + 2\zeta$, where $\gamma = \log_g k_0$. This can be rewritten as $t(\varepsilon_b + 2) + \frac{1}{t}(\varepsilon_b + \gamma) < 2(\varepsilon_c + 1) + 2\zeta$. Let $t = \sqrt{\frac{\varepsilon_b + \gamma}{\varepsilon_b + 2}}$. Then, $t \leq 1$. Observation 2.17 implies that for this t, the the L.H.S of the above inequality is minimized at the value $2\sqrt{(\varepsilon_b + 2)(\varepsilon_b + \gamma)}$. Thus, we get $\sqrt{(\varepsilon_b + 2)(\varepsilon_b + \gamma)} < \varepsilon_c + 1 + \zeta$ which is our assumption. Now, Corollary 2.15, Fact 2.2 imply that we can completely factorize N in $\mathrm{poly}(\log N, g^\zeta)$ time. □

Observation 4.3 implies $\gamma \leq 1 + \log_g 3$, where $\log_g 3 \ll 1$. On plugging this in Lemma 4.7, we get the following. Intuitively, we can "neglect" the term $\log_g 3$. We show a formal treatment.

Corollary 4.8. *Suppose $\sqrt{(\varepsilon_b + 1)(\varepsilon_b + 2)} < (\varepsilon_c + 1)$. Then, there exists an algorithm that takes input N, oracle access to Φ and outputs all the factors of N in time* $\mathrm{poly}(\log N)$.

Proof. We consider two cases. Suppose $\gamma \leq 1$. Then Lemma 4.7 implies the result ($\zeta = 0$). Now suppose $\gamma > 1$. However, by Observation 4.3 that $\gamma \leq 1 + \log_g 3$. Let us define $f(x) := \sqrt{(x+\gamma)(x+2)}$ and let $\tau > 0$. By the Mean value theorem, there exists $x - \tau \leq \xi \leq x$ such that $f(x) = f(x-\tau) + \tau \cdot f'(\xi)$. That is:

$$\sqrt{(x+\gamma)(x+2)} = \sqrt{(x+\gamma-\tau)(x+2-\tau)} + \tau \cdot \frac{2\xi + \gamma + 2}{2\sqrt{(\xi+\gamma)(\xi+2)}}.$$

Now, as $1 < \gamma \leq 1 + \log_g 3 \ll 2$, the function $\frac{2\xi+\gamma+2}{2\sqrt{(\xi+\gamma)(\xi+2)}}$ is a continuous function on the interval $[0, \infty) \times [1, 2]$ with a finite limit of 1 when $x \to \infty$. Therefore, it has an absolute upper bound S whenever $\gamma \in [1, 2]$ and $x \geq 0$. Set $\tau = \log_g 3$. We obtain $\sqrt{(\varepsilon_b + \gamma)(\varepsilon_b + 2)} \leq \sqrt{(\varepsilon_b + 1)(\varepsilon_b + 2)} + \log_g 3 \cdot S \leq \varepsilon_c + 1 + \log_g 3 \cdot S$. Observe that for $\zeta = \log_g 3 \cdot S$ we have that $g^\zeta = 3^S$ which is an absolute constant. Therefore, the result follows from Lemma 4.7. This completes the proof. □

Proposition 4.6 implies $k_0 = \alpha \cdot \frac{g^2}{b^2}$, where α is a constant. Thus, $\gamma \leq 2 - 2\varepsilon_b + \log_g \alpha$. Similarly, by substituting $\gamma = 2 - 2\varepsilon_b$ in Lemma 4.7 (and "neglecting" $\log_g \alpha \ll 1$) we get the following.

Corollary 4.9. *Suppose $\sqrt{4 - \varepsilon_b^2} < (\varepsilon_c + 1)$. Then, with oracle access to Φ, we can compute all the factors of N in time* $\mathrm{poly}(\log N)$.

Lemma 4.10. *Suppose $\varepsilon_b + \varepsilon_c < 1$. Then, with oracle access to Φ, we can completely factorize N in time* $\mathrm{poly}(\log N)$.

Proof. Let $M := \phi(N)$. Then, $M = bcg^3$. Let $D := g+1$. Then, $D \mid N$ and $(D-1)^3 \mid M$. As noted in the beginning of this section, $M \leq N$. Then, on substituting $\theta = 1$ and $\ell = 3$ in Lemma 2.16, we get that if $N^{\frac{1}{4}} < D - 1$ then we can factorize N in $\mathrm{poly}(\log N)$ time. Note that $N^{\frac{1}{4}} < g$ holds if and only if $(bcg^3) < g^4$. This happens if and only if $\varepsilon_b + \varepsilon_c < 1$, which is true by assumption. Now, by Lemma 2.16 and Fact 2.2, we can efficiently factorize N. □

The following lemma holds for a general square-free natural number with three factors. This improves Theorem 2 of [MRS23]. We give a comparison of this result with [MRS23] in Section 1.4.

Lemma 4.11. *Let $N \in \mathbb{N}$ be an arbitrary number such that $N = p_1 p_2 p_3$, where $p_3 < p_2 < p_1$ are primes. Let $\alpha_i := \log_N p_i$ for every $i \in \{1, 2, 3\}$. Suppose $\sqrt{(1-\alpha_2)(1-\alpha_1-\alpha_2)} < \alpha_2$. Then, given oracle access to Φ, we can completely factor N in* $\mathrm{poly}(\log N)$ *time.*

Proof. Let x, y be variables. Let $f := \phi(N)x + (x-1)xy - N(x-1)$. We first argue that f is irreducible over $\mathbb{Z}$. Suppose not. Then, $f = g \cdot h$, where $g, h \in \mathbb{Z}[x, y]$ are non-constant polynomials. As f is linear in y, we may assume without loss of generality that y only appears in g, i.e., $g = py + q$, where $p, q \in \mathbb{Z}[x]$ and $h \in \mathbb{Z}[x]$. As $f = gh$, we immediately get that $ph = x(x-1)$. If $h = x$ then f can not have a constant term. Thus, $h = x - 1$. Then, we get $q(x-1) = x(\phi(N) - N) + N$, which

implies that $q = -N$ and $q = \phi(N) - N$. But, this can not happen as $\phi(N) \neq 0$. Thus, we get a contradiction and hence f is irreducible.

Let $x_0 := p_3$ and $y_0 := p_2 + p_1 - 1$. Then, it is not difficult to verify that $f(x_0, y_0) = 0$. Let $X = p_3$ and $Y = 2p_1$. Then, as $p_2 < p_1$, $x_0 \leq X$ and $y_0 \leq Y$. Let $d = \deg_x(f)$ and $e = \deg_y(f)$. Then, $d = 2$ and $e = 1$. Let W be the parameter mentioned in Lemma 2.13. Then, it is not difficult to verify that $W = \Theta(p_3 N)$. We are given $\sqrt{(1-\alpha_2)(1-\alpha_1-\alpha_2)} < \alpha_2$. We will prove that there exists a $t > 0$ such that $X^{2+\frac{t}{2}} Y^{1+\frac{1}{t}} < 2^{1+\frac{1}{t}} \cdot u \cdot W$, where u is a constant. On substituting the values of X, Y, W and $N = p_1 p_2 p_3$, this happens if $(p_3)^{\frac{t}{2}} \cdot (p_1 p_3)^{\frac{1}{2t}} < p_2$. Taking log base N of the above inequality, we get $\left(\frac{\alpha_3}{2}\right) t + \left(\frac{\alpha_1+\alpha_3}{2}\right) \cdot \frac{1}{t} < \alpha_2$. Now, it follows from Observation 2.17 that when $t = \sqrt{\frac{\alpha_1+\alpha_3}{\alpha_3}}$, the minimum value of the L.H.S of the above inequality is $\sqrt{\alpha_3(\alpha_1+\alpha_3)}$. Thus, the above inequality holds if $\sqrt{\alpha_3(\alpha_1+\alpha_3)} < \alpha_2$. As $N = p_1 p_2 p_3$, clearly, $\alpha_1 + \alpha_2 + \alpha_3 = 1$. Using this in the previous inequality, we get $\sqrt{(1-\alpha_2-\alpha_1)(1-\alpha_2)} < \alpha_2$, which is true as per the assumption. Now, it follows from Lemma 2.14 that we can find a non-trivial factor of N in time $\text{poly}(\log N, 2^{1+\frac{1}{2t}})$ for $t = \sqrt{\frac{\alpha_1+\alpha_3}{\alpha_3}}$. Thus, $\frac{1}{2t} = \frac{1}{2} \cdot \sqrt{\frac{\alpha_3}{\alpha_3+\alpha_1}} \leq \frac{1}{2}$ as $0 < \alpha_1$. Hence, we get hold of a non-trivial factor of N in time $\text{poly}(\log N)$. Now, Fact 2.2 implies that we can completely factor N in $\text{poly}(\log N)$ time. □

4.2 Proof of Theorem 1

Let $N = pqr$ be a C_3-number, where $p < q < r$ are primes. Let $g = \gcd(p-1, q-1, r-1)$ and $a, b, c \in \mathbb{N}$ be such that $p = ag + 1, q = bg + 1$, and $r = cg + 1$. Recall that we are assuming $g = \omega(\text{poly}(\log N))$. We first write an equivalent version of Theorem 1 by replacing α_1 and α_2 with $\varepsilon_c := \log_g c$ and $\varepsilon_b := \log_g b$ respectively, where $\alpha_1 = \log_N r$ and $\alpha_2 = \log_N q$. A reason behind this switching is that it is more convenient to work with log base g than working with log base N. We stated Theorem 1 using log base N so that we can compare our results with [MRS23], where the 'size' bounds were stated with respect to log base N. Consider the following lemma, which helps us switch Theorem 1 to the log base g set-up.

Lemma 4.12. *Let $N = pqr$ be a simple C_3-number and $\varepsilon_b, \varepsilon_c$ be the quantities defined above. Then, N satisfies a bound given in Theorem 1 if and only if it satisfies a bound in the following list.*

1. $\sqrt{(\varepsilon_b + 2)(\varepsilon_b + 1)} < \varepsilon_c + 1$.
2. $\sqrt{4 - \varepsilon_b^2} < \varepsilon_c + 1$.
3. $\varepsilon_b + \varepsilon_c < 1$.
4. $\sqrt{\varepsilon_c + 2} < \varepsilon_b + 1$.

Proof. Recall $\alpha_1 = \log_N r$ and $\alpha_2 = \log_N q$. Then, note that $\alpha_1 = \frac{\log_g(cg+1)}{\log_g N}$ and $\alpha_2 = \frac{\log_g(bg+1)}{\log_g N}$. Notice that there exist $cg \leq \xi \leq cg + 1$ such that $\log_g(cg+1) = \log_g(cg) + \frac{1}{\xi \ln b}$. Using techniques similar to Lemma 4.7 and Corollary 4.8, we can "neglect" low-order terms and hence use the approximations $\log_g(cg+1) \approx \log_g cg$ and $\log_g(bg+1) \approx \log_g bg$. Thus, we get $\log_g cg + 1 = \varepsilon_c + 1$ and $\log_g bg + 1 = \varepsilon_b + 1$. Similarly, as $N = \Theta(bcg^3)$, $\log_g N = \varepsilon_b + \varepsilon_c + 3$. Then, the above equation can be written as $\alpha_1 = \frac{\varepsilon_c+1}{\varepsilon_b+\varepsilon_c+3}$ and $\alpha_2 = \frac{\varepsilon_b+1}{\varepsilon_b+\varepsilon_c+3}$. Now, using these equations, it is easy to show that the bounds given in Points 1, 2, 3 and 4 in Theorem 1 hold if and only if the bounds given in Points 1, 2, 3 and 4 of this lemma hold. □

The proof of this lemma and Lemma 4.11 imply the following.

Corollary 4.13. *Let $N = (g+1)(bg+1)(cg+1)$ be a simple C_3-number. Let $\varepsilon_b := \log_g b$ and $\varepsilon_c := \log_g c$. Suppose $\sqrt{\varepsilon_c + 2} < \varepsilon_b + 1$. Then, there exists an algorithm that takes input N, oracle access to Φ and outputs all the factors of N in time* $\text{poly}(\log N)$.

Lemma 4.12 implies the following reformulation of Theorem 1.

Theorem 4.14. (Restatement of Theorem 1) *Let $N = pqr$ be a simple C_3-number, where $p < q < r$ are primes. Let $g := \gcd(p-1, q-1, r-1)$. Then, there exist $b, c \in \mathbb{N}$ such that $p = g+1, q = bg+1$, and $r = cg+1$. Let $\varepsilon_b := \log_g b$ and $\varepsilon_c := \log_g c$. Suppose N satisfies one of bounds given in Point 1 to Point 4 of Lemma 4.12. Then, given N and oracle access to Φ, we can factor N in* $\text{poly}(\log N)$ *time.*

Proof. The desired factoring is done by Algorithm 1.

Algorithm 1 Factoring simple C_3-numbers using Φ

Input: An integer N and oracle access to Φ.
Output: The complete factorization of N if it is a simple C_3-number and satisfies a bound given in Theorem 4.14.

1: For $D \in \{2, \ldots, \log N\}$, check if $D \mid N$ and go to 4.
2: Let $M_1 := \gcd(N-1, \phi(N))$ and $M_2 := \phi(N)$. Let x, y be variables, $f_1 := yN - yx - M_1 x$, $f_2 := y(N-x)^3 - M_2 x^3$, $f_3 := M_2 x + (x-1)xy - N(x-1)$.
3: Find all *bounded* roots $x_{i,0}, y_{i,0}$ of f_i for every $i \in \{1, 2, 3\}$ using Lemma 2.14. If some $x_{i,0}$ is a non-trivial factor of N, $D := x_{i,0}$ and go to 4.
4: Run the algorithm given in Fact 2.2 on $\frac{N}{D}$. If it returns all factors of $\frac{N}{D}$, output these factors along with D.

It is easy to note that the running time of Algorithm 1 is $\text{poly}(\log N)$. Now, we argue the correctness of the algorithm. We assume that N is a simple C_3-number. Suppose either $\sqrt{(\varepsilon_b + 2)(\varepsilon_b + 1)} < \varepsilon_c + 1$ or $\sqrt{4 - \varepsilon_b^2} < \varepsilon_c + 1$ is true. Then, we know from Corollaries 4.8 and 4.9 that the algorithm works correctly. Now, suppose $\varepsilon_b + \varepsilon_c < 1$ holds. Then, Lemma 4.10 implies that the algorithm works correctly. At last, suppose $\sqrt{\varepsilon_c + 2} < \varepsilon_b + 1$ is true. Then, Corollary 4.13 implies the algorithm returns the complete factorization of N. As N is Carmichael, it follows from Definition 2.1 that if we compute a non-trivial factor D of N then $\frac{N}{D}$ is square-free. Hence, Step 4, which uses Fact 2.2 always works correctly. □

5 Open questions

1. **Deterministic factoring algorithm for C_3-numbers given access to either Φ or Λ.** We show in Theorem 1.1 that we can factor simple C_3-numbers efficiently, given oracle access to Λ. Yet, when instead given oracle access to Φ, our algorithm becomes subject to 'size' bounds. Can we get a deterministic polynomial-time algorithm for factoring C_3-numbers, given oracle access to either Λ or Φ, subject to *no bounds*? Can we extend the result to other "natural" families of integers?

2. **Deterministic polynomial-time reduction between the computation of $\phi(\cdot)$ and $\lambda(\cdot)$.** As noted in Section 1, integer factoring, computation of Euler's Totient function, and computing the Carmichael's Lambda functions are Turing-reducible to each other in randomized polynomial time. Is the same result holds true in the deterministic setting?

3. **Relation between $\phi(N)$ and $\lambda(N)$.** We know from Fact 2.5 that if N is a product of two distinct primes then $\phi(N) = \lambda(N) \cdot \gcd(N-1, \lambda(N))$. Thus, in this case, we can compute $\phi(N)$ using $\lambda(N)$ in deterministic polynomial time. If we could get such a neat relationship between $\phi(N)$ and some efficiently-computable function of $\lambda(N)$ when N is a product of at least three distinct primes then Theorem 1.1 will imply a deterministic polynomial-time algorithm to factor all simple C_3-numbers with oracle access to Φ.

6 Acknowledgment

Part of this project by N. Gupta was completed while he was a postdoctoral researcher at Boston College. He would like to sincerely thank Boston College for the financial and institutional support provided during this time. The authors would also like to thank the reviewers for their valuable comments.

References

[AB03] M. Agrawal and S. Biswas. Primality and identity testing via chinese remaindering. *JACM*, 50(4):429–443, 2003.

[AGP94] W. R. Alford, A. Granville, and C. Pomerance. There are infinitely many carmichael numbers. *Annals of Mathematics*, 139(3):703–722, 1994.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. Primes is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[ASS16] M. Agrawal, N. Saxena, and S. S. Srivastava. Integer factoring using small algebraic dependencies. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016*, volume 58 of *LIPIcs*, pages 6:1–6:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[BBC+88] P. Beauchemin, G. Brassard, C. Crépeau C., Goutier, and C. Pomerance. The generation of random numbers that are probably prime. *J. Cryptology*, 1:53–64, 01 1988.

[BDHG99] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $N = p^r q$ for large r. In *Advances in Cryptology — CRYPTO' 99*, pages 326–337. Springer Berlin Heidelberg, 1999.

[BGS07] A. Bostan, P. Gaudry, and É. Schost. Linear recurrences with polynomial coefficients and application to integer factorization and cartier–manin operator. *SIAM Journal on Computing*, 36(6):1777–1806, 2007.

[BLP93] J. P. Buhler, H. W. Lenstra, and C. Pomerance. Factoring integers with the number field sieve. In *Lecture notes in Mathematics - The development of the number field sieve*, pages 50–94. Springer Berlin Heidelberg, 1993.

[BM05] J. Blömer and A. May. A tool kit for finding small roots of bivariate polynomials over the integers. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494, pages 251–267. Springer, 2005.

[BMS86] E. Bach, G. L. Miller, and J. Shallit. Sums of divisors, perfect numbers and factoring. *SIAM J. Comput.*, 15(4):1143–1154, 1986.

[BN97] R. Balasubramanian and S. V. Nagaraj. Density of carmichael numbers with three prime factors. *Mathematics of Computation*, 66:1705–1708, 1997.

[Car10] R.D. Carmichael. Note on a new number theory function. *Bulletin of the American Mathematical Society*, 16(5):232–238, 1910.

[CH14] E. Costa and D. Harvey. Faster deterministic integer factorization. *Math. Comput.*, 83:339–345, 2014.

[Che39] J. Chernick. On Fermat's simple theorem. *Bulletin of the American Mathematical Society*, 45(4):269 – 274, 1939.

[CKT13] J.S. Coron, A. Kirichenko, and M. Tibouchi. A note on the bivariate coppersmith theorem. *Journal of Cryptology*, 26:246–250, 04 2013.

[Cop96] D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *LNCS*, pages 178–189. Springer, 1996.

[CZ18] J.S. Coron and R. Zeitoun. Improved factorization of $N = p^r q^s$. In *Topics in Cryptology – CT-RSA 2018*, pages 65–79. Springer International Publishing, 2018.

[dFTH94] P. de Fermat, P. Tannery, and C. Henry. *Oeuvres de Fermat: Vol.: 2*. Gauthier-Villars, 1894.

[DV21] Y. Du and I. Volkovich. Approximating the number of prime factors given an oracle to euler's totient function. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 213 of *LIPIcs*, pages 17:1–17:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Erd56] P. Erdös. On pseudoprimes and carmichael numbers. *Publ. Math. Debrecen*, 4:201–206, 1956.

[Fil12] Y. Filmus. Factorization methods: Very quick overview. 2012.

[For25] https://blog.computationalcomplexity.org/2025/11/factoring-carmichael-numbers.html, 2025.

[Har21] D. Harvey. An exponent one-fifth algorithm for deterministic integer factorisation. *Mathmatics of Computation*, 90:2937–2950, 2021.

[Hit21] M. Hittmeir. A time-space tradeoff for lehman's deterministic integer factorization method. *Mathematics of Computation*, 90(330):1999–2010, 2021.

[HP20] M. Hittmeir and J. Pomykala. Deterministic integer factorization with oracles for euler's totient function. *Fundam. Inform.*, 172(1):39–51, 2020.

[IW97] R. Impagliazzo and A. Wigderson. P=BPP unless E has subexponential circuits: derandomizing the xor lemma. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 220–229, 1997.

[Jam11] G. J. O. Jameson. Carmichael numbers with three prime factors. 2011. https://www.maths.lancs.ac.uk/~jameson/3car.pdf.

[Kar17] S. Karnik. On the classification and algorithmic analysis of carmichael numbers, 2017. https://arxiv.org/pdf/1702.08066.pdf.

[KVZ20] J. Kim, I. Volkovich, and N. X. Zhang. The power of leibniz-like functions as oracles. In *The 15th International Computer Science Symposium in Russia, CSR*, volume 12159 of *Lecture Notes in Computer Science*, pages 263–275. Springer, 2020.

[Lad75] R. E. Ladner. The circuit value problem is log space complete for *P*. *SIGACT News*, 7(1):18–20, 1975.

[Lan88] S. Landau. Some remarks on computing the square parts of integers. *Inf. Comput.*, 78(3):246–253, 1988.

[Lar22] D. Larsen. Bertrand's Postulate for Carmichael Numbers. *International Mathematics Research Notices*, 2023(15):13072–13098, 07 2022.

[Leh74] R. S. Lehman. Factoring large integers. *Mathematics of Computation*, 28:637–646, 1974.

[Leh76] D. H. Lehmer. Strong carmichael numbers. *Journal of the Australian Mathematical Society*, 21(4):508–510, 1976.

[Len87] H. W. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics*, 126(3):649–673, 1987.

[Len00] A. K. Lenstra. Integer factoring. *Designs, Codes and Cryptography*, 19:101–128, 2000.

[Lon81] D. L. Long. Random equivalence of factorization and computation of orders. Technical Report 284, Princeton University, 1981.

[May09] A. May. Using lll-reduction for solving rsa and factorization problems. In *The LLL algorithm*, pages 315–348, 02 2009.

[MB75] M. A. Morrison and J. Brillhart. A method of factoring and the factorization of F_7. *Mathematics of Computation*, 29(129):183–205, 1975.

[Mil76] G. L. Miller. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci.*, 13(3):300–317, 1976.

[MRS23] F. Morain, G. Renault, and B. Smith. Deterministic factoring with oracles. *Applicable Algebra in Engineering, Communication, and Computation*, 34:663–690, 2023.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *J. Comput. Syst. Sci.*, 49(2):149–167, 1994.

[OU98] T. Okamoto and S. Uchiyama. A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology — EUROCRYPT'98*, pages 308–318, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[Pin98] Richard G. E. Pinch. The carmichael numbers up to 10^{16}, 1998.

[Pol74] J. M. Pollard. Theorems on factorization and primality testing. *Mathematical Proceedings of the Cambridge Philosophical Society*, 76(3):521–528, 1974.

[Pol75] J. M. Pollard. A monte carlo method for factorization. *BIT*, 15(3):331–334, sep 1975.

[Pom85] C. Pomerance. The quadratic sieve factoring algorithm. In *Advances in Cryptology*, pages 169–182. Springer Berlin Heidelberg, 1985.

[Pom92] C. Pomerance. Carmichael numbers. *An elaborate version of the Beeger lecture given on April 22, 1992, during the 28th Nederlands Mathematisch Congress in Delft*, 1992. https://math.dartmouth.edu/~carlp/carmsurvey.pdf.

[Pom98] C. Pomerance. A tale of two sieves. *Notices of the AMS*, 1998.

[Rab80] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of number theory*, 12(1):128–138, 1980.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978.

[RV23] A. Ravi and I. Volkovich. New characterization of the factor refinement algorithm with applications. In *Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation, ISSAC*, pages 490–497. ACM, 2023.

[Sam24] S. Wagstaff Jr. Samuel. Pseudoprimes and fermat numbers. *INTEGERS*, 24, 02 2024.

[Sha71] D. Shanks. Class number, a theory of factorization, and genera. volume 20, pages 415–440, 1971.

[Sho97] P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. on Computing*, 26(5):1484–1509, 1997.

[Slo] N.J.A. Sloane. The on-line encyclopedia of integer sequencing. https://oeis.org/A002997/b002997.txt.

[Sta12] https://crypto.stackexchange.com/questions/5279/carmichael-number-factoring, 2012.

[Str77] V. Strassen. Einige resultate über berechnungskomplexität. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 78:1–8, 1976/77.

[Sut07] A. Sutherland. *Order computations in generic groups.* PhD thesis, Massachusetts Institute of Technology, 2007.

[SW26] A. Shallue and J. Webster. Algorithms for carmichael numbers, 2026.

[Tak98] T. Takagi. Fast rsa-type cryptosystem modulo $p^k q$. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference*, volume 1462 of *LNCS*, pages 318–326. Springer, 1998.

[Wil82] H. C. Williams. A $p+1$ method of factoring. *Mathematics of Computation*, 39(159):225–234, 1982.

[Wol87] H. Woll. Reductions among number theoretic problems. *Inf. Comput.*, 72(3):167–179, 1987.

[WRI16] T. WRIGHT. Factors of carmichael numbers and a weak k-tuples conjecture. *Journal of the Australian Mathematical Society*, 100(3):421–429, 2016.

[Wri19] T. Wright. Factors of carmichael numbers and even weaker k-tuples conjecture. *Bulletin of the Australian Mathematical Society*, 99(3):376–384, 2019.

[Źr10] B. Źrałek. A deterministic version of pollard's $p-1$ algorithm. *Mathematics of Computation*, 79(269):513–513, January 2010.

A Missing Proofs from Section 2

A.0.1 Proof of Fact 2.2

As p, q are distinct prime numbers, it is well-known that $\phi(pq) = \phi(p)\phi(q)$. Then,

$$\phi(N) = \phi(p)\phi(q) = (p-1)(q-1).$$

Then, $N - \phi(N) + 1 = p + q$. As we have oracle access to Φ, we compute $d := p + q$. Treat p and q as formal variables. Then, $q = d - p$. As $N = pq$, we get

$$p^2 - dp + N = 0.$$

We know that $\frac{d+\sqrt{d^2-4N}}{2}$ and $\frac{d-\sqrt{d^2-4N}}{2}$ are the two roots of the above equation. Using the values of d and N, we compute these roots and these are the prime factors p and q of N. As we can multiply two $\log N$ bit numbers and compute square roots of a perfect square in time $\text{poly}(\log N)$, clearly we compute p and q in $\text{poly}(\log N)$. □

A.1 Proof of Fact 2.9

We prove the three properties one by one. First, assume that n is even. As n is Carmichael, it is composite and it follows from Definition 2.7 that n is square-free. Then, there exists an odd prime number p such that $p \mid n$. Then, by Definition 2.7, $p - 1 \mid n - 1$. But, this can not happen as $p - 1$ is even and $n - 1$ is odd. Hence, we get a contradiction and n is odd.

Now suppose that $n = pq$, where p and q are prime numbers and $p < q$. Note that $pq - 1 = (p-1)(q-1) + (p-1) + (q-1)$. We know from Definition 2.7 that $p-1 \mid pq-1$ and $q-1 \mid pq-1$, which implies $p-1 \mid q-1$ and $q-1 \mid p-1$. As $p < q$, we get $p-1 < q-1$. As $q-1 \mid p-1$, we get that $p-1 = 0$, which implies $p = 1$. Thus, p is not prime, which is a contradiction. Hence, n has at least three prime factors.

Let p be a prime factor of n. As n is Carmichael, we know from Definition 2.7 that $p-1 \mid n-1$, which in turn implies that

$$p-1 \;\Big|\; (n-1) - (p-1) = n - p = p \cdot \left(\frac{n}{p} - 1\right)$$

Note that $\frac{n}{p} - 1$ is an integer number. Now, since $\gcd(p-1, p) = 1$ we obtain that $p-1 \mid \frac{n}{p} - 1$ and hence $p - 1 \leq \frac{n}{p} - 1$. This in turn implies that $p^2 \leq n$. Since n is square-free we cannot have equality. Therefore, $p^2 < n$ and hence $p < \sqrt{n}$. □

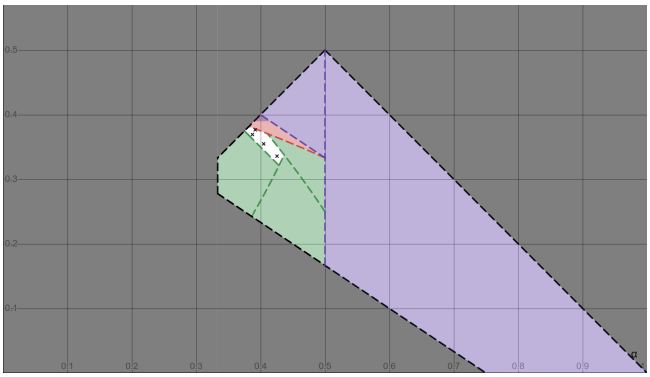
A.2 Proof of Observation 2.17

Proof. We know that if C is such that $\left(\frac{df}{dt}\right)|_{t=C} = 0$ then, $f|_{t=C}$ is the minimum value of f for any substitution of t. As $\frac{df}{dt} = A - \frac{B}{t^2}$, we get $C = \sqrt{\frac{B}{A}}$. This implies that $f|_{t=C} = 2\sqrt{AB}$ is the minimum value of f for any substitution of t. □

B Experimental results

In this section we provide some experimental results. For the experimental data, we applied the results on all the simple C_3-numbers up to 1.7×10^{13} taken from [Slo]. There are 787 such numbers. Carmichael numbers upto 10^{16} have also been studied in [Pin98].

First, we observe that the results of [MRS23] cover only 14 simple C_3-numbers in the range, which are less than 2%. Our result in Theorem 1 covers **all** except four simple C_3-numbers in the same range, which is more than 99%. The areas colored with red and green correspond to the simple C_3-numbers that satisfy 'size' bounds mentioned in Theorem 1. The dots inside the white region represent the four 'missed' numbers.



Now, we give some numerical results which show that these numbers are depicted by various size bounds in Theorem 1. For the following examples, let $\alpha_1 := \log_N r$ and $\alpha_2 := \log_N q$.

Example B.1. *Let* $N = 1635299688961$. *Then,* $p = 823, q = 4111$ *and* $r = 483337$. *In this case,* N *is a simple* C_3*-number,* $\alpha_1 = 0.4654034252$ *and* $\alpha_2 = 0.2958954166$. *It turns out that* $\sqrt{\alpha_2(1-\alpha_1)} = 0.3977243722 < \alpha_1$. *Clearly, it satisfies the bound given in Point 1 of Theorem 2. Thus, given oracle access to* Φ*, we can compute* p, q, r *in* $\mathrm{poly}(\log N)$ *time using Algorithm 1 along with Fact 2.2.*

Example B.2. *Let* $N = 1337024507209$. *Then,* $p = 313, q = 9049$ *and* $r = 472057$. *In this case,* N *is a simple* C_3*-number,* $\alpha_1 = 0.4679143353$ *and* $\alpha_2 = 0.3262869136$. *Then, we get that* $\sqrt{(1-\alpha_1)(3-3\alpha_1-4\alpha_2)} = 0.3935671563 < \alpha_1$. *Clearly, it satisfies the bound given in Point 2 of Theorem 2. Thus, given oracle access to* Φ*, we can compute* p, q, r *in* $\mathrm{poly}(\log N)$ *time using Algorithm 1 along with Fact 2.2.*

Example B.3. *Let* $N = 209850699601$. *Then,* $p = 1913, q = 5737$, *and* $r = 19121$. *In this case,* N *is a simple* C_3*-number,* $\alpha_1 = 0.37816149767438$ *and* $\alpha_2 = 0.33198327048916$. *Thus,* $\alpha_1 + \alpha_2 = 0.7101447682$ *and* $\alpha_1 + \alpha_2 < \frac{3}{4}$ *holds in this case. Clearly, it satisfies the bound given in Point 3 of Theorem 2. Thus, given oracle access to* Φ*, we can compute* p, q, r *in* $\mathrm{poly}(\log N)$ *time using Algorithm 1 along with Fact 2.2.*

Example B.4. *Let* $N = 1452393993961$. *Then,* $p = 311, q = 28211$ *and* $r = 165541$. *In this case,* N *is a simple* C_3*-number,* $\alpha_1 = 0.429112756$ *and* $\alpha_2 = 0.3659256356$. *Then, we get that* $\sqrt{(1-\alpha_2)(1-\alpha_1-\alpha-2)} = 0.3605009037 < \alpha_2$. *This implies that the bound given in Point 4 of Theorem 2 is satisfied. Thus, given oracle access to* Φ*, we can compute* p, q, r *in* $\mathrm{poly}(\log N)$ *time using Algorithm 1 along with Fact 2.2.*

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il