# Deterministic factoring with oracles

François Morain, Guénaël Renault, Benjamin Smith

# DETERMINISTIC FACTORING WITH ORACLES

FRANÇOIS MORAIN, GUÉNAËL RENAULT, AND BENJAMIN SMITH

Abstract. Can we factor an integer $N$ unconditionally, in deterministic polynomial time, given the value of its Euler totient $\varphi(N)$? We show that this can be done under certain size conditions on the prime factors of $N$. The key technique is lattice basis reduction using the LLL algorithm. Among our results, we show that if $N$ has a prime factor $p > \sqrt{N}$, then we can recover $p$ in deterministic polynomial time given $\varphi(N)$. We also shed some light on the analogous factorization problems given oracles for the sum-of-divisors function, Carmichael's function, and the order oracle that is used in Shor's quantum factoring algorithm.

## 1. Introduction

The *fundamental theorem of arithmetic* states that every positive integer $N$ can be written in a unique way, up to permutation of the factors, as

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

where the $p_i$ are distinct primes, and each $e_i > 0$. Making this theorem explicit by computing the prime factorization of $N$—that is, computing the $p_i$ and $e_i$—is a fundamental problem in algorithmic number theory. This article is concerned with *deterministic* factorization algorithms.

Some numbers are easy to factor deterministically. If $N$ is prime, then Miller [41] proved that $N$ can be proven prime in deterministic polynomial time assuming the Generalized Riemann Hypothesis (see also [35]). The same result was proven unconditionally in [2]. In practice, small numbers can be proven prime using a combination of pseudoprimality tests. For large numbers, several faster (though heuristic) methods exist: see [20] for details. Prime powers can be detected in quasi-linear time [6].

But when $N$ has more than one prime factor, hard work is generally required. In the quantum world, we can apply Shor's algorithm [51]. In the classical world, the fastest algorithms are non-deterministic: depending on the size of $N$, one may use Lenstra's ECM or the Number Field Sieve (NFS), the best general-purpose factoring algorithm, which runs in heuristic time $\exp((\sqrt[3]{64/9} + o(1))(\log N)^{1/3}(\log\log N)^{2/3})$ [20]. This complexity explains the success of the RSA cryptosystem, which is based on the supposed difficulty of factoring numbers with only two prime factors.

*Date*: August 12, 2021.

Deterministic unconditional factoring methods are rare; all such methods known have exponential running time for general $N$. The first such method was due to Fermat, followed by Lehman [33]; Pollard's approach [44] has been built on by recent methods including Bostan–Gaudry–Schost [9], Costa–Harvey [19], and Hittmeir [26], all in time $\tilde{O}(N^{1/4})$. More recently, this complexity has been improved to $\tilde{O}(N^{2/9})$ by Hittmeir [27], and to $\tilde{O}(N^{1/5})$ by Harvey [24] (see also [25] for a later speedup). Better results exist for numbers known to have special forms: for example, [8] describes a method to factor $N = p_1^r p_2$ that runs in polynomial time when $p_1$ and $p_2$ are of roughly the same size and $r$ is in $\Omega(\log p_1)$. This was extended in [18] to numbers $N = p_1^r p_2^s$ with $r$ and/or $s$ in $\Omega((\log p_1)^3)$.

The use of *oracles* allows us to abstract and encapsulate the availability of extra information about the number $N$. It is thus a traditional way of trying to understand the difficulty of factoring. In this work, we consider factoring algorithms with access to one of the following oracles (defined formally in §2.1):

- $\Phi$: on input $N$ returns $\varphi(N)$, the value of the Euler totient function;
- $\Lambda$: on input $N$ returns $\lambda(N)$, the value of the Carmichael lambda function;
- $\mathcal{O}$: on input $N$ and $a$ with $\gcd(a, N) = 1$, returns the order of $a$ modulo $N$;
- $\Sigma$: on input $N$ returns $\sigma(N)$, the sum of all positive divisors of $N$.

We study the conditions under which these oracles can be used to factor $N$ deterministically, unconditionally, and in a time complexity better than exponential, in the spirit of [1, Rem23$_{86}$].

The story of factoring with oracles began with Miller [41], who proved the equivalence of $\Phi$ and factoring under ERH. Long [38] proved that factoring is randomly polynomially equivalent to computing orders. Woll [54] explored relationships between number-theoretic problems including factorization and the $\Phi$ and $\mathcal{O}$ oracles. Źrałek [55] has shown that almost all integers $N$ can be factored in deterministic polynomial time given $\varphi(N)$; also, iterated calls to $\Phi$ allow deterministic factoring in subexponential time, after using Landau's algorithm to reduce to the squarefree case (see §5.1). This work was subsequently extended in [56] (using methods tangential to ours).

In a different direction, Bach, Miller, and Shallit [4] showed that $\Sigma$ allows efficient randomized factoring (see §2.3). Chow [10] has studied factoring with an oracle of a completely different nature, using coefficients of modular forms; this turns out to be very powerful, since it solves the integer factorization problem.

There is also an important practical motivation for oracles in factoring. In the context of RSA moduli $N = p_1 p_2$, the problem of factoring given additional information on $p_1$ and $p_2$ has been studied since 1985. For example, Rivest and Shamir showed in [47] that if $N$ has bitlength $n$ and the factors $p_1$ and $p_2$ are balanced (with bitlengths close to $\frac{n}{2}$), then $N$ can be factored in polynomial time if we have access to an oracle returning the $\frac{n}{3}$ most significant bits of $p_1$. Beyond their theoretical interest, these algorithms are motivated by cryptographic hardware attacks: the oracle is an abstraction representing side-channel analysis revealing some of the bits of the secret factors. In 1996, Coppersmith improved Rivest and Shamir's results by applying lattice-based methods to the problem of finding small integer roots of bivariate integer polynomials (what is now called *Coppersmith's method* [11]). For instance, knowing the $n/4$ most (or least) significant bits of $p$ is enough to factor $N$ in polynomial time. In the same cryptographic context, the Coppersmith approach was used to prove that given a pair of RSA exponents $(e, d)$ (with $d \equiv 1/e \bmod \varphi(N)$), one can recover the two prime factors of $N$ in deterministic polynomial time [15].

In this article we combine these approaches, applying lattice-based techniques to factoring with number-theoretic oracles. Our results rely on diophantine geometry, using classical continued fractions and the LLL algorithm in a manner inspired by the cryptographic work mentioned above. Our results include the following:

**Theorem 1.1.** *Assume $N$ is squarefree and has at least three prime factors, of which the largest $p$ satisfies $p > \sqrt{N}$. Then we can recover $p$ in deterministic polynomial time in* $\log(N)$ *given one of $\varphi(N)$, $\lambda(N)$, or $\sigma(N)$.*

*Proof.* See Theorem 5.4. □

**Theorem 1.2.** *Assume $N$ is squarefree and has exactly three prime factors $p_1 > p_2 > p_3$. Put $\alpha_i = \log p_i / \log N$. Then we can compute a nontrivial factor of $N$ in deterministic polynomial time in* $\log(N)$ *given $\varphi(N)$ or $\sigma(N)$ if* at least one *of the following conditions hold:*

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

*Proof.* Follows from Theorems 5.5, 5.6, 5.9, and 5.4. □

We define the oracles, and recall some associated number-theoretic results, in §2, before stating the relevant results of Coppersmith and Howgrave-Graham in §3. Our core results in §4 solve (generalizations of) the following problem: given $N$ and $M$ such that there exists a (large enough) prime $p$ with $p \mid N$ and $p \pm 1 \mid M$, recover $p$ in deterministic polynomial time. We apply these algorithms to factoring with $\Phi$, $\Lambda$, and $\Sigma$ in §5, and with $\mathcal{O}$ and other oracles in §6.

## 2. Number-theoretic oracles

As above, suppose $N = \prod_{i=1}^{k} p_i^{e_i}$, where the $p_i$ are distinct primes and $e_i > 0$. Let $\omega(N)$ denote the number of prime divisors of $N$ (so $\omega(N) = k$ above). Recall that $\omega(N)$ is trivially bounded above by $(\log N)/(\log 2)$, and is of order $\log \log N$ on average.

### 2.1. **The oracles.**

**Definition 2.1** (The $\Phi$ oracle)**.** Given $N$ as above, the oracle $\Phi$ returns the value of the Euler totient function

$$\varphi(N) = \prod_{i=1}^{\omega(N)} p_i^{e_i-1}(p_i-1)\,,$$

which counts the number of integers in $\{1,\ldots,N-1\}$ that are prime to $N$; that is, $\varphi(N)$ is the cardinality of the multiplicative group $(\mathbb{Z}/N\mathbb{Z})^\times$.

**Definition 2.2** (The $\Lambda$ oracle)**.** Given $N$ as above, the oracle $\Lambda$ returns the value of Carmichael's $\lambda$ function

$$\lambda(N) = \operatorname{lcm}_{i=1}^{\omega(N)} \lambda(p_i^{e_i}) \quad \text{where} \quad \lambda(p_i^{e_i}) = \begin{cases} 1 & \text{if } p_i = 2 \text{ and } e_i = 1, \\ 2 & \text{if } p_i = 2 \text{ and } e_i = 2, \\ \varphi(2^{e_i})/2 & \text{if } p_i = 2 \text{ and } e_i > 2. \\ \varphi(p_i^{e_i}) & \text{if } p_i > 2\,. \end{cases}$$

This is the exponent of $(\mathbb{Z}/N\mathbb{Z})^\times$: that is, the maximal multiplicative order of an element modulo $N$.

**Definition 2.3** (The $\mathcal{O}$ oracle)**.** Given $N$ as above and $a$ with $\gcd(N,a) = 1$, the oracle $\mathcal{O}$ returns the order

$$\operatorname{ord}_N(a) := \min\{r : r \in \mathbb{Z}_{>0} \mid a^r \equiv 1 \pmod{N}\}\,.$$

Shor's quantum factorization algorithm uses the Quantum Fourier Transform to construct a quantum polynomial-time order-finding algorithm, which yields an efficient factorization algorithm after some classical post-processing (similar to the process in §2.3 below). This order-finding algorithm is not a true realization of $\mathcal{O}$, since it is only guaranteed to return a *divisor* of $\operatorname{ord}_N(a)$, but for most inputs it returns the true order with very high probability. Factoring with $\mathcal{O}$ therefore gives us valuable intuition into Shor-style quantum factoring algorithms.

**Definition 2.4** (The $\Sigma$ oracle)**.** Given $N$ as above, the oracle $\Sigma$ returns the sum of the divisors of $N$: that is,

$$\sigma(N) := \sum_{d \mid N} d = \prod_{i=1}^{\omega(N)} \frac{p_i^{e_i+1}-1}{p_i-1}\,.$$

### 2.2. **Relationships between $\Phi$, $\Lambda$, and $\mathcal{O}$.**

Lagrange's theorem tells us that the order of an element divides the order, and indeed the exponent, of the group. Applying this to $(\mathbb{Z}/N\mathbb{Z})^\times$ gives

$$\operatorname{ord}_N(a) \mid \lambda(N) \quad \text{and} \quad \lambda(N) \mid \varphi(N)$$

for all $N$ and all $a$ prime to $N$.

While the $\varphi$ and $\lambda$ functions may seem very close, it is easy to see that $\varphi(N)/\lambda(N)$ can be made quite large. For example, if $N = p_1p_2$ where $p_1 - 1 = 2(p_2 - 1)$, then $\varphi(N)/\lambda(N) = p_2 - 1 = \Omega(\sqrt{N})$.

Recall that if $p$ is a prime, then the valuation $\nu_p(x)$ of an integer $x$ at $p$ is the maximal $e$ such that $p^e \mid x$. If $N$ is odd, then $\nu_2(\varphi(N)) = \sum_{i=1}^{\omega(N)} \nu_2(p_i - 1) \geq \omega(N)$ is an easy upper bound for $\omega(N)$, which may be useful when we have access to $\Phi$ (though this bound is generally far from tight). In contrast, $\nu_2(\lambda(N)) = \max_{i=1}^{\omega(N)} \nu_2(p_i - 1)$ gives us no information about $\omega(N)$ on its own—and so neither does $\nu_2(\mathrm{ord}_N(a))$ for any $a$.

### 2.3. **Randomized and conditional algorithms.**

All of these oracles give efficient *randomized* factoring algorithms (see [4]). When $N$ is composite, $\varphi(N)$ and $\lambda(N)$ are even, which enables us to find some $c \neq \pm 1$ in $\mathbb{Z}/N\mathbb{Z}$ such that $c^2 \equiv 1 \pmod{N}$, and then $\gcd(c - 1, N)$ is a nontrivial factor of $N$. See Appendix A for the corresponding algorithms. For $\Sigma$, we refer to [4] again.

Folklore tells us that there is a randomized polynomial-time reduction between computing square roots modulo $N$ and factoring $N$. Rabin gives a precise analysis when $N$ is a product of two primes in [45, Theorem 1]. To render this approach deterministic (as in [41]) one needs a bound on non-quadratic residues, but this bound is currently only known to hold under ERH.

## 3. Lattices, Coppersmith's method, and approximate common divisors

In this section we recall some essential results on our two basic tools: Coppersmith's method for finding small roots of polynomials, and Howgrave-Graham's approximate common divisors. We also introduce some elementary subroutines that we will use to improve the quality of our factorizations.

The Lenstra–Lenstra–Lovász lattice basis reduction algorithm (LLL) [34] is at the heart of both Coppersmith's and Howgrave-Graham's methods. Recall that if $L$ is a lattice of dimension $n$ in $\mathbb{R}^n$ (with the Euclidean norm $\|\cdot\|$), then LLL produces a basis $(b_1, b_2, \ldots, b_n)$ of $L$ satisfying (among other conditions)

$$\|b_1\| \leq 2^{(n-1)/4} \det(L)^{1/n} .$$

The LLL algorithm computes an LLL-reduced basis for $L$ in polynomial time in $n$, and in $\log B$ where $B$ is a bound on all $\|b_i\|^2$. The resulting $b_1$ is approximately as short as possible: $\|b_1\| \leq 2^{(n-1)/2} \min_{v \in L\setminus\{0\}} \|v\|$. Note that all our lattices will have integer coefficients.

Many variants of LLL have been designed for more speed and accuracy (e.g. [50, 42, 43]), but the original LLL algorithm suffices for our results.

### 3.1. **Bivariate Coppersmith.**

Theorem 3.1 describes the input and output of Coppersmith's method for finding small zeroes of integer bivariate polynomials [11, 12]. Coppersmith's algorithm is clarified in [16] and [7], and extended to the general multivariate case in [30], [13], [7], and [46].

**Theorem 3.1.** *Let $f(x, y) = \sum p_{i,j} x^i y^j \in \mathbb{Z}[x, y]$ be irreducible, of degree at most $\delta$ in $x$ and $y$, and suppose $f(x_0, y_0) = 0$ for some $|x_0| < X$, $|y_0| < Y$. If*

$$XY < \mathcal{W}^{2/(3\delta)} \quad \textit{where} \quad \mathcal{W} = \|f(xX, yY)\|^* := \max_{i,j} |p_{i,j}|X^i Y^j \ ,$$

*then we can find all such solutions $(x_0, y_0)$ in deterministic polynomial time in $\log \mathcal{W}$ and $\delta$.*

*Proof.* See Coron's treatment in [17]. □

In this article we will apply the special case of Theorem 3.1 where the polynomial $f$ is linear in each variable to find divisors of $N$. In another direction, but using the same techniques, Theorem 3.2 improves on a result of Lenstra [36].

**Theorem 3.2** (Coppersmith–Howgrave-Graham–Nagaraj [14])**.** *Let $0 \le r < s < N$ with $\gcd(r, s) = 1$ and $s \ge N^\alpha$ for some $\alpha > 1/4$. The number of divisors of $N$ that are congruent to $r \pmod{s}$ is in $O((\alpha - 1/4)^{-3/2})$. The divisors can be found in deterministic polynomial time.*

3.2. **Approximate common divisors.** One of the first applications of Coppersmith's method was to attack RSA moduli, factoring $N = p_1 p_2$ in polynomial time given half of the bits of $p_1$. The algorithmic presentation of these theorems used today is due to Howgrave-Graham [28], who later used this result to solve the *Approximate Common Divisor Problem* (ACDP) [29], which we formalize in Definition 3.3.

**Definition 3.3** (ACDP)**.** Given integers $A$ and $B$, and bounds $X$ and $D_0$ for which there exists at least one $(x, D)$ with $|x| \le X$ and $D > D_0$ such that $D \mid B$ and $D \mid (A + x)$, the ACDP is to find all such $(x, D)$.

Before going further, we must make the following very important observation (not present in [29]).

*Remark* 3.1. If $(x, D)$ is an ACDP solution for $(A, B, X, D_0)$, then so is $(x, D/z)$ for any divisor $z$ of $D$ such that $D/z > D_0$.

Howgrave-Graham gives two types of algorithms for solving ACDP instances in [29]. The first, using continued fractions, is described by Proposition 3.5 and Algorithm 1 (`ACD_CF`). The second approach, using LLL, is described by Theorem 3.6 and Algorithm 2 (`ACD_LLL`). Both algorithms run in deterministic polynomial time, unlike the algorithms for the *Generalized* Approximate Common Divisor problem (GACDL) also considered in [29].

As noted in [29], the continued fraction (`ACD_CF`) and lattice (`ACD_LLL`) approaches are subtly different: `ACD_CF` requires only a lower bound on one exponent $\alpha$, but `ACD_LLL` requires some relation between two exponents, $\alpha$

and $\beta$ (or $\epsilon$). We will encounter this difference in §5.4. Similar phenomena appear in the context of *implicit factorization* (e.g. [40, 48, 22, 49]), but in these cases the two exponents can be handled more easily.

### 3.3. Computing approximate common divisors via continued fractions.

The following is taken from [29]. We include the proof here, because we will need to be precise about what the algorithm actually outputs.

**Proposition 3.4** (Howgrave-Graham)**.** *Given integers $A < B$, and real $\alpha > 1/2$, we can find all integers $|x_0| < X = B^{2\alpha-1}/2$ such that there exists $D > B^\alpha$ dividing both $A + x_0$ and $B$, or decide that no such $x_0$ exists, in deterministic polynomial time in* $\log B$*.*

*Proof.* Suppose $(x, D)$ is one of the desired ACDP solutions: then $D \mid B$ and $D \mid (A + x)$, with $D > B^\alpha$ and $|x| \leq B^{2\alpha-1}/2$. Write $a' = (A + x)/D$ and $b' = B/D$; then $b' < B^{1-\alpha}$, from which

$$\left|\frac{A}{B} - \frac{a'}{b'}\right| = \frac{|x|}{B} < \frac{1}{2(b')^2}\ .$$

The classical theory of continued fraction approximations tells us that $a'/b'$ must be one of the convergents $(g_1/h_1, g_2/h_2, \ldots)$ of $A/B$ (see for example [37, Theorem 9.10]). We note that the convergents are obtained in reduced form, that is, with $\gcd(g_i, h_i) = 1$; the $h_i$ are strictly increasing; and the last term is $h_k = B$. Last but not least, this sequence is finite and has polynomial size in $\log B$ (this is closely related to the computation of $\gcd(A, B)$, and can be done in deterministic polynomial time [31, 52]).

A solution yields $(A + x)/B = g_i/h_i$: that is, $h_i(A + x) = Bg_i$. Since $g_i$ and $h_i$ are coprime, this implies that $g_i \mid A + x$. Put $D = (A + x)/g_i$. Now $h_iD = B$, and $h_i$ must divide $B$. If this is the case, then we have recovered the ACDP solution $(x, D) = (Dg_i - A, B/h_i)$. We can stop as soon as $h_i \geq B^{1-\alpha}$, because such $h_i$ cannot yield $D > B^\alpha$. □

*Remark* 3.2. As noted in [29], if our problem requires $A > B$, then we can replace $A$ by using $q$ such that $A - qB < B$.

**Proposition 3.5.** *Given integers $A < B$, Algorithm 1 (*`ACD_CF`*) computes all integers $|x_0| < X = B^{2\alpha-1}/2$ for some $\alpha > 1/2$ such that there exists $D > B^\alpha$ dividing both $A + x_0$ and $B$, or reports that no such $x_0$ exists. The algorithm runs in deterministic polynomial time in* $\log B$*.*

*Proof.* It is enough to use Proposition 3.4 for all possible $\alpha > 1/2$, or equivalently test all convergents of $A/B$ (as noted above, since $A/B$ is rational, its sequence of convergents is finite, and has polynomial length). Algorithm 1 begins by computing these convergents. □

*Remark* 3.3. The bound $X$ in Proposition 3.5 can be relaxed to $B^{2\alpha-1}$ (without the factor of $1/2$) if we use intermediate convergents, but asymptotically this has no real importance.

*Remark* 3.4. If we want *all* solutions $(x, D)$, then we have to include all solutions coming from divisors of $D$, in the sense of Remark 3.1—but finding the divisors of $D$ would imply resorting to non deterministic and/or non-polynomial-time algorithms.

**Algorithm 1:** Computing approximate common divisors using continued fractions.

```
Function ACD_CF($A$, $B$)
    Input  : $A < B$
    Output: The set of solutions $(x, D)$ to the ACDP for $(A, B)$ (so
            $D \mid (A + x)$ and $D \mid B$) with $|x| < X := \frac{1}{2}B^{2\alpha-1}$ and
            $D > D_0 := B^{\alpha}$ for some $\alpha > 1/2$.
    $(g_0/h_0, \ldots, g_n/h_n) \leftarrow$ continued fraction convergents of $A/B$
    $\mathcal{R} \leftarrow \emptyset$
    for $i \leftarrow 0$ to $n$ do
        if $h_i \mid B$ (and $h_i > 1$) then
            $D \leftarrow B/h_i$
            $x \leftarrow Dg_i - A$
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{(x, D)\}$
    return $\mathcal{R}$
```

3.4. **Computing approximate common divisors via lattice reduction.** Theorem 3.6 enlarges the set of $\alpha$ for which we can find the factorization of $N$. The proof of correctness can be found in [29]; optimal parameters are given in Algorithm 2. Remark 3.4 applies here too.

**Theorem 3.6** (Howgrave-Graham). *Given integers $A < B$, and $\alpha$ in $(1/2, 1)$ and $\beta$ in $(0, \alpha^2)$, Algorithm 2 (*`ACD_LLL`*) computes all $x$ such that there is some $D$ with $(x, D)$ a solution to the ACDP for $(A, B)$ with $|x| < X := B^{\beta}$, $D > D_0 = B^{\alpha}$, in deterministic polynomial time in $\log B$ and $1/\epsilon$ where $\epsilon = \alpha^2 - \beta$.*

3.5. **Algorithms to refine partial factorizations.** Many algorithms (including some given below) return nontrivial divisors of $N$, rather than complete prime factorizations. We can improve the quality of these partial factorizations using some basic auxiliary algorithms, that all run in deterministic polynomial time. The following two algorithms are taken from [3].

`Refine` takes a set of integers $\{M_1, \ldots, M_k\}$, and returns a set of pairs $(N_i, e_i)$ with each $N_i > 1$ and $e_i > 0$, and with the $N_i$ all pairwise coprime, such that $\prod_i M_i = \prod_i N_i^{e_i}$. This can be done by iterating the rewriting formula

$$M_1 M_2 = (M_1/d)(d^2)(M_2/d) \quad \text{where} \quad d = \gcd(M_1, M_2)\,.$$

**Algorithm 2:** Approximate common divisors using LLL

```
Function ACD_LLL($A$, $B$, $\alpha$, $\beta$)
    Input  : $A < B$ and $\alpha \in (1/2, 1)$, $\beta \in (0, \alpha^2)$
    Output: The set of solutions $(x, D)$ to the ACDP for $(A, B)$ (so
            $D \mid (A + x)$ and $D \mid B$) with $|x| < X := B^\beta$ and
            $D > D_0 := B^\alpha$.
    $h \leftarrow \lceil \alpha(1 - \alpha)/\epsilon \rceil - 1$ where $\epsilon = \alpha^2 - \beta > 0$
    $u \leftarrow \lceil h\alpha \rceil$
    $L \leftarrow$ the $(h + 1)$-dimensional lattice of $\tilde{p}_i$-coefficients defined in
      the proof of Theorem 3.6
    $(v_0, \ldots, v_h) \leftarrow$ ShortVector($L$)        // Use LLL; each $v_i$ is
      divisible by $X^i$
    $P(Z) \leftarrow \sum_{i=0}^{h} (v_i/X^i) Z^i$
    $\mathcal{X} \leftarrow$ integer roots of $P(Z)$
    $\mathcal{R} \leftarrow \emptyset$
    for $x \in \mathcal{X}$ do
        $D \leftarrow \gcd(A + x, B)$
        if $1 < D < B$ then
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{(x, D)\}$
    return $\mathcal{R}$
```

Faster algorithms for `Refine` appear in [5] and [6].

`CleanDivisors` takes an integer $m$ and a list of divisors $(d_1, \ldots, d_k)$ of $m$, and returns a set of pairs $(m_i, e_i)$ such that $m = \prod_i m_i^{e_i}$ where the $m_i$ are pairwise coprime and such that each $d_i = \prod_j m_j^{e_{i,j}}$ for some $e_{i,j} \geq 0$. This can be done by applying `Refine` to $\{d_1, m/d_1, \ldots, d_k, m/d_k\}$, which yields $\{(n_1, f_1), \ldots, (n_\ell, f_\ell)\}$ such that $\prod_{i=1}^{\ell} n_i^{f_i} = m^k$ with the $n_i$ pairwise coprime. The $f_i$ are all multiples of $k$, so the result is $\{(n_1, f_1/k), \ldots, (n_\ell, f_\ell/k)\}$.

## 4. Finding particular divisors of an integer

This section describes algorithms that find a large divisor $D$ of $N$ if $(D - z) \mid M$ for an auxiliary integer $M$ and some small $z$. We use the simplest case, where $D = p$ is prime and $z = 1$ (resp. $z = -1$) for factoring with $\Phi$ (resp. $\Sigma$) in §5, but we think that these more general results have independent interest.

### 4.1. Factoring with unknown difference.

First, consider the search for divisors $D$ of $N$ such that that $(D - z) \mid M$ where $M$ is *given* and a small $z \neq 0$ is *unknown*. For our needs (factoring $N$), the interesting case has $\gcd(N, M) = 1$. We can compute such $D$ in deterministic polynomial time by reduction to an ACDP instance as follows. Let $y = M/(D - z)$, so $M = y(D - z) = yD - yz$; computing the product $yz$ leads to the divisor

$D$ by computing $\gcd(N, M+yz)$ since $\gcd(N,M) = 1$. But $x = yz$ is the solution of the modular equation $M + x \equiv 0 \pmod{D}$, and thus $(x, D)$ is a solution to ACDP for $(A, B) = (M, N)$. At this point, we finish using the results of §3.

**Theorem 4.1.** *Let $1/2 < \alpha < 1$ be a real number. Let $N$ and $M$ be* coprime *integers and put $M = N^\theta$ with $1/2 < \alpha < \theta < 1$. Suppose there exists $D > N^\alpha$ such that $D \mid N$ and $(D - z) \mid M$ for a small* unknown *integer $z \neq 0$, with $|z| \leq N^\varepsilon$ such that $0 < \varepsilon < \alpha$. Then we can compute $D$ in deterministic polynomial time in the following two cases:*

1. $\varepsilon \leq 3\alpha - 1 - \theta$*;*
2. $\varepsilon < \alpha^2 + \alpha - \theta$*.*

*Proof.* Write $\beta = \log(M/(D-z))/\log N \approx \theta - \alpha$, so that $y = N^\beta$ and $x = yz = N^{\theta+\varepsilon-\alpha}$. In Case (1) we have $\theta+\varepsilon-\alpha \leq 2\alpha-1$, and Proposition 3.5 applies. In Case (2), since $|z| < N^{\alpha^2-\beta}$, we get $|x| \leq N^{\alpha^2}$, and we can compute $x$ in deterministic polynomial time by Theorem 3.6. □

4.2. **Factoring with known difference.** Now we consider the opposite case: finding $D \mid N$ such that $(D - z) \mid M$ where $z$ is *known*. In our applications with $\Phi$ and $\Lambda$, we take $z = 1$; with $\Sigma$ we take $z = -1$. In full generality, provided $z$ is especially small, we use Coppersmith's bivariate method from Theorem 3.1 to obtain the following result.

**Theorem 4.2.** *Let $N$, $u$, and $M$ be integers with $u \neq 0$ and $|u| = N^\varepsilon$, and $M = N^\theta$ with $0 \leq \varepsilon < \theta < 1$. Fix $0 \leq \varepsilon < \alpha < \theta$. Suppose there exists $D > N^\alpha$ such that $D \mid N$ and $(D - u) \mid M$. Then Algorithm 3 computes $D$ in deterministic polynomial time if*

$$\alpha > \frac{1}{4}(1+\theta)\,. \tag{1}$$

*Proof.* Rewrite the problem as $N = x_0 D$ and $M = y_0(D-u)$, so $M + uy_0 = y_0 D$. Eliminating $D$, we see that $(x_0, y_0)$ is a zero of $f(x,y) = Ny - x(M + uy) = -Mx + Ny - uxy$. If $D > N^\alpha$, then $x_0 < N^{1-\alpha}$ and $y_0 < N^{\theta-\alpha}$ are both small, and we can use Theorem 3.1. First, as in [16], we let

$$f^*(x,y) := f(x, y+1) = N - (M+u)x + Ny - uxy.$$

Now $f^*$ is irreducible, and linear in $x$ and $y$, so it meets the conditions of Theorem 3.1 with $\delta = 1$; and $f^*(0,-1) = 0$. Assume $|x_0| < X$ and $|y_0| < Y$. The crucial bound is

$$\mathcal{W} = \|f^*(xX, yY)\|^* = \max(N, (M+u)X, NY, XY).$$

Using $(X, Y) = (N^{1-\alpha}, N^{\theta-\alpha})$ gives $XY = N^{1+\theta-2\alpha}$ and

$$\mathcal{W} = \max(N, N^{1+\theta-\alpha}, N^{1+\theta-2\alpha}) = N^{1+\theta-\alpha}.$$

Ignoring small constants, we want

$$1 + \theta - 2\alpha < \frac{2}{3}(1 + \theta - \alpha)$$

which implies $\alpha > \frac{1}{4}(1+\theta)$; the result follows. □

*Remark* 4.1. A weaker but simpler result can be obtained using Coron's algorithm, as in [16, §2]: if we use $f^*(x,y) = N - (M+u)x + Ny - uxy$, then $\alpha > (1+\theta)/3$ is enough to recover $D$.

*Remark* 4.2. If $u < 0$, then we have $\theta \geq \alpha$ which leads to $\theta > 1/3$.

**Corollary 4.3.** *Using the notation of Theorem 4.2: we can recover $D$ in deterministic polynomial time provided $D > N^{1/2}$.*

*Proof.* It suffices to observe that $\alpha > 1/2 > (1+\theta)/4$ for $0 \leq \theta < 1$. □

**Algorithm 3:** Factoring with known difference

```
Function FactoringWithKnownDifference($N$, $M$, $u$, $\alpha$)
    Input : Positive integers $N$ and $M$, an integer $u$, and a real
            number $\alpha$ such that $(1+\theta)/4 < \alpha < \theta$ where
            $\theta := \log M / \log N$
    Output: $\{D > N^\alpha : D \mid N$ and $(D-u) \mid M\}$
    $f^* \leftarrow N - (M+u)x + Ny - uxy$ in $\mathbb{Z}[x,y]$
    $\mathcal{R} \leftarrow$ BivariateCoppersmith($f^*, X, Y$) where $X = N^{1-\alpha}$ and
     $Y = N^{\theta-\alpha}$
    $\mathcal{D} \leftarrow \emptyset$
    for $(x,y)$ in $\mathcal{R} \setminus \{(0,-1)\}$ do
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{N/x\}$
    return $\mathcal{D}$
```

## 5. Factoring with the $\Phi$, $\Lambda$, and $\Sigma$ oracles

We now return to factoring with oracles. We treat the closely-related problems of factoring with $\Phi$, $\Lambda$, or $\Sigma$ simultaneously here, before treating $\mathcal{O}$ in §6.2. We consider odd $N$, since detecting and removing powers of 2 is easy. Ordering the prime divisors of $N$ by decreasing size, we write

$$N = \prod_{i=1}^{\omega(N)} p_i^{e_i} \quad \text{with primes} \quad p_1 > p_2 > \cdots > p_{\omega(N)} > 2\,.$$

To simplify the exposition, the function associated with an oracle $\varpi$ will be denoted by $\mathcal{F}(\varpi)$ (e.g., $\mathcal{F}(\Phi) = \varphi$).

5.1. **Reduction to the squarefree case.** We begin by reducing to the case of squarefree $N$: that is, $e_1 = \cdots = e_{\omega(N)} = 1$.

**Theorem 5.1** (Landau [32]). *Given $N$ and $\varpi \in \{\Phi, \Lambda, \Sigma\}$, Landau's algorithm returns in deterministic polynomial time a list $(N_1, \ldots, N_r)$ such that $N = N_1 N_2^2 \cdots N_r^r$, each $N_i$ is squarefree or 1, and the $N_i$ are pairwise coprime using $O(\omega(N))$ calls to $\varpi$.*

5.2. **Reduction to the case** $\gcd(N, \mathcal{F}(\varpi)(N)) = 1$**.** Suppose $N$ is squarefree. For $\varpi = \Phi$, $\Lambda$, or $\Sigma$, if $\gcd(N, \mathcal{F}(\varpi)(N)) > 1$ then we obtain a nontrivial divisor $d$ of $N$, and we can combine the factorizations of $d$ and $N/d$ recursively. Thus, we reduce to the problem of factoring squarefree $N$ where $\gcd(N, \mathcal{F}(\varpi)(N)) = 1$.

5.3. **Products of two primes.** It is well-known that we can factor $N = p_1p_2$ given $\varphi(N)$, as we recall in Lemma 5.2. This immediately yields Algorithm 4 (`FactorizationWithPhi2`), which factors a squarefree integer $N$ with $\omega(N) = 2$ given $M = \varphi(N)$. Rephrased, this gives also that oracle $\Phi$ can answer the decision problem of determining whether $\omega(N) = 2$.

**Lemma 5.2.** *If $N$ is a product of two distinct primes, then the two primes are*

$$s/2 \pm \sqrt{(s/2)^2 - N} \quad \textit{where} \quad s := N + 1 - \varphi(N) \ .$$

*Proof.* If $N = p_1p_2$ with $p_1$ and $p_2$ prime, then $\varphi(N) = (p_1 - 1)(p_2 - 1) = N - (p_1 + p_2) + 1$; so $s = p_1 + p_2$, and $p_1$ and $p_2$ are the roots of the quadratic equation $X^2 - sX + N$. □

**Algorithm 4:** Factoring a 2-factor integer using $M = \varphi(N)$

```
Function FactorizationWithPhi2($N$, $M$)
    Input  : $N$ and $M = \varphi(N)$, where $N$ is squarefree
    Output: $\{p_1, p_2\}$ if $N$ is the product of two distinct primes, or $\emptyset$
    $s \leftarrow N + 1 - M$
    $\Delta \leftarrow s^2 - 4N$            // $\Delta =$ discriminant of $X^2 - sX + N$
    if $\Delta$ is not square then
        return $\emptyset$
    $p_1 \leftarrow \frac{1}{2}(s + \sqrt{\Delta})$
    $p_2 \leftarrow N/p_1$
    return $\{p_1, p_2\}$
```

To convert Algorithm 4 into an algorithm taking $\lambda(N)$ instead of $\varphi(N)$, we use Lemma 5.3, which shows that when $\omega(N) = 2$, we can efficiently compute $\varphi(N)$ from $\lambda(N)$. Thus, any algorithm calling $\Phi$ can be immediately transformed into an algorithm making the same number of calls to $\Lambda$. In particular, Algorithm 4 can be used with $M = \lambda(N) \cdot \gcd(N - 1, \lambda(N))$ instead of $\varphi(N)$.

**Lemma 5.3.** *If $N = p_1p_2$ is a product of two distinct primes, then $\varphi(N) = \lambda(N) \cdot \gcd(N - 1, \lambda(N))$.*

*Proof.* Suppose $N = p_1p_2$. Write $g = \gcd(p_1 - 1, p_2 - 1)$; then $p_1 - 1 = gq_1$ and $p_2 - 1 = gq_2$ with $\gcd(q_1, q_2) = 1$. Now

$$\lambda(N) = (p_1 - 1)(p_2 - 1)/g = gq_1q_2 \ ,$$

from which $\gcd(N-1, \lambda(N)) = g \cdot \gcd(gq_1q_2 + q_1 + q_2, q_1q_2)$, but $\gcd(gq_1q_2 + q_1 + q_2, q_1q_2) = 1$. □

Finally, for the oracle $\Sigma$, given $\sigma(N) = N + 1 + p_1 + p_2$, we immediately recover $p_1 + p_2$ and then compute $p_1$ and $p_2$ as above.

5.4. **Products of more than two primes.** Returning to the general squarefree case, suppose

$$N = p_1 \cdots p_k \quad \text{with primes} \quad p_1 > \cdots > p_k > 2 \quad \text{and} \quad \omega(N) = k \geq 3\,.$$

The relative sizes of the $p_i$ will be important in what follows. We set

$$\alpha_i := \log_N p_i\ , \quad \text{so} \quad p_i = N^{\alpha_i}\ .$$

Clearly $\sum_{i=1}^{k} \alpha_i = 1$ and $1 > \alpha_1 > \cdots > \alpha_k > 0$; so, in particular, $\alpha_1 > 1/k$ and $\alpha_k < 1/k$.

We first rephrase Corollary 4.3 to show that *unbalanced* numbers (having a large prime factor) are easy to factor with $\varpi \in \{\Phi, \Lambda, \Sigma\}$. In contrast, *compact* $N$ (with all prime factors $\leq N^{1/2}$) are harder to factor. This gives us a result already stated (in a simple form) as Theorem 1.1.

**Theorem 5.4.** *If $\omega(N) \geq 3$ and $\alpha_1 > 1/2$, then we can recover the divisor $D = p_1$ of $N$ in deterministic polynomial time in $\log(N)$ given $\mathcal{F}(\varpi)(N)$ for $\varpi \in \{\Phi, \Lambda, \Sigma\}$.*

*Proof.* We use $(D-1) \mid \varphi(N)$ (resp. $(D-1) \mid \lambda(N)$); the result follows directly from Corollary 4.3. The same holds for $\Sigma$ using $(D+1) \mid \sigma(N)$. □

*Remark* 5.1. When using $\lambda(N)$ in Theorem 5.4, we can recover $p_1$ in deterministic polynomial time provided $\alpha_1 > (1+\theta)/4$, where $\theta = \log \lambda(N)/\log N$. When $\theta$ is significantly smaller than 1, this gives a substantially lower bound on $\alpha_1$; but finding a condition analogous to Inequality (2) is not so easy in that case.

The results of §4 yield conditions on the $\alpha_i$ under which factors of $N$ can be computed with the algorithms of §3. Theorems 5.5 and 5.6 show that we can factor $N$ by solving ACDP instances if the $p_i$ satisfy certain relative size conditions. As a first step, Theorem 5.5 gives conditions for efficient factoring using Algorithm 5 (`SplitCF`), which applies `ACD_CF` using $\Phi$ or $\Sigma$.

**Theorem 5.5.** *Suppose $\omega(N) \geq 3$ and there exists $1 \leq r < \omega(N)$ such that*

$$\alpha_r \geq 2 \sum_{i=r+1}^{\omega(N)} \alpha_i\,. \tag{2}$$

*Then `ACD_CF` recovers the factor $D = \prod_{i=1}^{r} p_i$ in deterministic polynomial time given $\varphi(N)$ or $\sigma(N)$.*

*Proof.* Write $\alpha = \sum_{i=1}^{r} \alpha_i$. The hypothesis implies $\alpha > 1/2$; otherwise $\alpha_r \geq 2(1-\alpha)$ and $\alpha \leq 1/2$, hence $\alpha_r \geq 1$, which is impossible. Expanding the formula for $\varphi(N)$ yields $\varphi(N) = DQ_1 - N/p_r + Q_2$ for some $Q_1$ and $Q_2$. If

$x = N/p_r - Q_2$, then $(x, D)$ is a solution to the ACDP for $(A, B) = (\varphi(N), N)$ with $(M, X) = (N^\alpha, N^{2\alpha-1})$, and `ACD_CF` will find $(x, D)$ because $\alpha > 1/2$. In this case $x \approx N/D = N^{1-\alpha}$, and the condition becomes $1 - \alpha_r \le 2\alpha - 1$, which yields Inequality (2).

The same reasoning is valid for $\sigma(N)$, simply changing signs to get $\sigma(N) = DQ_1 + N/p_r + Q_2$. (Strictly speaking, we should use $(A, B) = (\sigma(N) - N, N)$ to get $A < B$: see Remark 3.2.) □

**Algorithm 5:** Splitting an integer using ACDCF

```
Function SplitCF($N$, $\varpi$)
    Input  : $N$ to be factored using oracle $\varpi \in \{\Phi, \Sigma\}$
    Output: $\emptyset$ or a set of pairs $(M_i, e_i)$, with the $M_i$ pairwise
            coprime and $N = \prod_i M_i^{e_i}$
    $M \overset{\varpi}{\longleftarrow} \mathcal{F}(\varpi)(N)$
    $sgn \leftarrow \begin{cases} -1 & \text{if } \varpi = \Phi \\ 1 & \text{if } \varpi = \Sigma \end{cases}$
    if $M = N + sgn$ then                                    // $N$ is prime
        return $\{(N, 1)\}$
    $\mathcal{A} \leftarrow$ ACD_CF$(M, N)$
    for $(x, D)$ in $\mathcal{A}$ do
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{D, N/D\}$
    if $\mathcal{D} = \emptyset$ then
        return $\emptyset$
    return CleanDivisors($N$, $\mathcal{D}$)
```

We can go further using `ACD_LLL` instead of `ACD_CF`. Theorem 5.6 is the corresponding analogue of Theorem 5.5.

**Theorem 5.6.** *If there exist $\alpha$ in $(1/2, 1)$ and $\beta$ in $(0, \alpha^2)$ such that $\alpha \le \sum_{i=1}^{r} \alpha_i$ and $1 - \alpha_r \le \beta$ for some $1 \le r < \omega(N)$, then we can recover the divisors $D = p_1 \cdots p_r$ and $N/D = p_{r+1} \cdots p_{\omega(N)}$ of $N$ in deterministic polynomial time given $\alpha$ and $\beta$, using $\Phi$ or $\Sigma$.*

*Proof.* Write

$$\varphi(N) = [(p_1 - 1)(p_2 - 1) \cdots (p_r - 1)]K = DK - p_1 p_2 p_{r-1} p_{r+1} \cdots p_{\omega(N)} + E$$

where $E$ is negligible with respect to $N/p_r$. We obtain

$$\varphi(N) = DK - (N/p_r) + E.$$

Now, Theorem 3.6 will use `ACD_LLL` given $A = \varphi(N)$, $B = N$, $\alpha \le \sum_{i=1}^{r} \alpha_i$, and $\beta \le 1 - \alpha_r$ to find $(x, D)$ where $D = p_1 p_2 \ldots p_r$ and $x = N/p_r - E \approx N^{1-\alpha_r}$.

The same conclusion holds for $\sigma(N) = DK + N/p_r + E'$. □

Theorem 5.6 is difficult to apply directly, because of the subtlety alluded to in §3.4: it is not enough to simply know that $\alpha$ and $\beta$ satisfying the bounds *exist*, because we need to use them as parameters to `ACD_LLL`. On the other hand, `ACD_LLL` does not need their *exact* values (indeed, if we knew the exact value for $\beta = 1 - \alpha_r$, then we would already know the prime factor $p_r = N^{\alpha_r}$). If we can guess that a suitable $r$ exists, then we can give a lower bound for $\alpha_r$ implying a lower bound for $\alpha$ and an upper bound for $\beta$ that allow us to apply `ACD_LLL`. While the bounds may be far from the optimal values of $\alpha$ and $\beta$, thus yielding sub-optimal performance for `ACD_LLL`, the solution is still polynomial time, and it allows us to factor some integers that `ACD_CF` cannot.

**Definition 5.7.** For each positive integer $r$, we define a constant

$$\overline{\alpha}_r := \frac{-1 + \sqrt{1 + 4r^2}}{2r^2}\,.$$

The first few of these constants are

$$\begin{aligned}\overline{\alpha}_1 &= (-1 + \sqrt{5})/2 \approx 0.618\,,\\ \overline{\alpha}_2 &= (-1 + \sqrt{17})/8 \approx 0.3904\,,\\ \overline{\alpha}_3 &= (-1 + \sqrt{37})/18 \approx 0.2824\,.\end{aligned}$$

**Lemma 5.8.** *If $\alpha_r > \overline{\alpha}_r$ for some $0 < r < \omega(N)$, then $r$, $\alpha = r\overline{\alpha}_r$, and $\beta = 1 - \overline{\alpha}_r$ meet the conditions of Theorem 5.6.*

*Proof.* Let $\widetilde{\alpha} = \sum_{i=1}^{r} \alpha_i$ and $\widetilde{\beta} = 1 - \alpha_r$; these are the ideal values for $\alpha$ and $\beta$ when applying Theorem 5.6. Clearly $\widetilde{\alpha} > r\alpha_r$. We can therefore use Theorem 5.6 with $\alpha = rX$ and $\beta = 1 - X$ for any $X \le \alpha_r$ such that $1 - X < (rX)^2$; that is, as long as $X > \overline{\alpha}_r$. Moreover, $1/2 < r\overline{a}_r < 1$ for all $r > 0$. Hence $(\alpha, \beta) = (r\overline{\alpha}_r, 1 - \overline{\alpha}_r)$ meets the conditions of the theorem for the given $r$. □

We emphasize that Lemma 5.8 only gives a *sufficient* condition for suitable $\alpha$ and $\beta$, but we can use it to turn the proof of Theorem 5.6 into an effective algorithm.

**Theorem 5.9.** *Fix an integer $R > 1$. If there exists an $0 < r < \min(R + 1, \omega(N))$ for which $\alpha_r \ge \overline{\alpha}_r$, then Algorithm 6 (*`SplitLLL`*) recovers the divisor $D = p_1 \cdots p_r = N^{\alpha}$ of $N$ in deterministic polynomial time using $\Phi$ or $\Sigma$.*

*Proof.* Algorithm 6 tries to factor $N$ by calling `ACD_LLL` using increasing values of $r$ (up to and including $\min(R + 1, \omega(N))$, which in any case is trivially bounded by $\log_2 N$, though much smaller values of $R$ are more interesting), with the bounds for $\alpha$ and $\beta$ suggested by Lemma 5.8. The result therefore follows from $R$ serial applications of Theorem 5.6. □

**Algorithm 6:** Splitting an integer using ACDL

```
Function SplitLLL($N$, $\varpi$, $R$)
    Input  : $N$ to be factored using oracle $\varpi \in \{\Phi, \Sigma\}$, and a bound
             $R > 1$ on putative $r$
    Output: $\emptyset$ or a set of pairs $(M_i, e_i)$, with the $M_i$ pairwise
             coprime and $N = \prod_i M_i^{e_i}$
    $M \xleftarrow{\varpi} \mathcal{F}(\varpi)(N)$
    $\mathcal{D} \leftarrow \emptyset$
    for $r \leftarrow 1$ to $R$ do
        $\overline{\alpha}_r \leftarrow (-1 + \sqrt{1 + 4r^2})/(2r^2)$
        $\mathcal{A} \leftarrow$ ACD_LLL$(M, N, r\overline{\alpha}_r, 1 - \overline{\alpha}_r)$          // use
         $(\alpha, \beta) = (r\overline{\alpha}_r, 1 - \overline{\alpha}_r)$
        for $(x, D)$ in $\mathcal{A}$ do
            $\mathcal{D} \leftarrow \mathcal{D} \cup \{(D, N/D)\}$
    if $\mathcal{D} = \emptyset$ then
        return $\emptyset$
    return CleanDivisors($N$, $\mathcal{D}$)
```

5.5. **Products of exactly three primes.** We can say a little more for the special case of squarefree $N$ with $\omega(N) = 3$. The difficult part is in breaking $N$: once a non-trivial divisor is found, we are left with a prime and a product of two primes that can be easily factored recursively using the oracle.

Write

$$N = p_1 p_2 p_3 \quad \text{where} \quad p_1 > p_2 > p_3 \ .$$

As usual, we set $\alpha_i = \log_N p_i$; by definition, $1 > \alpha_1 > \alpha_2 > \alpha_3 > 0$, and $\alpha_3$ is completely determined by $(\alpha_1, \alpha_2)$ because $\alpha_1 + \alpha_2 + \alpha_3 = 1$. Lemma 5.10 defines the polygon in the $(\alpha_1, \alpha_2)$-plane corresponding to the domain of validity of the exponents for $\omega(N) = 3$.

**Lemma 5.10.** *With $N$ and $\alpha_i = \log_N p_i$ defined as above, $(\alpha_1, \alpha_2)$ lies in the region of the $(\alpha_1, \alpha_2)$-plane defined by the inequalities*

$$0 < \alpha_2 < \alpha_1 \ , \qquad \alpha_1 + \alpha_2 < 1 \ , \qquad \alpha_1 > 1/3 \ , \qquad 2\alpha_1 + 3\alpha_2 > 3/2 \ .$$

*Proof.* The first three inequalities follow immediately from the definition of the $\alpha_i$. For the last, if $2\alpha_1 + 3\alpha_2 \leq 3/2$ then $\alpha_2 \leq (3/2 - 2\alpha_1)/3$, whence $1 - \alpha_1 = \alpha_2 + \alpha_3 < 2\alpha_2 \leq 2/3(3/2 - 2\alpha_1)$, so $\alpha_1/3 < 0$, which is impossible. □

Figure 1 depicts the values of $(\alpha_1, \alpha_2)$ that our methods can tackle, shading in various regions of the polygon of Lemma 5.10. Each result applies only to the *interior* of the corresponding region, and does not apply to points on the boundary lines. We can factor $N$ using $\Phi$ (resp. $\Sigma$) with

- Theorem 5.4 when $\alpha_1 > 1/2$, so $(\alpha_1, \alpha_2)$ is in the diagonally shaded polygon;
- Theorem 5.5 with $r = 2$ when $\alpha_2 \geq 2\alpha_3$, which translates as $2\alpha_1 + 3\alpha_2 \geq 2$, so $(\alpha_1, \alpha_2)$ is in the horizontally shaded polygon with vertices $(2/5, 2/5)$, $(1/2, 1/2)$, $(1/2, 1/3)$;
- Theorem 5.9 with $r = 2$ when $\alpha_2 > \overline{\alpha}_2$, so $(\alpha_1, \alpha_2)$ is in the tiny black triangle with vertices $(\overline{\alpha}_2, \overline{\alpha}_2)$, $(2/5, 2/5)$, $(1 - 3\overline{\alpha}_2/2 = 0.415, \overline{\alpha}_2)$.

The grey polygon with vertices $(1/3, 5/18)$, $(1/3, 1/3)$, $(\overline{\alpha}_2, \overline{\alpha}_2)$, $(1-3\overline{\alpha}_2/2, \overline{\alpha}_2)$, $(1/2, 1/3)$, and $(1/2, 1/6)$ is the zone where we cannot prove deterministic polynomial-time factorization.

A necessary condition to apply Theorem 5.6 in our case for $r = 2$ ($r = 1$ being uninteresting) is

$$1 - \alpha_2 < (\alpha_1 + \alpha_2)^2,$$

or

$$\alpha_2 > f(\alpha_1) := \frac{-(2\alpha_1 + 1) + \sqrt{4\alpha_1 + 5}}{2}.$$

The function $f$ is decreasing on $[0, 1]$ and is smaller than $\alpha_2$ for $\alpha_1 \geq (\sqrt{17} - 1)/8 \approx 0.3904$; note that $f(1/2) = (\sqrt{7} - 1)/2 \approx 0.3229$. This is the dash-dotted line, which corresponds to a sharp limit on using this theorem.

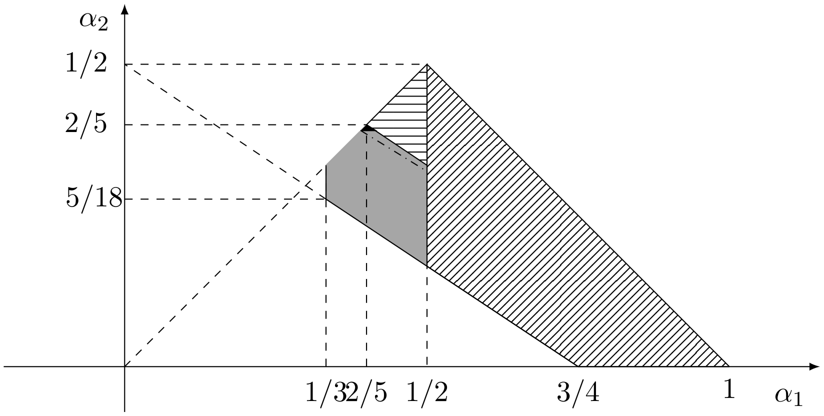

Figure 1. Cases covered by our results when $\omega(N) = 3$.

5.6. **Numerical examples.** We use Algorithms 1 (ACD_CF) and 2 (ACD_LLL) to factor various $N$ given $\varphi(N)$ or $\lambda(N)$. The algorithms succeed when the divisors of $N$ satisfy the required properties.

We start with a numerical example for each sub-region in Figure 1).

*Example* 5.6.1 (SplitCF with $\Phi$). Consider an attempt to factor

$$N = 143000000000000000000000000045617$$

using `SplitCF`. The oracle $\Phi$ tells us that

$$\varphi(N) = 12000000000000000000000000038160\,.$$

Applying `ACD_CF` with $A = \varphi(N)$ and $B = N$ reveals that $N$ has a divisor

$$D = 100000000000000000000000000319\,,$$

which turns out to be prime; the cofactor is $143 = 13 \cdot 11$. In this case, $\alpha_1 = 0.93082\ldots > 1/2$.

*Example* 5.6.2. Let us factor

$$N = 215453441884154813899608536725827949716396214692299863$$

using `SplitCF` again. The oracle $\Phi$ tells us that

$$\varphi(N) = 215453439729720123839763043257007943520875035231793568\,.$$

Applying `ACD_CF` with $A = \varphi(N)$ and $B = N$ reveals that $N$ has a divisor

$$D = 2154434690059745273387380265425792937208898747\,,$$

which has two prime factors. In this case $(\alpha_1, \alpha_2, \alpha_3) = (0.45, 0.4, 0.15)$, a point in the horizontally shaded part.

*Example* 5.6.3. Let us factor

$$N = 14300000027170000072930000138567$$

with `SplitCF`. The oracle $\Phi$ gives

$$\varphi(N) = 12000000021600000006000000108000\,,$$

and then `ACD_CF` with $(A, B) = (\varphi(N), N)$ finds a divisor

$$D = 100000000190000000510000000969$$

with two prime factors 10000000019 and 10000000000000000051, and the cofactor $N/D = 143$. We have $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (0.610, 0.321, 0.0358, 0.033)$ and Inequality (2) is satisfied for $r = 2$.

*Example* 5.6.4 (`SplitLLL` with $\Phi$). Let us factor

$$N = 5872731058374808693660010068837$$

with `SplitLLL`. The oracle $\Phi$ gives

$$\varphi(N) = 5872725180353869744164863505600\,,$$

and then `ACD_LLL` with $(A, B) = (\varphi(N), N)$ finds a divisor

$$D = 5878015394214207265992137\,,$$
$$x = 5544735287880571100\,,$$

and $D = 1425101895589 \cdot 4124628149333$, the third factor being 999101. We have $(\alpha_1, \alpha_2, \alpha_3) = (0.410, 0.395, 0.195)$ which is in the tiny triangle.

*Example* 5.6.5 (Factoring with $\Sigma$). Let us factor

$$\begin{aligned} N = {} & 2682776312933147882428349713219285333356964534315603933540 \\ & 90095217359233 \end{aligned}$$

using `SplitLLL`. The oracle $\Sigma$ tells us that

$$\begin{aligned} \sigma(N) = {} & 2682803140502033115557437331732180804000067423327043796969 \\ & 97748284763200\,. \end{aligned}$$

Trying $r = 2$, and calling `ACD_LLL` with $(A, B) = (\sigma(N) - N, N)$ and $(\alpha, \beta) = (2\overline{\alpha}_2, 1 - \overline{\alpha}_2)$ (implying lattice parameters $(h, u) = (25, 20)$), we find a solution

$$\begin{aligned} D &= 610540229658532834519888426420070208770724882201228981991\,, \\ x &= -23576265633281739760211511675892594424044680\,. \end{aligned}$$

*Example* 5.6.6 (Factoring with $\Lambda$). We apply `SplitCF` to

$$N = 14300000027170000072930000138567\,,$$

for which the oracle $\Lambda$ tells us that

$$\lambda(N) = 100000000180000000500000000900\,.$$

`ACD_CF` reveals a divisor

$$D = 100000000190000000510000000969$$

with two factors, 10000000019 and 10000000000000000051 (which we find recursively using `FactorizationWithPhi2`), and a cofactor $N/D = 143 = 11 \cdot 13$. We see that $\lambda(N)/\varphi(N) = 1/120$ (so $\lambda(N)$ is close to $\varphi(N)$, and the method may work), and

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (0.60984, 0.32097, 0.035754, 0.033425)\,.$$

## 6. Other oracles

Before concluding, we briefly survey some logical extensions to other oracles that do not yield useful results.

### 6.1. Using the factorization of $\varphi(N)$ or $\sigma(N)$.

Every odd prime $p \mid N$ is necessarily of the form $\delta + 1$ for some even $\delta \mid \varphi(N)$, so we can compute all prime factors $p$ of $N$ from the factors of $\varphi(N)$. Unfortunately this does not lead to polynomial-time algorithms, since the number of divisors of $\varphi(N)$ can be large, as shown in [39], and the same should hold for $\lambda(N)$ as well.

If $\omega(N) = k$ (for squarefree $N$, say), then the smallest prime factor of $N$ has $p_k < N^{1/k}$, so we might content ourselves with finding $p_k$ by enumerating divisors of $\varphi(N)$ less than $\varphi(N)^\alpha = N^{1/k}$. But this is not enough to get polynomial time, since this number can be lower bounded by $Cd(\varphi(N))^{-C'\alpha \log \alpha}$ for positive constants $C$ and $C'$ (see [53], studying a function introduced by P. Erdős in [21]).

We anticipate the same properties for $\sigma(N)$.

6.2. **Factoring with the order oracle.** We now consider factoring using the order oracle $\mathcal{O}$, whose quantum counterpart is the core of Shor's algorithm. As explained in [23], when $\lambda(N)$ has very few divisors, having the order of an element is enough to factor $N$. It is doubtful that we can find an algorithm for all integers, since $\lambda(N)$ may have a lot of divisors that cannot help factoring $N$.

Suppose we have the factorization of the order. As in §6.1, we might consider a modified $\mathcal{O}$ that yields not only the order $r$ of $a$ modulo $N$, but also the factorization of $r$. Algorithm 12 shows a straightforward way to make use of this additional information. If $N$ is not squarefree, then it is possible that $\gcd(r, N) \neq 1$, which gives us an easy factor of $N$ (hence the check in Line 7). Algorithm 12 fails, returning $\emptyset$, if $a$ has order $r$ modulo every prime factor $p_i$ of $N$, or if $r \mid p_i - 1$ for all $i$, which implies that all divisors of $N$ are congruent to 1 (mod $r$). Then, if $r > N^{1/4+\varepsilon}$, we can conclude in deterministic polynomial time using Theorem 3.2. Another approach for large $r$ is given in [55].

6.3. **Combining different oracles.** In another direction, having $\varphi(N)$ *and* $\sigma(N)$ yields the factorization of squarefree $N$ with three factors by finding the integer roots of the polynomial $(X - p_1)(X - p_2)(X - p_3) = X^3 + (N - (\sigma(N) + \varphi(N))/2)X^2 + ((\sigma(N) - \varphi(N))/2 - 1)X - N$, extending the result of `FactorizationWithPhi2`.

## 7. Conclusions

We have shown a range of partial results concerning the relationships between several elementary number theoretic functions and the integer factorization problem. In each case, we have used ideas coming from lattice reduction to improve what was known, while falling short of the goal of completely proving the sufficiency of these oracles for efficiently factoring all numbers.

As we saw in §6, adding more information does not pay: The complete factorizations of oracle values (or given $\varphi(N)$ *and* $\lambda(N)$, or even given their prime factorizations) *still* does not help factoring all $N$. These results may be surprising, but they show the fundamental difficulty of factoring.

**Acknowledgements.** We thank J. Shallit for sending us a copy of [38], and B. Źrałek for sending us a copy of his work [56]. We are grateful to W. George for bringing [10] to our attention. J.-L. Nicolas and G. Tenenbaum were kind enough to send us results related to Wolke's work. All algorithms were programmed and tested in Magma, and some computations were done in Maple.

## References

[1] Leonard M. Adleman and Kevin S. McCurley. Open problems in number theoretic complexity, II. In Leonard M. Adleman and Ming-Deh Huang, editors, *Algorithmic Number Theory*, pages 291–322, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[2] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Ann. of Math. (2)*, 160(2):781–793, 2004.

[3] Eric Bach, James Driscoll, and Jeffrey O. Shallit. Factor refinement. *J. Algorithms*, 15:199–222, 1993.

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

[5] Daniel J. Bernstein. Factoring into coprimes in essentially linear time. *J. Algorithms*, 54:1–30, 2005.

[6] Daniel J. Bernstein, Hendrik W. Lenstra, Jr., and Jonathan Pila. Detecting perfect powers by factoring into coprimes. *Math. Comp.*, 76(257):385–388, January 2007.

[7] Johannes Blömer and Alexander May. A tool kit for finding small roots of bivariate polynomials over the integers. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2005.

[8] Dan Boneh, Gary Durfee, and Nick Howgrave-Graham. Factoring $N = p^r q$ for large $r$. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 1999.

[9] Alin Bostan, Pierrick Gaudry, and Éric Schost. Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator. *SIAM J. Comput.*, 36(6):1777–1806, 2007.

[10] Aaron Chow. *Applications of Fourier coefficients of modular forms*. Phd thesis, University of Toronto, 2015. Available at `https://tspace.library.utoronto.ca/handle/1807/70815`.

[11] Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In Ueli M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996.

[12] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.

[13] Don Coppersmith. Finding small solutions to small degree polynomials. In Joseph H. Silverman, editor, *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*, volume 2146 of *Lecture Notes in Computer Science*, pages 20–31. Springer, 2001.

[14] Don Coppersmith, Nick Howgrave-Graham, and S. V. Nagaraj. Divisors in residue classes, constructively. *Math. Comp.*, 77(261):531–545, 2008.

[15] Jean-Sébastien Coron and Alexander May. Deterministic polynomial-time equivalence of computing the RSA secret key and factoring. *J. Cryptology*, 20(1):39–50, 2007.

[16] Jean-Sébastien Coron. Finding small roots of bivariate integer polynomial equations revisited. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 492–505. Springer, 2004.

[17] Jean-Sébastien Coron. Finding small roots of bivariate integer polynomial equations: A direct approach. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 2007.

[18] Jean-Sébastien Coron, Jean-Charles Faugère, Guénaël Renault, and Rina Zeitoun. Factoring $N = p^r q^s$ for large $r$ and $s$. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, volume 9610 of *Lecture Notes in Computer Science*, pages 448–464. Springer, 2016.

[19] Edgar Costa and David Harvey. Faster deterministic integer factorization. *Math. Comp.*, 83(285):339–345, 2014.

[20] Richard Crandall and Carl Pomerance. *Prime numbers – A Computational Perspective*. Springer Verlag, 2nd edition, 2005.

[21] Paul Erdős. On the sum $\sum_{k=1}^{x} d(f(k))$. *J. London Math. Soc.*, 27:7–15, 1952.

[22] Jean-Charles Faugère, Raphaël Marinier, and Guénaël Renault. Implicit factoring with shared most significant and middle bits. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2010.

[23] Frédéric Grosshans, Thomas Lawson, Benjamin Smith, and François Morain. Factoring Safe Semiprimes with a Single Quantum Query. working paper or preprint, September 2016.

[24] David Harvey. An exponent one-fifth algorithm for deterministic integer factorisation, 2020. Available at `https://arxiv.org/abs/2010.05450`.

[25] David Harvey and Markus Hittmeir. A log-log speedup for exponent one-fifth deterministic integer factorisation, 2021. Available at `https://arxiv.org/abs/2105.11105`.

[26] Markus Hittmeir. A babystep-giantstep method for faster deterministic integer factorization. *Math. Comput.*, 87(314):2915–2935, 2018.

[27] Markus Hittmeir. A time-space tradeoff for Lehman's deterministic integer factorization method, 2020. Available at `https://arxiv.org/abs/2006.16729`.

[28] Nick Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Cryptography and Coding, 6th IMA International Conference, Cirencester, UK, December 17-19, 1997, Proceedings*, pages 131–142, 1997.

[29] Nick Howgrave-Graham. Approximate integer common divisors. In Joseph H. Silverman, editor, *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*, volume 2146 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2001.

[30] Charanjit S. Jutla. On finding small solutions of modular multivariate polynomial equations. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 1998.

[31] Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, 3rd edition, 1997.

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

[33] R. Sherman Lehman. Factoring large integers. *Math. Comp.*, 28:637–646, 1974.

[34] Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and Lászlo Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[35] H. W. Lenstra, Jr. Miller's primality test. *Inform. Process. Lett.*, 8(2):86–88, 1979.

[36] Hendrik W. Lenstra, Jr. Divisors in residue classes. *Math. Comp.*, 42(165):331–340, January 1984.

[37] William J. LeVeque. *Fundamentals of number theory*. Dover, 1996.

[38] Douglas L. Long. Random equivalence of factorization and computation of orders. Technical Report 284, Princeton University, Depatment

of Electrical Engineering and Computer Science, April 1981. Available at http://www.lix.polytechnique.fr/Labo/Francois.Morain/Introuvables/long-orders.pdf.

[39] Florian Luca and Carl Pomerance. On the average number of divisors of the Euler function. *Publ. Math. Debrecen*, 70(1-2):125–148, 2007.

[40] Alexander May and Maike Ritzenhofen. Implicit factoring: On polynomial time factoring given only an implicit hint. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*, volume 5443 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2009.

[41] Gary L. Miller. Riemann's hypothesis and tests for primality. In *Proc. 7th STOC*, pages 234–239, 1975.

[42] Phong Q. Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. Comput.*, 39(3):874–903, 2009.

[43] Andrew Novocin, Damien Stehlé, and Gilles Villard. An lll-reduction algorithm with quasi-linear time complexity: extended abstract. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 403–412. ACM, 2011.

[44] John M. Pollard. Theorems on factorization and primality testing. *Proc. Cambr. Philos. Soc.*, 76:521–528, 1974.

[45] Michael O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1979.

[46] Maike Ritzenhofen. *On efficiently calculating small solutions of systems of polynomial equations: lattice-based methods and applications to cryptography.* PhD thesis, Ruhr University Bochum, 2010.

[47] Ronald L. Rivest and Adi Shamir. Efficient factoring based on partial information. In Franz Pichler, editor, *Advances in Cryptology - EUROCRYPT '85, Workshop on the Theory and Application of of Cryptographic Techniques, Linz, Austria, April 1985, Proceedings*, volume 219 of *Lecture Notes in Computer Science*, pages 31–34. Springer, 1985.

[48] Santanu Sarkar and Subhamoy Maitra. Further results on implicit factoring in polynomial time. *Advances in Mathematics of Communications*, 3(2):205–217, 2009.

[49] Santanu Sarkar and Subhamoy Maitra. Approximate integer common divisor problem relates to implicit factorization. *IEEE Trans. Information Theory*, 57(6):4002–4013, 2011.

[50] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

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

[52] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.).* Cambridge University Press, 2013.

[53] Dieter Wolke. A new proof of a theorem of van der Corput. *J. London Math. Soc. (2)*, 5:609–612, 1972.

[54] Heather Woll. Reductions among number theoretic problems. *Information and Computation*, 72:167–179, 1987.

[55] Bartosz Źrałek. A deterministic version of Pollard's p-1 algorithm. *Math. Comp.*, 79(269):513–533, 2010.

[56] Bartosz Źrałek. An extension of a result about divisors in a residue class and its application to reducing integer factorization to computing Euler's totient. *Math. Comp.*, 88(317):1261–1272, 2019.

## Appendix A. Splitting and factoring integers

In this appendix, we give several algorithms to split an integer (i.e., finding a non-trivial divisor) or to factor it completely, summarising our work.

- `FactorWithEvenPower` (Alg. 7) is a primitive for factoring using an element of even order;
- `SplitWithOracleRandom` (Alg. 8) is a randomized version of factoring with oracles, together with `FactorWithOracleRandom` (Alg. 9). A version for $\Sigma$ can be found in [4].
- `FactorWithOracle` is the main function for factoring using an oracle (Alg. 10), and its ancillary function `FWO` (Alg. 11);
- `FactorWithFactoredOrder` (Alg. 12) uses the factorization order oracle.

**Algorithm 7:** Factoring with an even power

```
Function FactorWithEvenPower($N$, $a$, $k$)
    input  : Integers $(N, a, k)$ with $k$ even, $\gcd(a, N) = 1$, and
             $a^k \equiv 1 \bmod N$
    output : a non-trivial divisor $1 < d < N$ or failure
    Compute $(s, t)$ such that $k = 2^s \cdot t$ with $s > 0$, $t$ odd
    // by hypothesis, $a^{2^s \cdot t} \equiv 1 \bmod N$
    $b \leftarrow a^t \bmod N$
    if $b \neq 1$ then
        find the smallest $s'$, $1 \leq s' \leq s$ such that $b^{2^{s'}} \equiv 1 \bmod N$
        $c \leftarrow b^{2^{s'-1}} \bmod N$        // $c$ is a square root of 1
        if $c \neq -1$ then
            return $\gcd(c - 1, N)$
    return failure
```

If $k = \operatorname{ord}_N(a)$, then surely, we cannot have $b = 1$. When $k = \varphi(N)$ (resp. $\lambda(N)$), the probability that $b = 1$ is bounded by $1/2^s \leq 1/2^{\omega(N)}$. There are $2^{\omega(N)}$ square roots of 1 (by the Chinese Remainder Theorem), including ±1. There are $2^{\omega(N)} - 1$ possible values for $c$ and only one is trivial. So $c \neq -1$ with probability $\geq 1 - 1/2^{\omega(N)-1} \geq 1/2$.

If we use the order oracle $\mathcal{O}$, then we may need to try several random values of $a$ until we find one with even order $r$. And again this happens with probability $\leq 1/2^s$.

(F. Morain and B. Smith) LIX - Laboratoire d'informatique de l'École polytechnique, GRACE - Inria Saclay - Ile de France

*Email address*, F. Morain: morain@lix.polytechnique.fr

*Email address*, B. Smith: smith@lix.polytechnique.fr

**Algorithm 8:** Splitting an integer with an oracle using randomness

```
Function SplitWithOracleRandom($N$, $\varpi$)
    input  : An integer $N$, an oracle $\varpi \in \{\Phi, \Lambda, \mathcal{O}\}$
    output : $N$ if $N$ is prime, otherwise a non-trivial divisor
             $1 < d < N$
    if $\varpi \in \{\Phi, \Lambda\}$ then
        $k \overset{\varpi}{\longleftarrow} \mathcal{F}(\varpi)(N)$
        if $k = N - 1$ then                                   // $N$ is prime
            return $N$
    while true do
        choose a random $a \in [2, N - 2]$
        $g \leftarrow \gcd(a, N)$
        if $g \neq 1$ then
            return $g$
        if $\varpi = \mathcal{O}$ then
            $k \overset{\varpi}{\longleftarrow} \mathrm{ord}_N(a)$
            if $k = N - 1$ then                               // $N$ is prime
                return $N$
        // by hypothesis, $a^k \equiv 1 \bmod N$
        if $k$ is even then
            $res \leftarrow$ FactorWithEvenPower($N$, $a$, $k$)
            if $res \neq$ failure then
                return $res$
```

(G. Renault) Agence Nationale de la Sécurité des Systèmes d'Information, and LIX - Laboratoire d'informatique de l'École polytechnique, CNRS, Institut Polytechnique de Paris *and* GRACE - Inria Saclay–Île-de-France

*Email address*, G. Renault: guenael.renault@ssi.gouv.fr

**Algorithm 9:** Factoring an integer with an oracle and randomness

```
Function FactorWithOracleRandom($N$, $\varpi$)
    input  : Integer $N$, oracle $\varpi \in \{\Phi, \Lambda, \mathcal{O}\}$
    output : A set $\{(p_1, e_1), \dots, (p_r, e_r)\}$ s.t. $N = \prod_{i=1}^{r} p_i^{e_i}$ with all
             $p_i$ prime
    $d \leftarrow$ SplitWithOracleRandom($N$, $\varpi$)
    if $d = N$ then                                        // $N$ is prime
        return $\{(N, 1)\}$
    else
        $\mathcal{L}_0 \leftarrow$ Refine($\{d, N/d\}$)
        // $\mathcal{L}_0 = \{(M_1, e_1), \dots, (M_s, e_s)\}, \gcd(M_i, M_j) = 1$ for $i \neq j$
        $\mathcal{L} \leftarrow \emptyset$
        for $(M, e) \in \mathcal{L}_0$ do
            $\mathcal{L}_1 \leftarrow$ FactorWithOracle($M$, $\varpi$)
            // $\mathcal{L}_1 = \{(p_1, f_1), \dots, (p_u, f_u)\}$, $p_i$ prime
            for $(p, f) \in \mathcal{L}_1$ do    // since all $M_i$ are coprime, $p$
             is not in $\mathcal{L}$
                $\mathcal{L} \leftarrow \mathcal{L} \cup \{(p, ef)\}$
        return $\mathcal{L}$
```

**Algorithm 10:** Factoring an integer with an oracle

```
Function FactorWithOracle($N$, $\varpi$)
    input  : A squarefree integer $N$, an oracle $\varpi \in \{\Phi, \Sigma\}$
    output : Sets $\mathcal{P}$ and $\mathcal{C}$ (possibly empty) containing prime and
             composite divisors of $N$, respectively
    if $\varpi = \Phi$ then
        $sgn \leftarrow -1$
    else
        $sgn \leftarrow +1$
    $M \xleftarrow{\varpi} \mathcal{F}(\varpi)(N)$
    if $M = N + sgn$ then                                  // $N$ is prime
        return $(\{N\}, \emptyset)$
    $\mathcal{P} \leftarrow$ FactorizationWithPhi2($N$)
    if $\mathcal{P} \neq \emptyset$ then
        return $(\mathcal{P}, \emptyset)$
    $g \leftarrow \gcd(N, M)$
    if $g \neq 1$ then
        return FWO($N$, $\{g\}$, $\varpi$)
    $\mathcal{D} \leftarrow$ FactoringWithKnownDifference($N$, $M$, $1$, $0.5$)
    if $\mathcal{D} \neq \emptyset$ then               // we have found some $D > N_1^{1/2}$
        return FWO($N$, $\mathcal{D}$, $\varpi$)
    // Theorem 5.5 can be used?
    $\mathcal{D} \leftarrow$ SplitCF($N$, $\varpi$)
    if $\mathcal{D} \neq \emptyset$ then
        return FWO($N$, $\mathcal{D}$, $\varpi$)
    $\mathcal{D} \leftarrow$ SplitLLL($N$, $\varpi$, $\lfloor \log_2 N \rfloor$)
    if $\mathcal{D} \neq \emptyset$ then
        return FWO($N$, $\mathcal{D}$, $\varpi$)
    return $(\emptyset, \{N\})$
```

**Algorithm 11:** Ancillary function for Algorithm 10

```
Function FWO($N$, $\mathcal{D}$, $\varpi$)
    input  : Squarefree integer $N$, an oracle $\varpi \in \{\Phi, \Sigma\}$, and a set
             $\mathcal{D}$ of non-trivial divisors of $N$
    output : Two sets (possibly empty) $\mathcal{P}$ and $\mathcal{C}$ where the former
             (resp. the latter) contains prime (resp. composite)
             divisors of $N$
    $\mathcal{D} \leftarrow \cup_{d \in \mathcal{D}}\{d, N/d\}$    // force $d$ and $N/d$ to be in the set
    $\mathcal{D} \leftarrow$ CleanDivisors($\mathcal{D}$)
    $(\mathcal{P}, \mathcal{C}) \leftarrow (\emptyset, \emptyset)$
    for $d \in \mathcal{D}$ do
        $(\mathcal{P}_d, \mathcal{C}_d) \leftarrow$ FactorWithOracle($d$, $\varpi$)
        $(\mathcal{P}, \mathcal{C}) \leftarrow (\mathcal{P} \cup \mathcal{P}_d, \mathcal{C} \cup \mathcal{C}_d)$
    return $(\mathcal{P}, \mathcal{C})$
```

**Algorithm 12:** Factoring with factorization of order

```
Function FactorWithFactoredOrder(N, a)
    Input  : $N$, $a$
    Output: A set of pairs $(M_i, e_i)$ with the $M_i$ pairwise coprime
            and $\prod_i M_i^{e_i} = N$
    $\{(\ell_1, e_1), \ldots, (\ell_u, e_u)\} \leftarrow$ Factorization($\mathcal{O}(a, N)$)
    $r \leftarrow \Pi_{i=1}^{u} \ell_i^{e_i}$
    if $r = N - 1$ then                                // $N$ is prime
        return $\{(N, 1)\}$
    $g \leftarrow \gcd(r, N)$
    if $g \neq 1$ then
        $\mathcal{L} \leftarrow \emptyset$
        for $i \leftarrow 1$ to $u$ do
            $v \leftarrow \nu_{\ell_i}(N)$        // maximal power of $\ell_i$ dividing $N$
            if $v > 0$ then
                $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\ell_i, v)\}$
                $N \leftarrow N/\ell_i^v$
        // $N > 1$ since $r < N$
        return $\mathcal{L} \cup$ FactorWithFactoredOrder($N$, $a$)
    $\mathcal{M} \leftarrow \emptyset$
    for $i \leftarrow 1$ to $u$ do
        $b \leftarrow a^{r/\ell_i} \bmod N$
        $g \leftarrow \gcd(b - 1, N)$
        if $1 < g < N$ then
            $\mathcal{M} \leftarrow \mathcal{M} \cup \{g, N/g\}$
    return CleanDivisors($N$, $\mathcal{M}$)
```