Intractable Problems in Number Theory

Eric Bach
Computer Sciences Department
University of Wisconsin
Madison, WI 53706

Abstract. This paper surveys computational problems related to integer factorization and the calculation of discrete logarithms in various groups. Its aim is to provide theory sufficient for the derivation of heuristic running time estimates, and at the same time introduce algorithms of practical value.

0. Introduction

Several problems in number theory are believed to be computationally intractable, a property that is potentially of great use in cryptography. Included in this category are problems related to integer factorization and the evaluation of discrete logarithms in various groups. The purpose of this paper is to summarize current knowledge about them, from a theoretical viewpoint.

In line with the long-term goals of complexity theory we should like to settle the question of whether these problems are really difficult, in the sense of having no probabilistic polynomial time algorithms. However, two features of this program seem inappropriate to the present context. First, a concentration on the asymptotic behavior of algorithms may be too restrictive, as a designer of public-key cryptosystems has to make compromises between efficiency and security and so must consider problems of a fixed size. Second, a restriction to algorithms that can be rigorously analyzed is too stringent if one wishes to design a system that will resist all known attacks. Since currently we cannot even prove asymptotic lower bounds on the complexity of these problems, design decisions must be based on what we believe to be the best algorithms. Such has been the state of affairs ever since the invention of public-key cryptology; it seems unlikely to change soon.

Preparation of this paper was supported by the National Science Foundation, via grants DCR-8504485 and DCR-8552596.

S. Goldwasser (Ed.): Advances in Cryptology - CRYPTO '88, LNCS 403, pp. 77-93, 1990.

Of course, there *have* been improvements in our ability to solve these problems, most strikingly for factorization. A paper written in the early 1980's [Pomerance 1982] noted that the available algorithms could factor numbers up to 50 digits; the record now stands at 100 digits [Lenstra and Manasse 1988]. Thus the size of numbers whose factorization is feasible has doubled in ten years, and more advances are sure to follow. Certainly, some of this progress has come from the use of more powerful computers; what may not be so evident is the impact of new techniques, most notably the elliptic curve [Lenstra 1987] and quadratic sieve [Pomerance 1984] algorithms. Both of these algorithms are easy to parallelize on currently available machines.

Given an algorithm, one should always try to find the most general structure to which it applies. Thus, to highlight similarities and hide details, I have used algebraic language wherever possible. Sometimes the level of abstraction is greater than that needed merely to describe an algorithm. I would argue, however, that from this vantage point one can see clearly how the algorithms arise from the basic ideas. Necessarily, some details are lost; for more complete descriptions I refer to the surveys in the reference list (marked with a "*") as well as to the original papers.

In considering running times the reader should equate "step" with "bit operation."

1. Problems related to factoring

The problem of factorization makes sense in any unique factorization domain, of which the most basic example is the ordinary integers $\mathbb{Z}$. Thus we wish to compute the prime divisors of a number n presented as input.

If n is prime, then the problem is easy, as there are efficient randomized algorithms to test primality. With no more work than that of evaluating a power modulo n — an $O(\log n)^3$ process — one can tell if a number is prime, with an error probability of at most 1/4 [Rabin 1980]. If certainty is needed, then a more complicated deterministic algorithm [Adleman et. al. 1983] will *prove* that n is prime in at most $(\log n^{\log\log\log n})^{c+o(1)}$ steps. This algorithm also has a randomized version that is likely to find such a proof within the same time bound; for this it is conjectured that $c = 1/\log 2 \equiv 1.442....$ Finally, a new test due to Atkin and based on complex multiplication has been recently implemented [Morain 1988]; this has proved useful for testing numbers up to 571 digits but it has not yet been analyzed.

In a statistical sense, we understand quite well how numbers factor. One can imagine that a random number n has prime factors whose *lengths* are selected by a "random bisection" process: choose a prime p whose length is uniformly distributed in the interval $(0, \log n)$, replace n by n/p and repeat, and so on. From this one gets intuition about how typical numbers factor as well as an efficient method for generating random numbers together with their factorizations [Bach 1988].

However, we do not know a polynomial time algorithm for factoring, even if we use randomness or make a reasonable assumption such as the extended Riemann hypothesis. We do not even know how to efficiently produce any useful information about the factors

of a number. For instance, one might ask (from a formal analogy with polynomials) if extracting the squarefree part of a number, or just deciding if it is squarefree, takes less time than computing the full factorization; no such result is known. Neither can we count the prime factors of a number in any way better than finding them all.

One often finds factorization problems represented as equation-solving problems. For instance, an algorithm to solve the congruence

$$x^2 \equiv a \pmod{n} \tag{1.1}$$

can be used to efficiently factor n [Rabin 1979]. One could make a formal analogy with (1.1) and speculate that for e relatively prime to the Euler function $\phi(n)$, the congruence

$$x^e \equiv a \pmod{n} \tag{1.2}$$

cannot be efficiently solved without finding information from which one could easily factor n. The security of the RSA cryptosystem [Rivest et. al. 1978] relies on this conjecture as well as on the belief that factoring is difficult.

There is also an *existence* problem related to square roots modulo n: decide whether

$$\exists x [x^2 \equiv a \pmod{n}]. \tag{1.3}$$

This was used in the design of a probabilistic encryption method [Goldwasser and Micali 1982]. A necessary but not sufficient condition for (1.3) to hold is that the Jacobi symbol $(a|n)$ equals 1; this is computable in $O(\log n)^2$ time [Collins and Loos 1982]. Problem (1.3) clearly has *some* relation to factoring, for if a is a quadratic residue modulo n, then for each p dividing n, a is a square modulo p. By quadratic reciprocity, the factors of n are restricted to certain arithmetic progressions. However, recovering the factors from this information seems not to be easy. There is also a relationship between deciding (1.3) and computing $\omega(n)$, the number of distinct prime factors of n, since for odd n, the fraction of quadratic residues in $(\mathbb{Z}/n\mathbb{Z})^*$ is $2^{-\omega(n)}$; however, this does not immediately imply a polynomial-time equivalence between these problems.

More generally, one might wish to decide if, for a number e not prime to $\phi(n)$,

$$\exists x [x^e \equiv a \pmod{n}]. \tag{1.4}$$

This problem has been applied to the design of election protocols [Cohen and Fischer 1985]. It has been argued on heuristic grounds that an efficient algorithm to solve (1.4) for general e and n would lead to an algorithm for factoring that, although not polynomial time, would outperform any currently known on certain numbers [Adleman and McDonnell 1983].

Problems (1.1)-(1.4) are all solvable in random polynomial time for prime moduli and hence (by the Chinese remainder theorem and Hensel's lemma) for moduli whose factorization is known. The first two might be called "zero-dimensional" problems, for the analogous equations over the complex numbers have only finitely many solutions. Despite our intuition that increasing the dimension increases the complexity, similar one-dimensional problems *are* efficiently solvable. In particular, there is an efficient algorithm [Pollard and Schnorr 1987] to solve

$$x^2 - dy^2 \equiv a \pmod{n} \tag{1.5}$$

as well as efficient algorithms for related problems in algebraic number rings [Adleman et. al. 1987].

All of the problems (1.1) - (1.4) make sense if $\mathbb{Z}$ is replaced by a ring and n is replaced by an ideal of finite index. Such generalizations appear not to have been studied much, although cryptographic schemes similar to the RSA have been proposed using algebraic numbers [Williams 1986].

2. Problems related to discrete logarithms

Just as the factorization problem is concerned with rings, the discrete logarithm problem is concerned with groups. Thus let G denote a finite cyclic group, in which the equality predicate, group multiplication, and inverses can be efficiently computed. If g is a generator of G and a another element of G, we wish to solve

$$g^x = a; \tag{2.1}$$

this is the *discrete logarithm problem*. (The restriction to cyclic groups is no constraint because the group generated by an element is always cyclic.) If G has order m, then

$$G \cong \mathbb{Z}/m\mathbb{Z}. \tag{2.2}$$

One can efficiently compute the reverse direction ($g^x \leftarrow x$) of this isomorphism by repeated squaring, with $O(\log x)$ group multiplications. The discrete logarithm problem is that of computing the forward direction. Of course $\mathbb{Z}/m\mathbb{Z}$ has a natural ring structure, and one might ask if the multiplication operation can be transplanted to G; that is, if one can efficiently

$$\text{compute } g^{xy} \text{ given } g^x, g^y. \tag{2.3}$$

This is the *Diffie–Hellman* problem; clearly an algorithm to compute the forward direction of (2.2) (that is, solve (2.1)) can be used to solve it. For most groups of interest, it is unknown if the converse holds, although this has been shown in certain cases for $(\mathbb{Z}/p\mathbb{Z})^*$ [den Boer 1988].

Various groups have been suggested in cryptographic applications of problems (2.1) and (2.3). The original key-exchange proposal [Diffie and Hellman 1978] suggested $(\mathbb{Z}/p\mathbb{Z})^*$ where p is prime; one might also use $\mathbb{F}_q^*$, the multiplicative group of a finite field. There are also possible applications where the ambient group is non-cyclic, employing the unit group $(\mathbb{Z}/n\mathbb{Z})^*$ [Shmuely 1985, McCurley 1987], class groups of imaginary quadratic fields [Buchmann and Williams 1988], and various algebraic groups such as elliptic curves [Miller 1985, Koblitz 1987], abelian varieties [Koblitz 1988], and matrix groups [Varadharajan 1986].

With such an abundance of examples, one might well ask how far the generalization can be pushed. It seems that nothing about (2.1) or (2.3) requires that the group be finite, or even that inverses be computable; perhaps one could use semigroups instead of groups.

3. Algorithms

Remarkably, many of the best algorithms for the problems discussed above have apparent running times that are moderate powers of the following function:

$$L(n) = e^{\sqrt{\log n \log\log n}} \tag{3.1}$$

(here n is the number to be factored or the size of the group and $\log n$ is its natural logarithm). Before presenting algorithms, it will be worthwhile to discuss this function and how it arises.

$L(n)$ is often called a subexponential function because it grows more slowly than n^{ε} for any $\varepsilon > 0$; the appellation "subexponential" is apt because n^{ε} is an exponential function of the length $\log n$. However, most of our intuition deals with polynomial time algorithms, so it is convenient to pretend that $L(n)$ is a polynomial in $\log n$ with a slowly growing exponent, and define $E(n)$ by $L(n) = (\log n)^{E(n)}$. The following values hold:

n	10^{50}	10^{100}	10^{200}	10^{500}	10^{1000}
$\log n$	115	230	460	1151	2303
$E(n)$	4.9	6.5	8.7	12.8	17.2

From the above chart, if an algorithm requires $L(n)^c$ steps, a small reduction in c will have a large effect on its running time.

$L(n)$ arises from considerations of smoothness (a number is *smooth* with respect to a bound M if all its prime factors are less than or equal to M). Briefly, there is a tradeoff between making smooth numbers plentiful (M should be large) and making smooth numbers easy to recognize (M should be small).

To quantify this, we can use the random bisection heuristic cited above to get a plausible estimate for the "probability" $P(\alpha)$ that a random number near q is composed of prime factors less than q^{α}. Conditioning on the first factor's relative length x (which is presumed to be uniformly distributed),

$$P(\alpha) = \int_0^{\alpha} P\left(\frac{\alpha}{1-x}\right) dx;$$

after the change of variable $\lambda = 1/\alpha$ this becomes

$$\rho(\lambda) = \frac{1}{\lambda} \int_{\lambda-1}^{\lambda} \rho(t)\, dt.$$

This equation, together with the initial condition $\rho(\lambda) = 1$ for $0 < \lambda < 1$, defines the *Dickman rho-function*. As a rule of thumb, $\rho(\lambda) \equiv \lambda^{-\lambda}$; consequently,

$$\Pr[x \le q \text{ is } q^{\alpha}\text{-smooth}] \equiv \alpha^{1/\alpha}. \tag{3.2}$$

This can be used in a simple argument that underlies many running time calculations. Consider a two-phase procedure that first assembles a set of M-smooth numbers (with some desired properties) and then processes this set further to complete the algorithm. The first phase simply chooses random candidates (of size roughly q) and adds them to the set if they are smooth. To find the work for this phase, multiply the requisite number of smooth numbers by the work necessary to check a number for smoothness, and divide by the probability that a random number near q is smooth. If first two factors combined produce a term around M^k and the second phase of the algorithm takes M^l steps, then by the approximation (3.2), the total time is roughly

$$T = T_1 + T_2 \cong M^k \lambda^\lambda + M^l \tag{3.3}$$

where $\lambda = (\log q)/(\log M)$. If $T_1 \gg T_2$, we can minimize $\log T_1$ by setting its derivative to zero and find that asymptotically

$$\lambda \cong \sqrt{(2k \log q)/\log\log q},$$

so

$$T = T_1 + T_2 \cong L(q)^{\sqrt{2k}} + L(q)^{\sqrt{l^2/2k}} \tag{3.4}$$

(the first term dominates if $2k \geq l$). Evidently we would like q, k, and l to be small; in fact, much of the progress in factorization and discrete logarithms has come from reducing these parameters.

Naturally, one would like to justify calculations such as the above, but this can be rigorously done only for certain algorithms. The problem is not with the approximation (3.2) — which can be sharpened — but with the tacit assumption that the numbers constructed by the algorithm are smooth with the same probability as random numbers of comparable size. Because in many important cases we are unable to prove this, there has arisen a notion of "heuristic" running time bounds for such algorithms. Thus we distinguish between proofs that an algorithm uses or expects to use only a certain number of steps (so-called "rigorous" bounds) and plausibility arguments for such assertions that always rely on unproved *ad hoc* assumptions. Of course, we can always try out a factoring or discrete logarithm algorithm and see if it works, since any answer produced can be quickly checked. For this reason, heuristic arguments are very useful, even if they are mathematically suspect.

In the descriptions below all running times will be heuristic, unless otherwise noted (the asymptotic notations 'O' and 'o' are reserved for proved results). Furthermore, the calculations are what might be called "first-order": they are only accurate enough to derive the correct value of c in an estimate of the form $L(n)^c$. In particular, they ignore relatively small factors such as powers of $\log n$.

Algorithms for factoring

Most factorization algorithms rely on what might be called a "functorial" approach. The idea is to associate with each ring $\mathbb{Z}/n\mathbb{Z}$ an object X_n in a generic fashion, so that the factorization given by the Chinese remainder theorem transfers to a

factorization of X_n, thus:

$$\begin{array}{ccc} \mathbb{Z}/n\mathbb{Z} & \cong & \mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/q\mathbb{Z} \\ & \downarrow & \\ X_n & \cong & X_p \times X_q \end{array} \tag{3.5}$$

(in this section assume that n has two distinct prime factors p and q). We then use the factorization of X_n to recover the factors of n, usually by constructing special elements of X_n. The easiest way to guarantee that (3.5) occurs is to define X_n with polynomial equations modulo n, though this may not be the only way to proceed.

The best algorithms for factoring numbers composed of two equally large primes are the *quadratic residue* family of algorithms. These algorithms work with the group $X_n = \{x : x^2 \equiv 1 \bmod n\}$, for any element of X_n that is not congruent to $\pm 1 \bmod n$ (at least half of the elements of X_n have this property) will allow us to factor n as $\gcd(x-1, n)$. Equivalently, we can homogenize and seek numbers x and y for which $x^2 \equiv y^2$ but $x \not\equiv \pm y$. The algorithms in this family all do this by performing three basic steps:

1) Generate many quadratic residues mod n.
2) Try to factor them using primes $p \leq M$, to construct congruences of the form $\prod_{p \leq M} p^{e_p} \equiv r^2$.
3) Using linear algebra on the exponents modulo 2, combine the congruences multiplicatively to find x and y with $x^2 \equiv y^2$.

The *continued-fraction* factoring algorithm [Morrison and Brillhart 1970] generates residues around $\sqrt{n}$ in size by evaluating the continued fraction of $\sqrt{n}$, factors them by trial division, and uses Gaussian elimination for the linear algebra. Since roughly M linear equations are needed, we can take $q = n^{1/2}$, $k = 2$, and $l = 3$ in (3.4) to find that the running time is approximately $L(n)^{\sqrt{2}}$.

The *quadratic sieve* algorithm [Pomerance 1984] dispenses with the need for trial division, by using values of a polynomial to form residues around $\sqrt{n}$ in size. Instead of factoring each residue separately, the algorithm processes polynomial arguments one prime at a time, only examining those for which the corresponding value will be divisible by that prime. Neglecting log factors, the amortized cost of factorization per residue may be taken as constant. Using the notation of (3.3), the number of polynomial arguments processed must be the number of smooth residues needed (M) times the inverse smoothness probability (λ^{λ}). If Gaussian elimination is used for the linear algebra, then the running time is the result of taking $k = 1$ and $l = 3$ in (3.3). A good choice for M is obtained by balancing T_1 (the cost of sieving) and T_2 (the cost of equation solving), which leads to a running time of approximately $L(n)^{\sqrt{9/8}}$.

Since a number m has no more than $\log_2 m$ prime factors, the running time of this and similar algorithms can be improved by exploiting the sparsity of the linear equations.

A randomized algorithm based on shift-register synthesis [Wiedemann 1984] will solve an $M \times M$ linear system of equations over a finite field with $O(Mw)$ field operations, if there are w nonzero coefficients. Therefore, for theoretical purposes we may take $l = 2$ in analyzing the Gaussian elimination phase of the quadratic sieve algorithm; this leads to the improved estimate $L(n)$ for the running time.

If one wishes to factor a number with a known or suspected small prime factor p, the algorithm of choice is the *elliptic curve* method [Lenstra 1987]. This takes X_n to be the set of solutions to $y^2 \equiv x^3 + ax + b \pmod{n}$. By the Chinese remainder theorem, $X_n \cong X_p \times X_q$, but X_p has some additional structure. Augmented by an additional "point at infinity" (0:1:0), it forms an abelian group $\bar{X}_p$ with (0:1:0) as the identity (this group is written additively). The group operations are given by rational functions, which can be evaluated mod n. By the Riemann hypothesis for finite fields, $p + 1 - 2\sqrt{p} \le |\bar{X}_p| \le p + 1 + 2\sqrt{p}$, and the group order can be randomized within this interval by varying a and b. If we are lucky and find an M-smooth group (that is, one of M-smooth order), then any element must become the identity when multiplied by $E = \prod_{p \le M} p^{\lfloor \log_p M \rfloor}$. Of course, no rational operations can produce the point at infinity, so a factor is detected when one attempts this multiplication and divides by a non-unit in $\mathbb{Z}/n\mathbb{Z}$. For success, we expect to need only *one* M-smooth group, but by the prime number theorem, multiplication by E requires roughly M operations. The running time is therefore estimated by taking $q = p$, $k = 1$ and $l = 0$ in (3.4); one expects to extract p in approximately $L(p)^{\sqrt{2}}$ steps.

A related algorithm — it does not fit the paradigm (3.5)! — is based on *class groups* [Schnorr and Lenstra 1984]. Here one chooses a random small multiplier μ, and forms a group from the invertible ideals modulo similarity of a subring A of $\mathbb{Q}(\sqrt{-\mu n})$. In the simplest case, $-\mu n$ is the field discriminant, whose divisors are exactly the ramified primes. Solutions to $x^2 = 1$ in the class group lead in a straightforward way to these primes. (Factors can also extracted from square roots of 1 in the general case, but the theory is more complicated). If the group order h depends "randomly" on μ, as suggested by heuristic considerations [Cohen and Lenstra 1984], we may try many values of μ and hope that one of the resulting groups is M-smooth. If so we can annihilate the odd part of the group by brute force, then square repeatedly to find solutions to $x^2 = 1$. Since $h \cong \sqrt{n}$, we can evaluate the running time by taking $q = \sqrt{n}$, $k = 1$ and $l = 0$ in (3.4) and find it to be roughly $L(n)$.

The above discussion cites *three* factorization methods with a conjectured running time near $L(n)$, and one might suspect that this is the true complexity of factoring. However, the algorithms are all based on similar ideas, so it is equally plausible that the $L(n)$ running times are simply a consequence of this similarity. Of these algorithms, the quadratic sieve is the best algorithm in practice (unless we think the number to be factored might have a small prime divisor). It is superior because a typical step in its execution is a single-precision subtraction; a step of the elliptic curve algorithm must evaluate a pair of rational functions (at a cost of $O(\log n)^2$), and a step of the class group algorithm must perform a gcd calculation followed by a 2-dimensional lattice reduction (again, an $O(\log n)^2$ operation).

The *cyclotomic* family of factoring algorithms takes $X_n = (A/nA)^*$, where A is a ring of algebraic integers. In these cases, X_p is a direct sum of finite fields, each of order p^k-1 for some k, and we can easily factor n when any algebraic factor of p^k-1 is smooth [Bach and Shallit 1985]. The practically important cases are $k = 1,2$; that is, the method is useful when $p \pm 1$ is smooth. For example, if $A = \mathbb{Z}$, then the unit group modulo p has order $p-1$, and by raising to a large enough power E we can annihilate this group, factoring n with $\gcd(x^E-1, n)$ [Guy 1976]. The $p+1$ method [Williams 1982] works in a similar fashion with the group of elements in the finite field $\mathbb{F}_{p^2}$ that have norm 1. Both methods have a refinement in which the running time is proportional to the square root of the smoothness bound [Montgomery 1987]; they are useful as preliminary steps in factorization, before a complicated method like the quadratic sieve is used.

Some attention has also been paid to the effects of "second-order" smoothness, that is, smoothness of the automorphism group of $(A/pA)^*$. For example, if the map $x \rightarrow x^e$, an automorphism of $(\mathbb{Z}/p\mathbb{Z})^*$, has a small order t, then we can split n with $\gcd(x^{e^t}-x, n)$. This leads to a requirement that $\phi(p-1)$, the order of the automorphism group, have at least one large factor if p is going to be difficult to remove from n. Similarly, by considering automorphisms of the group of norm-1 elements in $\mathbb{F}_{p^2}$, we see that $\phi(p+1)$ should be chosen to have a large factor.

By properly building primes, the methods of the previous two paragraphs are easy to defend against. What appears to be more difficult is constructing a number that resists the elliptic curve or class group factorization methods. No one knows how to make the smoothness of the groups that occur in these algorithms less likely than the smoothness of random numbers of a comparable size.

A few words should be said here about rigorous analyses of factorization algorithms. Surprisingly, the best known running time for a deterministic factoring algorithm is $n^{1/4+o(1)}$ [Pollard 1974]; this can be lowered to $n^{1/5+o(1)}$ if the Extended Riemann Hypothesis is assumed [Schoof 1982]. The best randomized algorithm for factoring takes expected time $L(n)^{\sqrt{4/3}+o(1)}$ [Vallée 1988], although assuming the ERH, a randomized algorithm related to the class group method has an expected running time of $L(n)^{1+o(1)}$ steps [Lenstra 1987].

Contrasted with the variety of factoring algorithms, very little seems to be known about direct attacks on the RSA encryption scheme (1.2) or the residue problems (1.3) and (1.4). It has been shown that an algorithm to find or guess individual bits of a solution to (1.2) could be used to efficiently find complete solutions [Chor 1986], and that the cost of obtaining individual solutions to (1.2) can be reduced by accumulating other solutions [Desmedt and Odlyzko 1986], but no method to attack these problems has surfaced that is substantially better than factorization. Unfortunately, we cannot rule out the possibility that one exists.

Algorithms for discrete logarithms

The complexity of the discrete logarithm problem depends very much on the group considered. The most general algorithms are "canonical" in the sense that they use only the group operations; however their running times are exponential. In several important cases, though, we know methods with subexponential running times, equal to or better than those of the best factorization algorithms. However, these methods require the group to be specified as part of a larger structure.

The *baby-step/giant-step* algorithm [Shanks 1971] works in any group, as follows. Assume that $|G| \le t^2$, then a solution to $g^x = a$ can be written $x_0 + x_1 t$ with $0 \le x_i < t$. By computing the $2t$ elements g^{x_0} and $a \cdot g^{-x_1 t}$ and looking for a match (one can either sort or use hashing), x can be found in roughly $|G|^{1/2}$ steps (the space requirement is comparable; if $|G|$ is known, this can be reduced with a variant of the "rho" algorithm [Pollard 1978]).

This idea can be extended [Pohlig and Hellman 1978] if G is smooth in the sense of having a long chain factorization, where $|G_i/G_{i-1}| = p$:

$$1 = G_0 \subset G_1 \subset G_2 \subset \cdots \subset G_k = G.$$

Then the index x is expressible as $\sum x_i p^i$, $0 \le x_i < p$, and via the homomorphism $G_i \to G_1$ (raise to the power p^{i-1}), computation of the x_i's reduces to the solution of k discrete logarithm problems in G_1. Using the above algorithm, the complexity is roughly $k\sqrt{p}$.

Finally, assume that the factorization of $m = |G|$ is known: $m = \prod m_i$, where the m_i's are relatively prime. This induces a factorization of G into groups of relatively prime order, and if the m_i's are small we can solve the discrete log problem by going counterclockwise around the following diagram:

$$\begin{array}{ccc} G & \to & \mathbb{Z}/m\mathbb{Z} \\ \downarrow & & \uparrow \\ \prod G_i & \to & \prod \mathbb{Z}/m_i\mathbb{Z} \end{array}$$

(to project G into G_i, raise to the power m/m_i, to go across, solve the problem in each group G_i, and to go up, use the Chinese remainder theorem).

By combining the last two algorithms one sees that, except for a factor that is polynomial in $\log|G|$, the discrete log problem for a p-smooth group is solvable in time roughly $\sqrt{p}$.

In certain groups one can use the *index-calculus* family of algorithms, which work essentially by doing factorization on the left of (2.2) and linear algebra on the right. To use these algorithms G must be specifiable in the following way: start with a ring A that has unique factorization (or more generally, unique ideal factorization), take the free

Abelian group generated by the primes (certain "exceptional" primes may be omitted), and form the quotient group modulo a set of multiplicative identities. If G is represented as such a group, then factorizations in A lead to identities in G, which can be exploited to compute discrete logarithms. An important feature of this family of algorithms is that once one logarithm is computed, others can be found relatively quickly (typically in time equal to the square root of that needed to compute the first logarithm).

For example, take $G = (\mathbb{Z}/p\mathbb{Z})^*$ and $A = \mathbb{Z}$ [Adleman 1980]. For a smoothness bound M, roughly M smooth numbers of the form g^x will serve to tell us the discrete logarithms of all primes up to M. To find them, we try to factor random powers of g using primes less than or equal to M; each successful factorization gives a linear equation in $\mathbb{Z}/(p-1)\mathbb{Z}$ for the logarithms. The time required to construct this "database" can be estimated by taking $k = 1$ and $l = 2$ in (3.3), assuming that a subexponential factorization algorithm and sparse matrix techniques (generalized to finite rings) are used. This gives a time of roughly $L(p)^{\sqrt{2}}$ for the first phase of the algorithm. Once this is completed, computing the logarithm of a requires *one* smooth number of the form $a \cdot g^r$; if r is chosen at random, this will succeed after approximately λ^λ trials, in approximately $L(p)^{\sqrt{2}/2}$ steps.

This method can be modified so that it uses smooth numbers near $\sqrt{p}$ rather than near p [Coppersmith et. al. 1985]. In the analysis one has to replace p by $p^{1/2}$ in the above formulas; if this is done one finds that roughly $L(p)$ steps are needed to find the logarithms of small primes, and the work per additional logarithm is close to $L(p)^{1/2}$.

Similar methods are available for $\mathbb{F}_q^*$ when $q = 2^n$ (or, more generally, a power of a small prime); they have been exhaustively surveyed [Odlyzko 1985]. To study them, one needs an analog of (3.3) for polynomials (since elements of $\mathbb{F}_{2^n}$ are represented in this fashion). Calling a polynomial (over $\mathbb{F}_2$) *d-smooth* if all of its irreducible factors have degree at most d, the analogous approximation to (3.2) is

$$\Pr[\, f \text{ of degree } d \text{ is } \alpha d - \text{smooth} \,] \cong \alpha^{1/\alpha}. \tag{3.6}$$

Assume that the algorithm requires a collection of m-smooth polynomials, each of degree roughly d. Again, the work in assembling them is the size of the collection times the work required to test a candidate (estimated as 2^{mk}) times the inverse smoothness probability λ^λ. Taking $\lambda = d/m$, and assuming a second phase of complexity 2^{ml}, the total time is

$$T = T_1 + T_2 \cong 2^{mk}\lambda^\lambda + 2^{ml} \tag{3.7}$$

which is minimized asymptotically for $m = \sqrt{(d \log d)/(2k \log 2)}$, and leads to

$$T_1 + T_2 \cong M(d)^{\sqrt{2k \log 2}} + M(d)^{\sqrt{l^2 \log 2 / 2k}} \tag{3.8}$$

where $M(d) = e^{\sqrt{d \log d}}$.

The basic index-calculus algorithm in $\mathbb{F}_{2^n}^*$ first tries to find m-smooth polynomials g^x which have degree n. Ignoring log factors, roughly 2^m polynomials are needed. Taking $d = n$, $k = 1$ and $l = 2$ in (3.8) (the time to factor can be neglected [Berlekamp 1967],

and as usual the linear equations are sparse), we find the time for the first phase to be roughly $M(n)^{\sqrt{2\log 2}}$, and the time to extract additional logarithms to be about $M(n)^{\sqrt{\log 2/2}}$. As with $(\mathbb{Z}/p\mathbb{Z})^*$, this can be improved by working with smooth polynomials whose degree is a constant fraction of n [Odlyzko 1985].

The asymptotically fastest algorithm for discrete logarithms in $\mathbb{F}_{2^n}^*$ is an extension of the index-calculus idea that works with smooth polynomials of degree around $n^{2/3}$ [Coppersmith 1984]. It requires time roughly $K(n)^c$, where $K(n) = \exp(n^{1/3}\log^{2/3} n)$ and $c \cong 1.41$ (*not* the square root of 2!).

The above algorithms will compute discrete logarithms in $\mathbb{F}_{p^m}^*$ when p is small or m is 1. Perhaps due to a lack of applications, there are no algorithms known to be efficient when both m and p vary. The basic algorithm can be generalized by replacing $\mathbb{Z}$ by a ring of algebraic integers [ElGamal 1986]; this handles $\mathbb{F}_{p^m}^*$ when m is fixed, but it is unclear how it can be generalized to take account of all cases.

One can use the index-calculus method to find logarithms in $(\mathbb{Z}/n\mathbb{Z})^*$ (this was used in Desmedt and Odlyzko's attack on the RSA scheme), but there is a simpler approach: just factor the group (by factoring n), and solve the problem in each group separately. In some sense, this is the best possible method, because an algorithm to solve arbitrary discrete logarithms modulo n can be used to efficiently factor n. It can also be shown that discrete logarithms in $(\mathbb{Z}/p^e\mathbb{Z})^*$ reduce in polynomial time to discrete logarithms in $(\mathbb{Z}/p\mathbb{Z})^*$, via p-adic logarithms [Bach 1984]. The group $(\mathbb{Z}/n\mathbb{Z})^*$ does have one advantage: we *know* that the Diffie-Hellman problem (2.3) for this group is difficult, if factoring is hard [Shmuely 1985]; this holds in some cases even if the generator g is fixed [McCurley 1987].

There is also an index-calculus algorithm for the class group of an imaginary quadratic field of discriminant $-\Delta$, if the class number h is known [McCurley 1988]. In this case, an ideal $\mathbf{A}$ is called M-smooth if each prime ideal $\mathbf{p}$ dividing it satisfies $N\mathbf{p} \le M$ (the number of prime ideals of norm at most M is roughly the number of ordinary primes at most M, by the prime ideal theorem). Each ideal class contains an ideal of norm at most $\sqrt{\Delta}$, and we can attempt to find the indices of all small prime ideals in the group generated by $\mathbf{g}$ by factoring enough M-smooth ideals of the form $\mathbf{g}^x$ (factorization of an ideal reduces to factorization of its norm in $\mathbb{Z}$), and using linear algebra in $\mathbb{Z}/h\mathbb{Z}$. Analogously to (3.2),

$$\Pr[\mathbf{A} \text{ with } N\mathbf{A} \le q \text{ is } q^\alpha\text{-smooth}] \cong \alpha^{1/\alpha}, \tag{3.9}$$

[Hazlewood 1977], so that the asymptotic complexity of the first stage can be found by taking $q = \sqrt{\Delta}$, $k = 1$, and $l = 2$ in (3.3); it is roughly $L(\Delta)$. To solve $\mathbf{g}^x = \mathbf{A}$ given logarithms of all small prime ideals requires one smooth ideal (of the form $\mathbf{g}^r\mathbf{A}$), therefore time roughly $L(\Delta)^{1/2}$.

Discrete logarithms in elliptic curves and abelian varieties have also been considered [Miller 1985, Koblitz 1987, Koblitz 1988]. These groups have the advantage that the index-calculus algorithm appears not to generalize to them, and if the order of the group is properly chosen, the exponential-time algorithms outlined earlier in this section can be made very expensive.

Since all the discrete logarithm algorithms (except for the baby-step/giant-step procedure) require knowledge of the group order, it is worthwhile to summarize how difficult this is to compute. For $\mathbb{F}_q^*$, the group order is just $q-1$. The orders of the last three groups are more refractory. It is known that any algorithm to compute $\phi(n)$, the order of $(\mathbb{Z}/n\mathbb{Z})^*$, allows one to easily factor n [Miller 1976]; a similar result holds for the class number [Shanks 1971], although $h(-\Delta)$ can be computed in roughly $L(\Delta)^{\sqrt{9/8}}$ steps [McCurley 1988]. Finally, although the number of solutions to $y^2 = x^3 + ax + b \pmod{p}$ can be found in $O(\log p)^8$ steps [Schoof 1985], the degree of this bound is too high for the algorithm to be practical.

Perhaps because the problem lacks the notoriety of factoring, the rigorous analysis of discrete logarithm procedures has not received as much attention. The exponential-time algorithms are easy to analyze; the index-calculus methods, relying on smoothness, are not. However, there are randomized algorithms for discrete logarithms in $(\mathbb{Z}/p\mathbb{Z})^*$ and $\mathbb{F}_{2^n}^*$ whose expected running times can be proved to be $L(p)^{\sqrt{2}+o(1)}$ and $M(n)^{\sqrt{2\log 2}+o(1)}$, respectively [Pomerance 1987].

In contrast to factorization, there is also not much known about the special cases in which discrete logarithms are easy to compute. If the group is smooth, then one can use the factorization of the group to advantage as explained above. In particular, taking $G = (\mathbb{Z}/p\mathbb{Z})^*$, discrete logarithms can be easily found if $p-1$ is smooth. No one knows if the smoothness of $p+1$ (or higher cyclotomic polynomials) helps in this case.

An intriguing unanswered question asks if the complexity of the discrete logarithm problem in $(\mathbb{Z}/p\mathbb{Z})^*$ equals that of the factorization problem. More generally, one would like to classify these and similar problems into degrees of difficulty; although partial results along these lines are known [Shallit and Shamir 1985, Woll 1987, Landau 1988], a complete theory has not yet been developed.

4. Practical considerations

From the above discussion, if one wishes to concoct difficult instances of a factorization or discrete logarithm problem, one must avoid smoothness. In particular, not only must the original structure not be smooth, but neither must any related structures have this property. Unfortunately, without any good lower bounds on computational complexity, we are uncertain exactly what structures count as related. In addition, all of the algorithms discussed in this paper are in some sense algebraic, but this does not eliminate the possibility that methods of a more combinatorial nature could be useful.

In using the heuristic running times developed above, it is important to recognize that first-order formulas like (3.3) tend to overestimate running times, often by several orders of magnitude. For example, evaluating $L(n)^{\sqrt{9/8}}$ (the running time of the quadratic sieve algorithm with Gaussian elimination) at $n = 10^{92}$ gives 3×10^{15} operations, or almost a year if an operation takes 8 nanoseconds. However, an actual 92-digit factorization [te Riele 1988] took 3 days on an NEC SX-2, a machine whose cycle time *is* 8 nanoseconds.

For factorization and discrete logarithms, it would be useful to have a simple "second-order" theory accurate enough to account for such discrepancies. This has yet to be worked out in any detail, but some techniques for improving the estimates can be suggested.

First, although the rough estimate $\rho(\lambda) \cong \lambda^{-\lambda}$ is surprisingly useful for values of practical interest (if $5 \le \lambda \le 10$, it overestimates ρ, by a factor of 4 at most), it is not hard to get better estimates. For example, if ξ denotes the positive root of $e^{\xi} - 1 = \lambda\xi$, and $\mathrm{Ei}(\xi)$ denotes the exponential integral function (that is, the Cauchy principal value of $\int_{-\infty}^{\xi} t^{-1}e^{t}\,dt$), then as $\lambda \to \infty$,

$$\rho(\lambda) \sim \frac{1}{\sqrt{2\pi\lambda}} \cdot \frac{1}{\xi} \cdot e^{-\lambda\xi + \mathrm{Ei}(\xi)}$$

[de Bruijn 1951]. This is already quite accurate; when $\lambda = 5$ the error is only 10%. To get more precision, one can replace the integral in the definition of ρ by an approximation such as Simpson's rule and solve for $\rho(\lambda)$ in terms of "previous" values; this gives an iterative scheme from which it is easy to compute ρ numerically [van de Lune and Wattel 1969]. There is also an asymptotic expression for ρ in terms of elementary functions ($\lambda^{-\lambda}$ is its dominant factor), but it is not very precise unless λ is large.

From the published data [Schnorr and Lenstra 1984] it appears that the probability of smoothness is estimated very well by the Dickman rho-function; this conclusion is also supported by the asymptotic theory [Canfield et. al. 1983]. Once one has a good method to estimate this function, it is not hard to restore the "missing" log factors in formulas such as (3.3) and find a good value for λ numerically. This has been done at least for the continued fraction algorithm [Wunderlich 1985].

For polynomials over IF_2, analogs to the Dickman rho-function have been tabulated and the running times of various discrete logarithm algorithms worked out [Odlyzko 1985]. However, much less is known about the accuracy of estimates such as (3.9), which give the smoothness probability of ideals in algebraic number fields and therefore affect running time estimates for computing discrete logarithms in class groups and extension fields of $\mathbb{Z}/p\mathbb{Z}$.

Of course, one can simply try out algorithms and see how they perform on a variety of machines. The most comprehensive such experiments have been performed with factoring algorithms, most notably using benchmark numbers from the Cunningham project [Brillhart et. al. 1983]. For algorithms such as the quadratic sieve that collect many rows of a matrix, one expects by the law of large numbers that the running time can be extrapolated from the time needed to find a few rows. The elliptic curve and similar algorithms are more chancy; since only one smooth group is required, there is no reliable way to predict when the algorithm will finish.

Finally, some mention should be made of the parallel versions of these algorithms. Algorithms such as the elliptic curve and class group factorization method have a straightforward parallelization: give each processor its own group to try. For the quadratic sieve, much benefit can be gained by using the multiple-polynomial version

[Silverman 1987]. This has the theoretical advantage that the residues sieved are smaller than those of the unadorned algorithm, as well as the practical advantage that each processor can be given its own polynomial from which to generate values for sieving. This was the algorithm that factored the 100-digit Cunningham number $(11^{104}+1)/(11^8+1)$.

5. Acknowledgements

I would like to thank Josh Benaloh, Susan Landau, Kevin McCurley, François Morain, Andrew Odlyzko, Carl Pomerance, and Jeffrey Shallit for helpful comments. The support of the National Science Foundation is also gratefully acknowledged.

6. References

L.M. Adleman, A subexponential algorithm for the discrete logarithm problem with applications to cryptography, in Proceedings of the 1980 IEEE Symposium on Foundations of Computer Science, New York: IEEE (1980).

L.M. Adleman and R. McDonnell, An application of higher reciprocity to computational number theory, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, New York: IEEE (1983).

L.M. Adleman, C. Pomerance, and R.S. Rumely, On distinguishing prime numbers from composite numbers, Annals of Mathematics 117 (1983), pp. 173-206.

* L.M. Adleman and K.S. McCurley, Open problems in number theoretic complexity, in Discrete Algorithms and Complexity (Proceedings of the Japan-US Joint Seminar), London: Academic Press (1987).

L.M. Adleman, D.R. Estes, and K.S. McCurley, Solving bivariate quadratic congruences in random polynomial time, Mathematics of Computation 48 (1987), pp. 17-28.

E. Bach, Discrete logarithms and factoring, University of California at Berkeley Computer Science Division Report 84/186 (1984).

E. Bach and J. Shallit, Factoring with cyclotomic polynomials, Mathematics of Computation 52 (1989).

E. Bach, How to generate factored random numbers, SIAM Journal on Computing 17 (1988), pp. 179-193.

E.R. Berlekamp, Factoring polynomials over finite fields, Bell System Technical Journal 46 (1967), pp. 1853-1859.

B. den Boer, Diffie-Hellman is as strong as discrete log for certain primes, preprint, Centre for Mathematics and Computer Science, Amsterdam (1988).

J. Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, and S.S. Wagstaff, Jr., Factorizations of $b^n \pm 1$, $b = 2,3,5,6,7,10,11,12$ up to High Powers, Providence: American Mathematical Society (1983).

N.G. de Bruijn, The asymptotic behavior of a function occurring in the theory of primes, Journal of the Indian Mathematical Society 15 (1951), pp. 25-32.

J. Buchmann and H.C. Williams, A key-exchange system based on imaginary quadratic fields, Journal of Cryptology 1 (1988).

E.R. Canfield, P. Erdös, and C. Pomerance, On a problem of Oppenheim concerning "Factorisatio Numerorum," Journal of Number Theory 17 (1983), pp. 1-28.

B.-Z. Chor, Two Issues in Public Key Cryptography, Cambridge: MIT Press (1986).

H. Cohen and H.W. Lenstra, Jr., Heuristics on class groups of number fields, in Number Theory (Lecture Notes in Mathematics 1068), Berlin: Springer (1984).

J.D. Cohen and M.J. Fischer, A robust and verifiable cryptographically secure election scheme, in Proceedings of the 26th Annual ACM Symposium on Foundations of Computer Science, New York: IEEE (1985).

G. Collins and R. Loos, The Jacobi symbol algorithm, SIGSAM Bulletin 16 (1982), pp. 12-16.

D. Coppersmith, Fast evaluation of logarithms in fields of characteristic two, IEEE Transactions on Information Theory 30 (1984), pp. 587-594.

D. Coppersmith, A.M. Odlyzko, and R. Schroeppel, Discrete logarithms in GF(p), Algorithmica 1 (1986), pp. 1-15.

Y. Desmedt and A.M. Odlyzko, A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes, in Proceedings of CRYPTO '85 (Lecture Notes in Computer Science 218), Berlin: Springer (1986).

W. Diffie and M. Hellman, New directions in cryptography, IEEE Transactions on Information Theory 22 (1978), pp. 644-654.

T. ElGamal, On computing logarithms over finite fields, in Proceedings of CRYPTO '85 (Lecture Notes in Computer Science 218), Berlin: Springer (1986).

S. Goldwasser and S. Micali, Probabilistic encryption, Journal of Computer and System Sciences 28 (1984), pp. 270-299.

*R.K. Guy, How to factor a number, in Proceedings of the Fifth Manitoba Conference on Numerical Mathematics (1976).

D.G. Hazlewood, On ideals having only small prime factors, Rocky Mountain Journal of Mathematics 7 (1977), pp. 753-768.

N. Koblitz, Elliptic curve cryptosystems, Mathematics of Computation 48 (1987), pp. 203-209.

N. Koblitz, A family of Jacobians suitable for discrete log cryptosystems, Proceedings of CRYPTO '88, Berlin: Springer (1989).

S. Landau, Some remarks on computing the square parts of integers, Information and Computation 78 (1988), pp. 246-253.

A.K. Lenstra, Fast and rigorous factorization under the generalized Riemann hypothesis, University of Chicago Computer Science Department Report 87-007 (1987) [to appear, Indagationes Mathematicae].

*A.K. Lenstra and H.W. Lenstra, Jr., Algorithms in number theory, to appear in Handbook of Theoretical Computer Science, Amsterdam: North-Holland.

A.K. Lenstra and M. Manasse, 100 digit factorization, announcement (1988).

H.W. Lenstra, Jr., Factoring integers with elliptic curves, Annals of Mathematics 126 (1987), pp. 649-673

J. van de Lune and E. Wattel, On the numerical solution of a differential-difference equation arising in analytic number theory, Mathematics of Computation 23 (1969), pp. 417-421.

K.S. McCurley, A key distribution system equivalent to factoring, preprint, IBM Almaden Research Center (1987).

K.S. McCurley, Cryptographic key distribution and computation in class groups, to appear in Proceedings of the NATO Advanced Study Institute on Number Theory and Applications (Banff, May 1988), Dordrecht: Reidel. [Available as IBM Almaden Research Center Technical Report #6433.]

G.L. Miller, Riemann's hypothesis and tests for primality, Journal of Computer and System Sciences 13 (1976), pp. 300-317.

V. Miller, Use of elliptic curves in cryptography, in Proceedings of CRYPTO '85 (Lecture Notes in Computer Science 218), Berlin: Springer (1986).

P.L. Montgomery, Speeding the Pollard and elliptic curve methods of factoring, Mathematics of Computation 48 (1987), pp. 243-264.

F. Morain, Implementation of the Goldwasser-Kilian-Atkin primality testing algorithm, University of Limoges / INRIA Report (1988).

M.A. Morrison and J. Brillhart, A method of factoring and the factorization of F_7, Mathematics of Computation 29 (1975), pp. 183-205.

J.M. Pollard, Theorems on factorization and primality testing, Proceedings of the Cambridge Philosophical Society 76 (1974), pp. 521-528.

J.M. Pollard, Monte Carlo methods for index computation (mod p), Mathematics of Computation 32 (1978), pp. 918-924.

J.M. Pollard and C.-P. Schnorr, An efficient solution of the congruence $x^2+ky^2\equiv m \pmod n$, IEEE Transactions on Information Theory IT-33 (1987), pp. 702-709.

S. Pohlig and M. Hellman, An improved algorithm for computing logarithms over GF(p) and its cryptographic significance, IEEE Transactions on Information Theory 24 (1978), pp. 106-110.

* C. Pomerance, Analysis and comparison of some integer factoring algorithms, in Computational Methods in Number Theory (v. 1), edited by H.W. Lenstra, Jr., and R. Tijdeman, Amsterdam Mathematical Centre Tract #154 (1982).

C. Pomerance, The quadratic sieve factoring algorithm, in Proceedings of EUROCRYPT 84 (Lecture Notes in Computer Science 209) Berlin: Springer (1985).

C. Pomerance, Fast rigorous factorization and discrete logarithm algorithms, in Discrete Algorithms and Complexity, Proceedings of the Japan-US Joint Seminar, London: Academic Press (1987).

* A.M. Odlyzko, Discrete logarithms and their cryptographic significance, Proceedings of EUROCRYPT '84 (Lecture Notes in Computer Science 209), Berlin: Springer (1985).

M.O. Rabin, Digitalized signatures and public-key functions as intractable as factorization, MIT Laboratory for Computer Science Report TR-212 (1979).

M.O. Rabin, Probabilistic algorithm for testing primality, Journal of Number Theory 12 (1980), pp. 128-138.

H. te Riele, W. Lioen and Dik Winter, New factorization records, announcement (1988).

R. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (1978), pp. 120-126.

D. Shanks, Class number, a theory of factorization, and genera, in Proceedings of Symposia in Pure Mathematics 20, Providence: American Mathematical Society (1971).

C. Schnorr and H.W. Lenstra, Jr., A Monte Carlo factoring algorithm with linear storage, Mathematics of Computation 43 (1984), pp. 289-311.

R. Schoof, Quadratic fields and factorization, in Computational Methods in Number Theory (v. 2), edited by H.W. Lenstra, Jr., and R. Tijdeman, Amsterdam Mathematical Centre Tract #155 (1982).

R. Schoof, Elliptic curves over finite fields and the computation of square roots mod p, Mathematics of Computation 44 (1985), pp. 483-494.

J. Shallit and A. Shamir, Number-theoretic functions which are equivalent to number of divisors, Information Processing Letters 20 (1985), pp. 151-153.

Z. Shmuely, Composite Diffie-Hellman public-key systems are hard to break, Technion Computer Science Department Report 356 (1985).

R.D. Silverman, The multiple polynomial quadratic sieve, Mathematics of Computation 48 (1987), pp. 329-339.

B. Vallée, Quasi-uniform algorithms for finding small quadratic residues and application to integer factorisation, preprint, Université de Caen (1988) [Presented at 1988 AMS Computational Number Theory Conference].

V. Varadharajan, Trapdoor rings and their use in cryptosystems, in Proceedings of CRYPTO '85 (Lecture Notes in Computer Science 218), Berlin: Springer (1986).

D.H. Wiedemann, Solving sparse linear equations over finite fields, IEEE Transactions on Information Thery 32 (1986), pp. 54-62.

H.C. Williams, A $p+1$ method of factoring, Mathematics of Computation 39 (1982), pp. 225-234.

H.C. Williams, An M^3 public-key encryption scheme, in Proceedings of CRYPTO '85 (Lecture Notes in Computer Science 218), Berlin: Springer (1986).

* H. Woll, Reductions among number-theoretic problems, Information and Computation 72 (1987), pp. 167-169.

M. Wunderlich, Implementing the continued fraction factoring algorithm on parallel machines, Mathematics of Computation 44 (1985), pp. 251-260.