Deterministic factoring with oracles

François Morain, Guénaël Renault, Benjamin Smith

DETERMINISTIC FACTORING WITH ORACLES

FRANÇOIS MORAIN, GUÉNAËL RENAULT, AND BENJAMIN SMITH

Abstract. We revisit the problem of integer factorization with number-theoretic oracles, including a well-known problem: can we factor an integer N unconditionally, in deterministic polynomial time, given the value of the 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 for example that if N is a squarefree integer with 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 problems for Carmichael's function, and the order oracle that is used in Shor's quantum factoring algorithm.

1. Introduction

The well-known *fundamental theorem of arithmetic* asserts that every integer N can be written as 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, even deterministically. For example, if N is prime, then this can be proven in deterministic polynomial time in theory [1], and more efficiently (though heuristically) in practice [35]. Prime powers can be detected in linear time [5].

But when N has more than one prime factor, hard work is generally required. In the quantum world, we can apply Shor's algorithm [41]. 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 time $O(\exp((c + o(1))(\log N)^{1/3}(\log\log N)^{2/3}))$ [19]. This complexity explains the success of the RSA cryptosystem, which is based on the supposed difficulty of factoring numbers with only two prime factors.

Deterministic unconditional methods of factoring are rare, and all have exponential running time for general N. The first such method was due to Fermat, followed by Lehman [28]; more recent methods include Bostan–Gaudry–Schost [9] and Costa–Harvey [18], which is currently the fastest deterministic factoring algorithm for general N. Better results exist for numbers with 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 $O(\log p_1)$. This was extended in [17] to numbers $N = p_1^r p_2^s$ with r and/or s in $O((\log p_1)^3)$.

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 the following oracles in particular:

Date: February 23, 2018.

- Φ: on input N returns $\varphi(N)$, the value of the Euler totient function (see §2.1);
- Λ: on input N returns $\lambda(N)$, the value of the Carmichael lambda function (see §2.2);
- $\mathcal{O}$: on input N and a with $\gcd(a, N) = 1$, returns the order of a modulo N (see §2.3).

We study the conditions under which these oracles can be used to factor N deterministically, unconditionally, and in a time complexity better than exponential.

The story of factoring with oracles began with Miller [34] and Long [30], who considered randomized factoring algorithms with access to Φ. Woll [43] explored relationships between number-theoretic problems including factorization and the Φ and $\mathcal{O}$ oracles. Recently, Żralek [44] has shown that iterated calls to Φ allow deterministic factoring in subexponential time, after using Landau's algorithm to reduce to the squarefree case as in §5.1. In a different direction, Bach, Miller, and Shallit [3] showed that an oracle yielding the sum of the divisors of N allows efficient factoring. 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 [38] 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 its 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]). This requires only half of the most or least significant bits of p to be known to factor N with a polynomial time complexity.

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. We obtain 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 $\varphi(N)$ or $\lambda(N)$.*

Proof. See Theorem 5.10. □

Theorem 1.2. *Assume N is squarefree and has exactly three prime factors $p_i = N^{\alpha_i}$, where $\alpha_1 > \alpha_2 > \alpha_3$. Then we can compute a nontrivial factor of N in deterministic polynomial time in* $\log(N)$ *given $\varphi(N)$ or $\lambda(N)$ 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 5.4, 5.6, 5.9, and 5.10. □

We recall the definition of our oracles, and some associated number-theoretic results, in §2. We then state the relevant results of Coppersmith and Howgrave-Graham in §3. These underpin our core results in §4, which solve (generalizations of) the following problem: given N and M such that there exists a (large enough) prime p with $p \mid N$ and $p - 1 \mid M$, recover p in deterministic polynomial time. We apply these algorithms to factoring with Φ and Λ in §5, and with $\mathcal{O}$ in §6.

Remark 1.1. Similar algorithms and results hold given an oracle yielding the value of the sum-of-divisors function $\sigma(N) = \sum_{d|N} d = N \prod_{p|N}(1 + 1/p)$, but we do not pursue these analogues here. We also note that all results involving $p - 1$ can be easily adapted to use $p + 1$ instead.

2. The oracles

As above, we suppose $N = \prod_{i=1}^{k} p_i^{e_i}$, where the p_i are distinct primes and $e_i > 0$. We 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 Φ oracle.** Given N as above, the oracle Φ returns the value of the Euler totient function,

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

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

2.2. **The Λ oracle.** Given N as above, the oracle Λ returns the value $\lambda(N)$ of Carmichael's λ function. This is the exponent of $(\mathbb{Z}/N\mathbb{Z})^\times$, the maximal order of an element modulo N; so

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

2.3. **The $\mathcal{O}$ oracle.** Given N 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 applies the Quantum Fourier Transform to construct a quantum polynomial-time order-finding algorithm, which yields an efficient factorization algorithm after some classical postprocessing (similar to the process in §2.5 below). This order-finding algorithm should not be seen as a true realization of $\mathcal{O}$, since it is only guaranteed to return a *divisor* of $\operatorname{ord}_N(a)$; however, for most inputs it returns the true order with very high probability. Hence, factoring with $\mathcal{O}$ can give us some valuable intuition into Shor-style quantum factoring algorithms.

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

$$\mathcal{O}(N, a) \mid \Lambda(N) \quad \text{and} \quad \Lambda(N) \mid \Phi(N)$$

for all N and all a prime to N.

While the φ and λ functions may seem very close, it is easy to see that $\varphi(N)/\lambda(N)$ can be made quite large. For example, if $N = p_1 p_2$ where $p_1 - 1 = 2(p_2 - 1)$ (so p_2 is a Sophie Germain prime), then $\varphi(N)/\lambda(N) = p_2 - 1 = O(\sqrt{N})$. However, the following easy result will be useful to us.

Lemma 2.1. *If N is odd, then* $\gcd(N, \lambda(N)) = \gcd(N, \varphi(N))$.

Proof. Expanding $\varphi(N)$ and $\lambda(N)$, we have $\gcd(N, \lambda(N)) = \left(\prod_i p_i^{e_i-1}\right) \gcd\left(\prod_i p_i, \prod_j (p_j - 1)\right) = \left(\prod_i p_i^{e_i-1}\right) \gcd\left(\prod_i p_i, \operatorname{lcm}_j (p_j - 1)\right) = \gcd(N, \varphi(N))$. □

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

2.5. **Randomized algorithms.** All three oracles give efficient randomized factoring algorithms. The key is to find some $b \neq \pm 1$ in $\mathbb{Z}/N\mathbb{Z}$ such that $b^2 \equiv 1 \pmod{N}$ (that is, a non-trivial square root of 1); then $\gcd(b-1, N)$ is a nontrivial factor of N. Such b always exist if N is not prime.

To find a nontrivial square root of 1 modulo N using Φ, note that $\varphi(N)$ is even, so we can write $\varphi(N) = 2^s t$ with t odd. Then, from $(a^t)^{2^s} \equiv 1 \pmod{N}$ we can deduce b such that $b^2 \equiv 1 \pmod{N}$. The same relations hold with $\lambda(N)$ in place of $\varphi(N)$ (though generally with different values of s and t), so the same algorithm works with Λ in place of Φ.

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. Then, with probability $\geq 1 - 1/2^{\omega(N)-1}$, the element $b \equiv a^{r/2} \pmod{N}$ will be a non-trivial square root of 1 and therefore yield a nontrivial factor of N.

Remark 2.1. Folklore tells us that there is a randomized polynomial-time reduction between computating square roots modulo N and factoring N. Rabin gives a precise analysis when N is a product of two primes in [36, Theorem 1]. To render this approach deterministic (as in [34]) one needs a bound on non-quadratic residues, but this bound can only be obtained under ERH.

3. Lattices, Coppersmith's method, and approximate GCDs

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 GCDs. We also introduce some elementary subroutines that we will use to improve the quality of our factorizations.

The Lenstra–Lenstra–Lovasz lattice basis reduction algorithm (LLL) is at the heart of both Coppersmith's and Howgrave-Graham's methods. Recall that if L is a lattice of dimension d in $\mathbb{R}^n$ (with the Euclidean norm $\|\cdot\|$), then we say that a basis $\{b_i : 1 \leq i \leq d\}$ of L is *LLL-reduced* if

$$\|b_1\| \leq \|b_2\| \leq \cdots \leq \|b_i\| \leq 2^{\frac{d(d-1)}{4(d+1-i)}} \det(L)^{1/(d+1-i)} .$$

The LLL algorithm computes an LLL-reduced basis for L in polynomial time in d, n, and $\log B$. Moreover, the resulting b_1 is approximately as short as possible: $\|b_1\| \leq 2^{n-1} \min_{v \in L \setminus \{0\}} \|v\|$.

3.1. **Univariate Coppersmith.** Coppersmith's breakthrough algorithms [11, 12] use lattice reduction to find small roots of modular univariate or multivariate integer polynomials. The general principle is to build a lattice of coefficient vectors $(h_0, \ldots, h_n)$ of real polynomials $h(x) = \sum_{i=0}^{n} h_i x^i$ sharing common roots, and then find short vectors in this lattice using LLL. Some cleverness is needed to construct small lattices that yield large bounds on the expected roots. We refer the reader to [33] for a survey on this topic, and [6] for the best complexity result.

Theorem 3.1 states Coppersmith's main result on solving modular univariate equations [11], following Howgrave-Graham's simpler formulation in [24].

Theorem 3.1. *Let N be an integer with an* unknown *factor $D \geq N^\beta$ with $0 < \beta \leq 1$. Let $f(x)$ be a univariate monic polynomial of degree δ. Then we can find all solutions x_0 to the equation*

$$f(x) \equiv 0 \pmod{D}$$

with $|x_0| \leq N^{\beta^2/\delta}$ in polynomial time in δ and $\log(N)$.

3.2. **Bivariate Coppersmith.** Theorem 3.2 describes Coppersmith's method for finding small zeroes of bivariate polynomials. Unlike the univariate result above, these bivariate equations are not modular. Subsequent clarifications of Coppersmith's algorithm appear in [15] and [7], and extensions to the general multivariate case in [25], [14], [7], and [37]. For Theorem 3.2, we refer to Coron's treatment in [16].

Theorem 3.2. *Let $f(x, y) \in \mathbb{Z}[x, y]$ be irreducible, of degree at most δ in each variable, and suppose $f(x_0, y_0) = 0$ for some $|x_0| < X$, $|y_0| < Y$. If*

$$XY < \mathcal{W}^{2/(3\delta)} \quad \text{where} \quad \mathcal{W} = \|f(xX, yY)\| ,$$

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

In this article we will apply the special cases of Theorems 3.1 and 3.2 where the polynomial f is linear in each variable to find divisors of N. In another direction, but using the same techniques, Theorem 3.3 improves on a result of Lenstra [29].

Theorem 3.3 (Coppersmith–Howgrave-Graham–Nagaraj [13])**.** *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.*

Remark 3.1. The results obtained by using Coppersmith's methods are asymptotic: they use lattices of dimensions which tend to $\log(N)$. To make these methods practical generally requires an exhaustive search (see [6] for the best known practical algorithm).

3.3. **Approximate GCDs.** One of the first applications of Coppersmith's theorems 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 the one due to Howgrave-Graham [24], who later used this result to solve the *Approximate GCD* problem [23], formalized in Definition 3.4.

Definition 3.4. Given integers A and B, and bounds X and M for which there exists some $D > M$ and x with $|x| \le X$ such that $D \mid B$ and $D \mid (A + x)$, the PACDP problem is to find all such (D, x).

Howgrave-Graham gives two types of algorithms for solving PACDP instances in [23]. The first, using continued fractions, is described by Proposition 3.5 and Algorithm 1 (PacdCF). The second approach, using LLL, is described by Theorem 3.6 and Algorithm 2 (PacdL).

3.4. **Computing approximate GCDs via continued fractions.**

Proposition 3.5 (Howgrave-Graham [23])**.** *Given integers $A < B$, Algorithm 1 (PacdCF) computes all solutions (D, x) to the PACDP for (A, B) with $(M, X) = (B^\alpha, B^{(2\alpha-1)/2})$ for any $\alpha > 1/2$ in deterministic polynomial time in $\log B$.*

Proof. Suppose (D, x) is one of the desired PACDP solutions: then $D \mid B$ and $D \mid (A + x)$, and $D > B^\alpha$ and $|x| \le B^{(2\alpha-1)/2}$ for some $\alpha > 1/2$. Write $a' = (A + x)/D$ and $b' = B/D$; then $a' < \frac{A+B^{2\alpha-1}}{B^\alpha} < B^{1-\alpha}$ and $b' < B^{1-\alpha}$, 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 a convergent of A/B. Algorithm 1 therefore begins by computing the convergents (this is closely related to the computation of $\gcd(A, B)$, and can be done in deterministic polynomial time [26, 42]). For each convergent g_i/h_i, if $h_i \mid B$ then we recover the PACDP solution $(D, x) = (B/h_i, Dg_i - A)$. We can stop as soon as $h_i > B$, because such h_i cannot divide B. □

Remark 3.2. 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.

Algorithm 1: PacdCF: Approximate GCD using continued fractions.

```
1 Function PacdCF(A, B)
      Input  : A < B
      Output: The set of solutions (D, x) to the PACDP for (A, B) with (M, X) = (B^α, B^{2α−1})
              for some α > 1/2 (so D | (A + x) and D | B).
2     R ← ∅
3     (g_0/h_0, ..., g_n/h_n) ← continued fraction convergents of A/B, stopping when h_{n+1} > B
4     for i = 0 up to n do
5         if h_i | B (and h_i > 1) then
6             (D, x) ← (B/h_i, D g_i − A)
7             R ← R ∪ {(D, x)}
8     return R
```

3.5. **Computing approximate GCDs via lattice reduction.**

Theorem 3.6 (Howgrave-Graham [23])**.** *Given integers $A < B$, and α in $(1/2, 1)$ and β in $(0, \alpha^2)$, Algorithm 2 (*PacdL*) computes all solutions (D, x) to the PACDP for (A, B) with $(M, X) = (B^\alpha, B^\beta)$ in deterministic polynomial time in $\log B$ and $1/\epsilon$ where $\epsilon = \alpha^2 - \beta$.*

Sketch of proof. We use two auxiliary integer parameters $h \geq u$, whose precise values will be determined later. The idea is to build an $(h+1)$-dimensional lattice of polynomials that are multiples of $A + x$ and B, and so have a common root modulo d. Let

$$p_i(Z) := \begin{cases} B^i(A+Z)^{u-i} & \text{for } 0 \leq i < u, \\ Z^{i-u}(A+Z)^u & \text{for } u \leq i \leq h \end{cases}$$

and set $\tilde{p}_i := p_i(XZ)$. Finding a small row vector in the lattice of $\tilde{p}_i$ yields a polynomial with a small root over $\mathbb{Z}$. The parameters $u \leq h$ are chosen such that

$$\beta < \frac{u(2(h+1)\alpha - (u+1))}{h(h+1)} ,$$

in order to optimize the size of the shortest vector found by LLL. This gives an optimal value of h, close to u/α, for which

$$\beta < \alpha^2 - \frac{\alpha(1-\alpha)}{h+1} .$$

It remains to take $(h, u) = (\lceil \alpha(1-\alpha)/\epsilon \rceil - 1, \lceil h\alpha \rceil)$. □

As noted in [23], the *continued fraction* (PacdCF) and *lattice* (PacdL) approaches are subtly different: PacdCF requires only a lower bound on one exponent α, whose precise value does not matter, but PacdL requires some relation between two exponents, α and β (or ϵ). We will encounter this difference in §5.4. Similar phenomena appear in the context of *implicit factorization* (e.g. [32, 39, 20, 40]), but in these cases the two exponents can be handled more easily.

Algorithm 2: PacdL: Approximate GCDs using LLL

Function PacdL*(A, B, α, β)*
- **Input :** $A < B$ and $\alpha, \beta \in (0,1)$, with $\epsilon = \alpha^2 - \beta > 0$
- **Output:** the set of solutions (D, x) to the PACDP for (A, B) with $(M, X) = (B^\alpha, B^\beta)$ (so $D \mid (A+x)$ and $D \mid B$).
- $h \leftarrow \lceil \alpha(1-\alpha)/\epsilon \rceil - 1$
- $u \leftarrow \lceil h\alpha \rceil$
- $L \leftarrow$ 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`
- $P(Z) \leftarrow \sum_{i=0}^{h} v_i (Z/X)^i$
- $\mathcal{X} \leftarrow$ integer roots of $P(Z)$
- $\mathcal{R} \leftarrow \emptyset$
- **for** $x \in \mathcal{X}$ **do**
  - $D \leftarrow \gcd(A+x, B)$
  - **if** $D > 1$ *and* $D < B$ **then**
    - $\mathcal{R} \leftarrow \mathcal{R} \cup \{(D, x)\}$
- **return** $\mathcal{R}$

3.6. **Refining partial factorizations.** PacdCF and PacdL both return nontrivial divisors of N, rather than complete factorizations. We can improve the quality of these partial factorizations using some basic auxiliary algorithms.

- Refine takes integers $M_1, \ldots, M_k$, and returns a sequence 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 $\prod_i M_i = \prod_i N_i^{e_i}$. Refine can be implemented by iterating the rewriting formula
$$M_1 M_2 = (M_1/d)(d^2)(M_2/d) \quad \text{where} \quad d = \gcd(M_1, M_2)\,;$$
references start with [2], and faster algorithms are given in [4, 5].
- 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 $m = \prod_i m_i^{e_i}$. This can be done by applying Refine to $d_1, m/d_1, \ldots, d_k, m/d_k$, which yields $\{(n_1, e_1), \ldots, (n_\ell, e_\ell)\}$ such that $\prod_{i=1}^{\ell} n_i^{e_i} = m^k$. These e_j are all multiples of k, so the result to be returned is $\{(n_1, e_1/k), (n_2, e_2/k), \ldots, (n_\ell, e_\ell/k)\}$.

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 $z = 1$, for factoring with Φ in §5, but we think that these more general results have independent interest.

4.1. **Factoring with known auxiliary and unknown difference.** First, consider the search for divisors D of N such that that $(D-z) \mid M$ where M is *given* and the small $z \neq 0$ is *unknown*. We can compute such D in deterministic polynomial time by solving a univariate modular equation.

Theorem 4.1. *Let N and M be integers, and suppose there exists $D \mid N$ such that $(D-z) \mid M$ for a small* unknown *integer $z \neq 0$. Then we can compute D in deterministic polynomial time if*

$$|z| < N^{\alpha^2 - \beta} \quad \textit{where} \quad D = N^\alpha \quad \textit{and} \quad M/(D-z) = N^\beta\,. \tag{1}$$

Proof. Let $y = M/(D-z)$, so $M = y(D-z) = yD - yz$; computing the product yz leads to the divisor D by computing $\gcd(N, M+yz)$. But $x = yz$ is the solution of the modular equation $M + x \equiv 0 \pmod{D}$, and thus (D, x) is a solution to PACDP. By Theorem 3.1, we can compute x in polynomial time if $|x| \leq N^{\alpha^2}$, which is equivalent to $|z| < N^{\alpha^2-\beta}$. □

Corollary 4.2. *Using the notation of Theorem 4.1: if $M \sim N$ and $\log(|z_0|)/\log(N) \sim 0$ then we can recover D in deterministic polynomial time provided $\alpha > (-1+\sqrt{5})/2$.*

Proof. The hypothesis implies $\beta = 1-\alpha$; hence $\alpha^2 + \alpha - 1 > 0$, and the result follows. □

4.2. **Factoring with known difference and unknown auxiliary.** Now we consider the opposite case: finding $D \mid N$ such that $(D-z) \mid M$ where z is *known* and M is *unknown*. In our applications, for example, $z = 1$. In full generality, provided z is especially small, say $\log(|z|)/\log(N) \approx 0$, we can use Coppersmith's bivariate method from Theorem 3.2 to obtain the following results.

Theorem 4.3. *Let N and u be integers with $u \neq 0$ and $\log(|u|)/\log(N) \sim 0$, and suppose there exists $D \mid N$ such that $(D-u) \mid M$ for some integer $M = N^\theta$ with $0 \leq \theta < 1$. Then we can compute D in deterministic polynomial time if*

$$D = N^\alpha \quad \textit{with} \quad \alpha > \frac{1}{4}(1+\theta). \tag{2}$$

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 $x_0 \sim N^{1-\alpha}$ and $y_0 \sim N^{\theta-\alpha}$ are both small, and we can use Theorem 3.2 in a deterministic way. First, as in [15], 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 meets the conditions of Theorem 3.2 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 $\mathcal{W} = \max(N^{1+\theta-\alpha}, N^{2\theta-\alpha}, N^{1+\theta-2\alpha}) = N^{1+\theta-\alpha}$. 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. □

We use Corollary 4.4 in §5.4 to show that *unbalanced* numbers (having a large prime factor) are easy to factor with Φ. In contrast, *compact* N (with all prime factors $\leq N^{1/2}$) are harder to factor.

Corollary 4.4. *Using the notation of Theorem 4.3: if $M \sim N$, then we can recover D in deterministic polynomial time provided $D > N^{1/2}$.*

Remark 4.1. A weaker but simpler result can be obtained using Coron's algorithm, as in [15, §2]: if we use $f^*(x,y) = N - (M+u)x + Ny - uxy$, then $\alpha > (1+\theta)/3$ is enough to recover D.

5. Factoring with the Φ and Λ oracles

We now return to factoring with oracles. We treat the closely-related problems of factoring with Φ and Λ simultaneously here, before treating $\mathcal{O}$ in §6. We consider odd N, since detecting and removing powers of 2 is easy. Ordering the prime divisors of N by decreasing size, we write

$$N = \prod_{i=1}^{k} p_i^{e_i} \quad \text{with primes} \quad p_1 > p_2 > \cdots > p_k > 2\,.$$

The two arithmetical functions we are interested in are

$$\varphi(N) = \prod_{i=1}^{k} p_i^{e_i-1}(p_i-1) \qquad \text{and} \qquad \lambda(N) = \operatorname{lcm}_{i=1}^{k} p_i^{e_i-1}(p_i-1)\,.$$

5.1. **Reduction to the squarefree case.** We begin by reducing to the case of squarefree N: that is, N with $e_1 = \cdots = e_k = 1$. We do this using Landau's algorithm (Algorithm 3), which factors an integer into a product of pairwise coprime perfect powers using either Φ or Λ.

Algorithm 3: Landau's algorithm

```
Function SquareFree(N,ϖ)
    Input  : N an integer ≥ 1, and an oracle ϖ
    Output: (N_1, N_2, ..., N_r) such that N = N_1 · N_2^2 · ··· · N_r^r with each N_i = 1 or squarefree,
            and all of the N_i pairwise coprime
    φ ←ϖ— ϖ(N)
    g ← gcd(N, φ)
    if g = 1 then
        return (N)
    s ← ⌊log N⌋
    (v_1, v_2, ..., v_s) ← SquareFree(g, ϖ)
    R ← N/g
    for i ← 1 to s do
        t ← gcd(R, v_i)
        if t > 1 then
            (v_i, v_{i+1}) ← (v_i/t, v_{i+1} t)
            R ← R/t
    v_1 ← v_1 R
    return (v_1, ..., v_s)
```

Theorem 5.1 (Landau [27])**.** *Given N and $\varpi = \Phi$ or Λ, Algorithm 3 (*SquareFree*) returns $(N_1, \ldots, N_r)$ 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 ϖ.*

Proof. Let $g := \gcd(N, \varpi(N))$. Lemma 2.1 shows that if ϖ is either Φ or Λ, then

$$g = \Big(\prod_{i=1}^{k} p_i^{e_i-1}\Big) \gcd\Big(\prod_{i=1}^{k} p_i\,, \prod_{j=1}^{k}(p_j-1)\Big) = \Big(\prod_{i=1}^{k} p_i^{e_i-1}\Big)\Big(\prod_{i=1}^{k} p_i^{\varepsilon_i}\Big)\,,$$

where each ε_i is 1 or 0 according to whether or not $p_i \mid p_j - 1$ for some j.

If $g = 1$ then N is already squarefree, and we are done. Otherwise, let $R = N/g = \prod_{i=1}^{k} p_i^{1-\varepsilon_i}$. We observe that $p_1 \mid R$, since p_1 divides N but cannot divide g, because it is too large to divide any $p_i - 1$ for $i > 1$. If p is a prime dividing both g and R, then $\nu_p(N) = \nu_p(g) + 1$, and there is a unique i such that $p \mid v_i$: we should replace v_i by v_i/p and v_{i+1} by $v_{i+1}p$. This is dealt with in the loop at Line 9. The primes p dividing R but not g should be put with v_1, as in Line 14. □

5.2. **Reduction to the case** $\gcd(N, \varphi(N))) = 1$**.** Suppose N is squarefree. If $\gcd(N, \varphi(N)) > 1$ (resp. $\gcd(N, \lambda(N)) > 1$), then obviously we learn a nontrivial factor of N; but further, we learn that some p_j also divides at least one of the $p_i - 1$. As a consequence, we get a factorization of N that can be continued with the number of prime factors decreasing on each cofactor. Thus, we reduce to the problem of factoring squarefree N where $\gcd(N, \varphi(N)) = 1$ (resp. $\gcd(N, \lambda(N)) = 1$).

5.3. **Products of two primes.** It is well-known that we can factor a product N of two distinct primes 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 $M = \varphi(N)$.

Lemma 5.2. *If N is a product of two distinct primes and $\varphi(N)$ is known, then the two primes are*

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

Proof. If $N = p_1 p_2$ 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 $M = \varphi(N)$

```
1 Function FactorizationWithPhi2
    Input  : N and M = φ(N), where N is squarefree
    Output: (p1, p2) if N is the product of two distinct primes, or ∅
2   s ← N + 1 − M
3   Δ ← s^2 − 4N                          // Δ = discriminant of X^2 − sX + N
4   if Δ is not square then
5     return ∅
6   p1 ← (1/2)(s + √Δ)
7   p2 ← N/p1
8   return (p1, p2)
```

To convert Algorithm 4 into an algorithm that takes $\lambda(N)$ 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 Φ can be immediately transformed into an algorithm making the same number of calls to Λ. 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 is a product of two distinct primes, then $\varphi(N) = \lambda(N) \cdot \gcd(N - 1, \lambda(N))$.*

Proof. Suppose $N = p_1 p_2$. 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 $\gcd(gq_1q_2 + q_1 + q_2, q_1q_2) = 1$. □

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

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

Theorems 5.4 and 5.6 show how we can factor N by solving PACDP instances if the p_i satisfy certain relative size conditions. To make this precise, 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 $\alpha_k < 1/k$.

The results of §4 yield conditions on the α_i under which factors of N can be computed with the algorithms of §3. As a first step, Theorem 5.4 and Corollary 5.5 give conditions for efficient factoring using Algorithm 5 (SplitCF), which applies PacdCF using Φ or Λ.

Theorem 5.4. *Suppose $\omega(N) \geq 3$ and there exists $1 \leq r < \omega(N)$ such that*

$$\alpha_r \geq 2 \sum_{i=r+1}^{\omega(N)} \alpha_i\,. \tag{3}$$

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

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 (D, x) is a solution to the PACDP for $(A, B) = (\varphi(N), N)$ with $(M, X) = (N^{\alpha}, N^{2\alpha-1})$, and PacdCF will find (D, x) because $\alpha > 1/2$. In this case $x \approx N/D = N^{1-\alpha}$, and the condition becomes $1 - \alpha_r \leq 2\alpha - 1$, which yields Inequality (3). □

Corollary 5.5. *If $\alpha_1 > 2/3$, then Algorithm 5 (*SplitCF*) recovers p_1 in deterministic polynomial time after a single call to Φ or Λ.*

Proof. This is just the special case $r = 1$ of Theorem 5.4 (for which $\alpha_1 > 2/3$). □

Algorithm 5: Splitting an integer using PacdCF

```
Function SplitCF(N, ϖ)
    Input  : N to be factored using oracle ϖ
    Output: ∅ or a set of pairs (M_i, e_i), with the M_i pairwise coprime and N = ∏_i M_i^{e_i}
    M ←ϖ— ϖ(N)
    D ← PacdCF(M, N)          // D = {(D_1, x_1), ..., (D_ℓ, x_ℓ)} with each D_i | N
    if D = ∅ then
        return ∅
    return CleanDivisors(N, {D_1, N/D_1, ..., D_ℓ, N/D_ℓ})
```

We can go further using PacdL instead of PacdCF. Theorem 5.6 is the corresponding analogue of Theorem 5.4.

Theorem 5.6. *If there exist α in $(1/2, 1)$ and β in $(0, \alpha^2)$ such that $\alpha \leq \sum_{i=1}^{r} \alpha_i$ and $1 - \alpha_r \leq \beta$ for some $0 < r < \omega(N)$, 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 α, β, and a single call to Φ or Λ.*

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

Theorem 5.6 is difficult to apply directly, because of the subtlety alluded to in §3.5: it is not enough to simply know that the parameters α and β satisfying the bounds *exist*, because we need to use them as parameters to PacdL. On the other hand, PacdL 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 α_r implying a lower bound for α and an upper bound for β that allow us to apply PacdL. While the bounds may be far from the optimal values of α and β, thus yielding suboptimal performance for PacdL, the solution is still polynomial time, and it allows us to factor some integers that PacdCF 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

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

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 α and β 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 $X \leq \alpha_r$ 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 α and β. Better values may exist, and in any case Theorem 5.10 below will improve the bound for the case $r = 1$ from $\alpha_1 > \overline{\alpha}_1$ to $\alpha_1 > 1/2$ using Coppersmith's method. But in the meantime, we can use Lemma 5.8 to turn the proof of Theorem 5.6 into an effective algorithm.

Theorem 5.9. *Fix an integer $R > 1$. If there exists any $0 < r < \min(R+1, \omega(N))$ for which $\alpha_r \geq \overline{\alpha}_r$, then Algorithm 6 (SplitLLL) recovers the divisor $D = p_1 \cdots p_r = N^{\alpha}$ of N in deterministic polynomial time after a single call to Φ or Λ.*

Proof. Algorithm 6 tries to factor N by calling PacdL 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 α and β suggested by Lemma 5.8. The result therefore follows from r serial applications of Theorem 5.6. □

For $r = 1$, we get a much better lower bound on α_1 from Corollary 4.4. This gives us a result already stated (in a simple form) as Theorem 1.1.

Theorem 5.10. *Assume $\omega(N) \geq 3$ and $\alpha_1 > 1/2$. Then we can recover the divisor $D = p_1$ of N in deterministic polynomial time in $\log(N)$ given $\varphi(N)$ or $\lambda(N)$.*

Proof. We have $\varphi(N) \sim N$, and $(D - 1) \mid \varphi(N)$; the result follows directly from Corollary 4.4. □

When using $\lambda(N)$ instead of $\varphi(N)$ in Theorem 5.10, we can recover p_1 in deterministic polynomial time provided $\alpha_1 > (1+\theta)/4$, where $\theta = \log \lambda(N) / \log N$. When θ is significantly smaller than 1, this gives us a substantially lower bound on α_1. However, finding a condition analogous to Inequality (3) is not so easy for $\lambda(N)$.

Algorithm 6: Splitting an integer using PacdL

Function SplitLLL(N, ϖ)
  Input : N to be factored using oracle ϖ, 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 $N = \prod_i M_i^{e_i}$
  $M \xleftarrow{\varpi} \varpi(n)$
  $\mathcal{D} \leftarrow \emptyset$
  for $r = 1$ **to** R **do**
    $\overline{\alpha}_r \leftarrow (-1 + \sqrt{1 + 4r^2})/(2r^2)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \text{PacdL}(M, N, r\overline{\alpha}_r, 1 - \overline{\alpha}_r)$ // use $(\alpha, \beta) = (r\overline{\alpha}_r, 1 - \overline{\alpha}_r)$
  // $\mathcal{D} = \{(D_1, x_1), \ldots, (D_\ell, x_\ell)\}$ with each $D_i \mid N$
  if $\mathcal{D} = \emptyset$ **then**
    return $\emptyset$
  return CleanDivisors$(N, \{D_1, N/D_1, \ldots, D_\ell, N/D_\ell\})$

5.5. **Products of exactly three primes.** We can say a little more for the special case of squarefree N with $\omega(N) = 3$: that is,

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

As usual, we set $\alpha_i = \log_N p_i$; by definition, $1 > \alpha_1 > \alpha_2 > \alpha_3 > 0$, and α_3 is completely determined by (α_1, α_2) because $\alpha_1 + \alpha_2 + \alpha_3 = 1$. Lemma 5.11 defines the polygon in the (α_1, α_2)-plane corresponding to the domain of validity of the exponents for $\omega(N) = 3$.

Lemma 5.11. *With N and $\alpha_i = \log_N p_i$ defined as above, (α_1, α_2) lies in the region of the (α_1, α_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 α_i. For the last, if $2\alpha_1 + 3\alpha_2 \le 3/2$ then $\alpha_2 \le (3/2 - 2\alpha_1)/3$, whence $1 - \alpha_1 = \alpha_2 + \alpha_3 < 2\alpha_2 \le 2/3(3/2 - 2\alpha_1)$, so $\alpha_1/3 < 0$, which is impossible. □

Figure 1 depicts the values of (α_1, α_2) that our methods can tackle, shading in various regions of the polygon of Lemma 5.11. 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 with

- Theorem 5.4 with $r = 1$ when $\alpha_1 > 2/3$, so (α_1, α_2) is in the diagonally shaded polygon;
- Theorem 5.4 with $r = 2$ when $\alpha_2 \ge 2\alpha_3$, which translates as $2\alpha_1 + 3\alpha_2 \ge 2$, so (α_1, α_2) is in the horizontally shaded polygon, with vertices $(2/5, 2/5)$, $(1/2, 1/2)$, $(2/3, 2/9)$, $(2/3, 1/3)$;
- Theorems 5.6 and 5.9 with $r = 2$ when $\alpha_2 > \overline{\alpha}_2$, so (α_1, α_2) is in the triangle with vertices $(\overline{\alpha}_2, \overline{\alpha}_2)$, $(2/5, 2/5)$, $(1 - 3\overline{\alpha}_2/2 = 0.415, \overline{\alpha}_2)$ (which is too small to be easily seen in Figure 1);
- Theorems 5.6 and 5.9 when (α_1, α_2) is in the dotted polygon;
- Theorem 5.10 when (α_1, α_2) is in the crosshatched polygon.

The grey polygon corresponds to the zone where we cannot prove deterministic polynomial-time factorization.

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

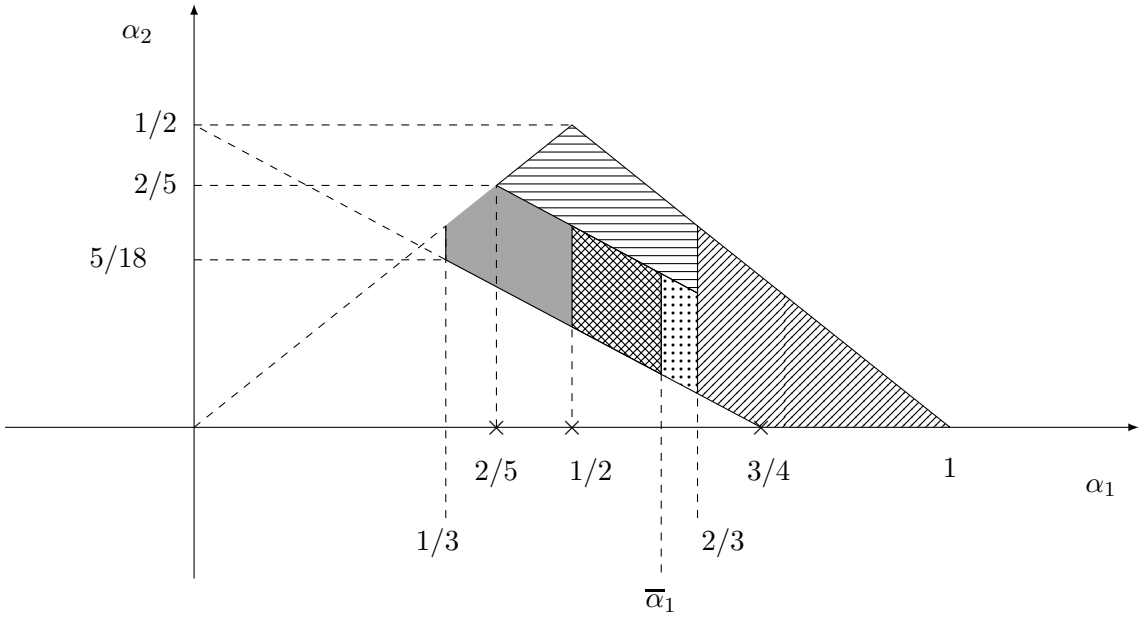

FIGURE 1. Cases covered by our results when $\omega(N) = 3$.

Example 5.6.1. Consider an attempt to factor

$$N = 14300000000000000000000000045617$$

using SPLITCF. The oracle Φ tells us that

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

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

$$D = 10000000000000000000000000319\,,$$

which turns out to be prime; the cofactor is $143 = 13 \cdot 11$. In this case, $\alpha_1 = 0.93082\ldots > 2/3$.

Example 5.6.2. Let us factor

$$N = 14300000027170000072930000138567$$

with SPLITCF. The oracle Φ gives

$$\varphi(N) = 12000000021600000060000000108000\,,$$

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

$$D = 10000000019000000510000000969$$

with two prime factors, and the cofactor $N/D = 143$. Here Inequality (3) is satisfied for $r = 2$.

Example 5.6.3. We apply SPLITCF to

$$N = 14300000027170000072930000138567\,,$$

for which the oracle Λ tells us that

$$\lambda(N) = 10000000018000000000500000000900\,.$$

PacdCF reveals a divisor

$$D = 1000000001900000000510000000969$$

with two factors, 10000000019 and 10000000000000000051 (which we find recursively by computing putative φ values), and a cofactor $N/D = 143 = 11 \cdot 13$. We see that $\lambda(N)/\varphi(N) = 1/120$, and

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

Example 5.6.4. Next, consider an attempt to factor

$$N = 268277631293314788242834971321928533335696453431560393354090095217359233\,,$$

using SplitLLL. The oracle Φ tells us that

$$\varphi(N) = 268274948536427486010385526536308497574852756752201586122353237944164160\,.$$

Trying $r = 2$, and calling PacdL with $(A, B) = (\varphi(N), N)$ and $(\alpha, \beta) = (2\overline{\alpha}_2, 1 - \overline{\alpha}_2)$ (which implies lattice parameters $(h, u) = (24, 19)$), we find a solution

$$\begin{aligned} D &= 610540229658532834519888426420070208770724882201228981991\,,\\ x &= 235762656332817397602115116758925944240446 80\,. \end{aligned}$$

In this case, D and N/D have two prime factors each (easily discovered using $\varphi(D)$ and $\varphi(N/D)$):

$$\begin{aligned} D &= 16378937069540641432773673229 \cdot 372759372031494016617249061 79\,,\\ N/D &= 100003 \cdot 4393970621\,. \end{aligned}$$

The exponent vector is

$$(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (0.400, 0.395, 0.135, 0.070)\,,$$

for which $\alpha_1 + \alpha_2 \geq 2\alpha_2 > \sqrt{1 - \alpha_2} \approx 0.7778$.

5.7. **Using the factorization of $\varphi(N)$ or $\lambda(N)$.** Every prime factor $p \mid N$ is necessarily of the form $\delta + 1$ for some even $\delta \mid \varphi(N)$, so we can compute all of the p from the factors of $\varphi(N)$. A general $N = \prod_{i=1}^{k} p_i^{e_i}$ has $d(N) = \prod_{i=1}^{k}(1 + e_i)$ divisors. Though $d(N)$ is on the order of $\log N$ on average, we know that $\overline{\lim} \log d(N)/(\log N/\log\log N) = \log 2$ (this being effective, see [22]). For most N, then, $d(\varphi(N))$ is polynomial in $O(\log N)$; but there can be exceptional N, and it is shown in [31] that $d(\varphi(N))$ tends to be relatively large. For example, $N = 1088641 \cdot 39916801 \cdot 958003201$ has $\omega(N) = 3$, but $\varphi(N) = 2^{26} \cdot 3^{14} \cdot 5^5 \cdot 7^3 \cdot 11^2$, a number with 29160/2 even divisors.

6. Factoring with the order oracle

We now consider factoring using the order oracle $\mathcal{O}$. Our starting point is the efficient algorithm for factoring products of two strong primes which is motivated, proved, and further discussed in [21]. We then consider extensions of this approach to more general N.

6.1. **Strong RSA keys are easy to factor.** Consider the case $N = p_1p_2$; for example, N might be an RSA modulus. One historically important subclass of RSA moduli is formed by products of *strong primes*, which are primes p such that $(p-1)/2$ is also prime.

Proposition 6.1. *Let $N = p_1p_2$ be the product of two strong primes, so $p_1 = 2q_1+1$ and $p_2 = 2q_2+1$ where p_1, p_2, q_1, and q_2 are distinct odd primes. Then we can efficiently and deterministically factor N with precisely **one** call to $\mathcal{O}$.*

Proof. Observe that $\lambda(N) = 2q_1q_2$ has very few divisors: 1, 2, q_1, q_2, q_1q_2, $2q_1$, $2q_2$, and $2q_1q_2$. Let $a = 2$, and $r = \mathcal{O}(a, N)$. Now r cannot be 1 (because $a \neq 1$) or 2 (because $N > 5$); so, replacing r with $r/2$ if r is even, we can assume that r is either one of the q_i, or else q_1q_2. We can distinguish between these cases by primality testing: if r is prime, then we can return $p_1 = 2r + 1$ and $p_2 = N/p_1$. Otherwise, $r = q_1q_2$, and we can use $\varphi(N) = 2\lambda(N) = 4q_1q_2$ to recover the factors using FactorizationWithPhi2 (Algorithm 4). □

6.2. **Can we factor general numbers with $\omega(N) = 2$?** Proposition 6.1 exploits the fact that $\lambda(N)$ has very few divisors when $N = p_1p_2$ is a product of two strong primes. We would like to extend Proposition 6.1 to factor a larger class of numbers with two prime factors without requiring their prime factors to be strong. It is doubtful that we can find an algorithm for all such N's, in particular if $\gcd(p_1 - 1, p_2 - 1)$ is large and factors mix in the order modulo N, see Example 6.2.1 below. However, when the order is large, we can conclude.

Proposition 6.2. *Let $N = p_1p_2$ be a product of two primes, and suppose r is the order of an element mod N. If $r > p_1 + p_2$, then we can efficiently factor N.*

Proof. Since $r \mid \varphi(N)$, we deduce that $p_1+p_2 \equiv (N+1) \pmod r$. If $p_1+p_2 < r$, then this yields the exact value of p_1+p_2, and then we recover p_1 and p_2 with FactorizationWithPhi2$(N, \varphi(N))$. □

Example 6.2.1. Let us give an example of the techniques we can use in the special case where $p_1 = 2q_1 + 1$ and $p_2 = 2q_2q_2' + 1$, with p_1, p_2, q_1, q_2, and q_2' all prime and all distinct (equality cases are left to the reader); then $r \mid 2q_1q_2q_2'$, and not yet covered cases are

$$r \in \{2, (2)q_1, (2)q_2, (2)q_2', (2)q_1q_2, (2)q_1q_2', (2)q_2q_2'\}.$$

The case $r = 2$ is uninteresting, since we can certainly not use $a = -1 \pmod N$. The cases $r = q_1$ and $r = 2q_1$ are easy, since $p_1 - 1 = (2)r$ and then $\gcd(r + 1, N)$ or $\gcd(2r + 1, N)$ yield p_1. This covers the case $r = (2)q_2q_2'$ in a symmetric way.

For $r = q_2$, we get $a^{q_2} \equiv 1 \pmod N$, and $a^{q_2} \equiv 1 \pmod p)_1$, from which $a \equiv 1 \pmod p)_1$ and $\gcd(a - 1, N)$ recovers p_1. For $r = 2q_2$, either $\gcd(a^{q_2} - 1, N)$ reveals p_2 or we are back to the preceding case with $b^{q_2} \equiv 1 \pmod N$. The same results hold for q_2' replacing q_2.

For $r = (2)q_1q_2$ (resp. $(2)q_1q_2'$), no easy conclusions can be drawn. We can look for p_1 such that $p_1 \mid N$ and $p_1 - 1 \mid r$. Letting $p_i = N^{\alpha_i}$ (with $\alpha_1 + \alpha_2 = 1$) and $r = N^\theta$, we can conclude with Theorem 4.3 if $4\alpha_1 > 1 + \theta$, which is certainly true when $\alpha_1 > 1/2 > \alpha_2$. We are left with the case $\alpha_2 > 1/2 > \alpha_1$. First, note that $p_1 < 2r$. If $q_2' \leq q_1$, then $p_2 - 1 \leq 2r$ which yields $p_1 + p_2 \leq 4r$, so that we have a few possible values for $p_1 + p_2$, from which p_1 and p_2 are recovered as usual. Otherwise, if $q_2' > q_1$, write $q_2 = N^{\beta_2}$ and $q_2' = N^{\beta_2'}$ with $\beta_2 + \beta_2' = \alpha_2$. Then $\theta = \alpha_1 + \beta_2 = 1 - \beta_2'$ and the condition of Theorem 4.3 becomes $\alpha_1 > 1/2 - \beta_2'/4$. When $\beta_2 = \beta_2' = 1/4$, we get $4\alpha_1 > 2 - 1/4$ or $\alpha_1 > 7/16$. Note that when β_2' goes from 0 to α_2, the inequality goes from $\alpha_1 > 1/2$ to $\alpha_1 > 1/2 - \alpha_2/4$, implying that α_2 can be as large as $2/3$ (and α_1 as small as $1/3$).

Alternatively, we can look for q_2 such that $q_2 \mid r$ and $q_2 \mid N - p_1$. Take $a_0 = N \bmod r$ and $b_0 = r$; we look for $d = q_2$ and $x_0 = -p_1$ such that $d \mid a_0 + x_0$ and $d \mid b_0$. In the notation of §3.3, we need $q_2 = r^\alpha$ with $\alpha > 1/2$ and $p_1 = r^{2\alpha-1}$.

We cannot go further towards completely solving even this apparently simple case. For example, consider $N = 1469 = 13 \cdot 113$, for which 144 values of a have order $r = 2^2 \cdot 3 \cdot 7 = 84$. In this case $(p_1 - 1) \mid r$, but this p_1 is too small to be recovered with any of the methods given above.

6.3. **The remaining cases.** If N is not squarefree, then we might have $\gcd(r, N) > 1$, and thus a factor of N. In general, the reason why we succeeded in the preceding cases is that the possible orders were few and also non-interleaving. But when $\lambda(N)$ has many divisors, r can take many values, including small ones. Therefore, one base a and its order r will generally not be enough to split N, let alone completely factor it.

6.4. **Factoring when the factorization of the order is known.** As in §5.7, 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 7 shows a straightforward way of making 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 5). Algorithm 7 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 3.3.

Algorithm 7: Factoring with factorization of order

```
1  Function FactorWithFactoredOrder(N, a)
       Input  : N, a
       Output: A set of pairs (M_i, e_i) with the M_i pairwise coprime and ∏_i M_i^{e_i} = N
2      {(ℓ_1, e_1), ..., (ℓ_u, e_u)} ← Factorization(O(a, N))
3      r ← Π_{i=1}^{u} ℓ_i^{e_i}
4      g ← gcd(r, N)
5      if g ≠ 1 then
6          L ← ∅
7          for i ← 1 to u do
8              v ← ν_{ℓ_i}(N)                          // maximal power of ℓ_i dividing N
9              if v > 0 then
10                 L ← L ∪ {(ℓ_i, v)}
11                 N ← N/ℓ_i^v
12         if N = 1 then
13             return L
14         return L ∪ FactorWithFactoredOrder(N, a);
15     M ← ∅
16     for i ← 1 to u do
17         b ← a^{r/ℓ_i} mod N
18         g ← gcd(b − 1, N)
19         if 1 < g < N then
20             M ← M ∪ {(g, N/g)}
21     return CleanDivisors(N, M)
```

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. Even with more information, such as the complete factorizations of oracle values, we *still* cannot factor all N. This may be surprising, but it shows the fundamental difficulty of factoring.

Acknowledgements. We thank J. Shallit for sending us a copy of [30] and W. George for bringing [10] to our attention. All algorithms were programmed and tested in Magma, and some computations were done in Maple.

References

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Ann. of Math. (2)*, 160(2):781–793, 2004.
[2] E. Bach, J. Driscoll, and J. Shallit. Factor refinement. *J. Algorithms*, 15:199–222, 1993.
[3] E. Bach, G. L. Miller, and J. Shallit. Sums of divisors, perfect numbers and factoring. *SIAM J. Comput.*, 15(4):1143–1154, 1986.
[4] D. Bernstein. Factoring into coprimes in essentially linear time. *J. Algorithms*, 54:1–30, 2005.
[5] D. J. Bernstein, H. W. Lenstra, Jr., and J. Pila. Detecting perfect powers by factoring into coprimes. *Math. Comp.*, 76(257):385–388, January 2007.
[6] Jingguo Bi, Jean-Sébastien Coron, Jean-Charles Faugère, Phong Q. Nguyen, Guénaël Renault, and Rina Zeitoun. Rounding and chaining LLL: finding faster small roots of univariate polynomial congruences. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 185–202, 2014.
[7] J. Blömer and A. 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] D. Boneh, G. Durfee, and N. 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] A. 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] D. Coppersmith. Finding a small root of a univariate modular equation. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *Lecture Notes in Comput. Sci.*, pages 155–165. Springer-Verlag, 1996. International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 1996.
[12] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.
[13] D. Coppersmith, N. Howgrave-Graham, and S. V. Nagaraj. Divisors in residue classes, constructively. *Math. Comput.*, 77(261):531–545, 2008.
[14] 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.
[15] J.-S. 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.

[16] J.-S. 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.

[17] J.-S. Coron, J.-C. Faugère, G. Renault, and R. 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.

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

[19] R. Crandall and C. Pomerance. *Prime numbers – A Computational Perspective*. Springer Verlag, 2nd edition, 2005.

[20] J.-C. Faugère, R. Marinier, and G. 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.

[21] F. Grosshans, T. Lawson, B. Smith, and F. Morain. Factoring Safe Semiprimes with a Single Quantum Query. working paper or preprint, September 2016.

[22] G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers*. Clarendon Press, 5th edition, 1985.

[23] N. 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.

[24] 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.

[25] 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.

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

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

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

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

[30] D. Long. Random equivalence of factorization and computation of orders. Technical Report 284, Princeton University, Depatment of Electrical Engineering and Computer Science, April 1981.

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

[32] A. May and M. 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.

[33] Alexander May. Using LLL-reduction for solving RSA and factorization problems. In *The LLL Algorithm*, Information Security and Cryptography, pages 315–348. Springer, 2010.

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

[35] F. Morain. Implementing the asymptotically fast version of the elliptic curve primality proving algorithm. *Math. Comp.*, 76:493–505, 2007.

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

[37] 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.

[38] 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.

[39] S. Sarkar and S. Maitra. Further results on implicit factoring in polynomial time. *Advances in Mathematics of Communications*, 3(2):205–217, 2009.
[40] Santanu Sarkar and Subhamoy Maitra. Approximate integer common divisor problem relates to implicit factorization. *IEEE Trans. Information Theory*, 57(6):4002–4013, 2011.
[41] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.
[42] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013.
[43] H. Woll. Reductions among number theoretic problems. *Information and Computation*, 72:167–179, 1987.
[44] B. Żralek. A deterministic version of Pollard's p-1 algorithm. *Math. Comput.*, 79(269):513–533, 2010.

(F. Morain and B. Smith) LIX - Laboratoire d'informatique de l'École polytechnique, GRACE - Inria Saclay - Ile de France
E-mail address, F. Morain: `morain@lix.polytechnique.fr`
E-mail address, B. Smith: `smith@lix.polytechnique.fr`

(G. Renault) Agence Nationale de la Sécurité des Systèmes d'Information, POLSYS - INRIA Paris - UPMC - LIP6
E-mail address, G. Renault: `guenael.renault@ssi.gouv.fr`