Factoring

Arjen K. Lenstra

Room MRE-2Q334, Bellcore, 445 South Street, Morristown, NJ 07960, USA
E-mail: lenstra@bellcore.com
July 13, 1994

Abstract. A brief survey of general purpose integer factoring algorithms and their implementations.

1 Introduction

Factoring, finding a non-trivial factorization of a composite positive integer, is believed to be a hard problem. How hard we think it is, however, changes almost on a daily basis. Predicting how hard factoring will be in the future, an important issue for cryptographic applications of composite numbers, is therefore a challenging task.

So far, such predictions have not been very successful. In a 1976 survey paper on factoring [21], the factorization of numbers 'without special form' having 80 or more decimal digits was not expected before the end of the century. In 1977, it was thought that 126 digits would require 40 quadrillion years [18], on an imaginary computer that is 10^5 times faster than a Sparc 10 workstation [3; 45]. Right now, many people rely on the unbreakability of 155-digit composites to secure their data.

General 80-digit numbers can be factored since the mid-eighties, at first with considerable effort [50], currently within a few days on an ordinary workstation — at least fifteen years ahead of the expected schedule. Composites in the 100 to 115 digit range can now routinely be factored on a moderately fast supercomputer or a network of workstations, with run times varying from a day to a few months [15]. The record factorization of a 129-digit number in April 1994 on a world-wide network of 1600 machines took 8 months — about 10^{19} times easier than predicted [3]. Data protected by 155-digit keys might still be secure, but it is 'a bit unnerving' [34] that we are getting so close.

We sketch some of the techniques that contributed to these developments. While making no attempt to predict the future, we also touch upon more recent methods that might soon enable us to prove that the situation is actually more than just a bit unnerving for 155-digit key users.

The methods on which the above predictions are based would now be called *special purpose* algorithms: methods for which the run-time depends strongly on special properties of the factor to be found, but only polynomially on the size of the number being factored. Examples are *trial division*, *Pollard's $p-1$ method* [39], *Pollard's rho method* [40], *Shanks's 'squfof'* [47], and the *elliptic curve*

method (ECM) [31]. Very large numbers can be factored using these methods, if they have at most one large factor. For instance, the 617-digit eleventh Fermat number $F_{11} = 2^{2^{11}} + 1$ was completely factored into two 6, one 21, one 22, and one 564-digit prime factor using ECM [5; 6], in 1988. On the other hand, many 100-digit numbers are too hard for current implementations of any of these methods — the largest penultimate factor ever found using a special purpose algorithm has 43 digits.

In these notes we restrict ourselves to *general purpose* algorithms: methods for which the run-time depends solely on the size of the number being factored. At least one such method existed back in 1970 — it would have led to more realistic predictions, had its expected run-time been known. This method and its successors are reviewed in Section 2. Various methods that have been developed to meet the ever-increasing run-time demands of modern factoring projects are discussed in Section 3. It can be concluded that if factoring research would get even a fraction of the money that flowed into cancelled high energy physics or ill-focussed space projects, no serious cryptographic application would use 155-digit composites.

2 General purpose factoring algorithms

Let $n > 1$ be an odd integer which is not a prime power.[1] In the algorithms to factor n that we discuss here,[2] several pairs of integers x, y satisfying $x^2 \equiv y^2 \bmod n$ are constructed in a more or less random fashion. Because n divides $x^2 - y^2 = (x-y)(x+y)$ we find that $n = \gcd(n, x-y)\gcd(n, x+y)$. As shown in [16] there is a probability $\geq 1/2$ that this factorization is non-trivial for each randomly constructed pair. A few of them will therefore suffice to factor n.

To find such pairs, the algorithms proceed in two steps, a *relation collection step* and a *matrix step*. Let P be some set of integers, the *factor base*. It usually consists of some subset of the primes $\leq B$ for some bound B, and often includes -1 as well. In the first step a set V of $> \#P$ integers v is collected such that

$$v^2 \equiv \prod_{p \in P} p^{e_p(v)} \bmod n, \tag{2.1}$$

for $e(v) = (e_p(v))_{p \in P}$ in $\mathbf{Z}^{\#P}$. In the second step linear dependencies modulo 2 among the sparse bit-vectors $e(v) \bmod 2$ are computed. Each dependency corresponds to a set $W \subset V$ for which $\sum_{w \in W} e(w) = (2w_p)_{p \in P}$ for integers w_p. Consequently, $x = \prod_{w \in W} w$ and $y = \prod_{p \in P} p^{w_p}$ satisfy $x^2 \equiv y^2 \bmod n$.

The matrix step is the same for all algorithms. For factor bases having up to about half a million elements *structured Gaussian elimination* [24; 43] works

---

[1] It can easily be verified that n satisfies these conditions, without getting additional knowledge about its factors [25: 5.1].

[2] For other, less practical general purpose factoring algorithms, see [25: 4.A, 4.10].

well [3]. It first reduces the original sparse matrix to a much smaller but matrix, using 'cheap' elimination steps that do not cause fill-in. The re- dense matrix is then processed using ordinary Gaussian elimination. See r an informal description of this method. For larger factor bases the more methods from [10; 11; 14; 23; 36; 52] look promising [35].

ow the pairs $(v, e(v))$, the *relations*, are found and how P is chosen depends e algorithm. The simplest way to generate relations is the *random squares* od [16]: pick integers v at random, and test if the least absolute remainder modulo n can be factored using the elements of P. The advantage of this od is that it is one of the few factoring algorithms that allows a rigorous ted run-time analysis. The disadvantage is, however, that the quadratic ues to be tested are of the same order of magnitude as n. Other methods ate much smaller quadratic residues, which are therefore more likely to r over P, thus making those methods more practical.

One way to do this was suggested by Morrison and Brillhart [37], who the first to use the two step approach in their *continued fraction method* RAC), based on a method from 1931. If a_i/b_i is the ith continued fraction ergent to $\sqrt{n}$, then $r_i = a_i^2 - nb_i^2$ satisfies $|r_i| < 2\sqrt{n}$ (cf. [22]). Each i for h r_i can be factored over P yields a relation as in (2.1), with v replaced $_i$. Therefore, for $i = 1, 2, \ldots$ in succession, the quadratic residue r_i is in- ted using for instance trial division, until sufficiently many relations have found. If the continued fraction expansion of $\sqrt{n}$ is too short, n can be aced by a small multiple. The most notable success of CFRAC was the fac- zation of the 39-digit seventh Fermat number $F_7 = 2^{2^7} + 1$, in 1970. In the y eighties, 60-digit numbers could be factored in about a day on a special- pose CFRAC-machine — the 'Georgia Cracker'.

A heuristic analysis of the expected run-time of CFRAC appeared only in 4 in [42]. By that time a heuristic run-time estimate of the relation collection p of yet another general purpose factoring method had already appeared in], in 1978. The algorithm in question, Schroeppel's *linear sieve*, not only used all quadratic residues, as CFRAC, but also replaced the trial divisions by a ve to collectively test the quadratic residues. Schroeppel was on the brink of toring the 78-digit eighth Fermat number $F_8 = 2^{2^8} + 1$ using his method [48], ien its 'rho'-factorization was announced by Brent and Pollard [7] in 1980.

Although this was a remarkable success for Brent's modified version of the o-method, it was an unfortunate set-back for general purpose factoring, sim- y caused by the fact that F_8 happened to have a small penultimate factor (of ily 16 digits). Had F_8 been a bit unluckier for the rho-method, then Schroep- el might have been more inspired to push his methods, and they would have ceived the attention they deserved.

As it was, the only person who paid enough attention to Schroeppel's method as Pomerance. He came up with a more efficient modification, which he coined ne *quadratic sieve method* (QS) [42]. Let $f(i) = (i + [\sqrt{n}])^2 - n$ with $i \in \mathbf{Z}$, then $f(i)| \approx 2|i|\sqrt{n}$ for small i. If $f(i)$ factors over P, then $v = i + [\sqrt{n}]$ satisfies (2.1) nd yields a relation. The quadratic residue $f(i)$ is reasonably small, though for

large i considerably bigger than the quadratic residues in CFRAC. But they can be tested much faster, using a sieve, because if p divides $f(i)$ for some $i \in \mathbf{Z}$, it divides $f(i + kp)$ for any integer k.

The fact that $|f(i)|$ grows linearly with $|i|$ makes finding relations harder with growing $|i|$. Still, the idea of QS looked promising and various implementations were attempted. After a 47-digit number had been factored using the original QS [19], an improvement called the *special-q variation* made headlines in 1983 by factoring a 71-digit number [12] — a new general purpose factoring record, but well short of the 78-digit number almost factored by Schroeppel.

The 80-digit barrier was broken by the even more efficient *multiple polynomial variation* (MPQS) [50]. In MPQS, which is due to P.L. Montgomery, the single polynomial f is replaced by a sequence of polynomials with similar properties. This makes it possible to switch from one polynomial to the next, as soon as the quadratic residues would become too large. Another advantage is that the algorithm is ideally suited for distributed implementation. The first few billion polynomials in the sequence each produce more or less the same number of relations, and the polynomials can be processed in any order. Furthermore, each relation is just as good as any other, irrespective of what polynomial was used to find it. Therefore different machines can collect relations by sieving with different polynomials or with different subsequences of polynomials. There is no need for synchronization, the relative speed of the machines is irrelevant, and communication is only needed to ensure that the machines process different tasks and to collect the resulting relations.

Combined with 'large prime variations', the most recent one being the *double large prime variation* [30], these techniques led to a series of factoring records from 1985 to the present day: the first 100-digit number in 1988 [29], 120 digits in 1993 [13], and the current general purpose factoring record, a 129-digit number in April 1994 [3]. Faster but more space-consuming methods to switch between polynomials can be found in [2; 38; 44]

In all algorithms discussed so far, the quadratic residues are roughly speaking of the order $n^{O(1)}$. It follows that, asymptotically, all algorithms have expected run time $L_n[1/2, c]$, for different values of c, where

$$L_x[v, \lambda] = exp((\lambda + o(1))(\log x)^v (\log\log x)^{1-v}),$$

for $v, \lambda \in \mathbf{R}$ and $x \to \infty$. Only for the random squares method this can be proved rigorously, for the others presented here it is based on heuristic assumptions. For the linear sieve and all variations of QS we get $c = 1$, where we use the result from [52] that a dependency among the rows of a sparse $(s + 1) \times s$ bit-matrix can be found in time $s^{2+o(1)}$, for $s \to \infty$. We also get $c = 1$ for CFRAC if we replace the ordinary trial division by ECM. For the random squares method this leads to $c = \sqrt{2}$. Thus, we have three different general purpose factoring methods that all have the same heuristic asymptotic expected run-time [3] $L_n[1/2, 1]$. And,

[3] For a fourth one, based on the use of class groups, see [25: 4.10]. For a variant of this method the expected run-time can rigorously be proved to be $L_n[1/2, 1]$, cf. [32].

maybe more surprisingly, they share this run-time with the worst case of ECM (where n is the product of only two factors of about the same size). For proofs of these statements, see [25; 42].

Notice that $L_x[0,\lambda] = (\log x)^{\lambda+o(1)}$ and that $L_x[1,\lambda] = x^{\lambda+o(1)}$. Thus, $L_n[v,\lambda]$ for v going from 0 to 1 interpolates between polynomial-time and exponential-time. The above run-time $L_n[1/2,c]$ for factoring algorithms can therefore be interpreted as being halfway between polynomial-time and exponential-time.

This remarkable observation, that we have several conceptually different factoring methods that have the same asymptotic run-time, was made in 1985, right after the invention of ECM. The hope among 'composite traffickers', however, that we had hit upon the 'true complexity' of factoring and that we were forever stuck halfway, was dashed in 1989 by the analysis of Pollard's 1988 *number field sieve* (NFS) [27; 41]. The quadratic residues of order $n^{O(1)}$ that have to factor over P are, in NFS, replaced by numbers of order only $n^{o(1)}$. This causes a substantial step in the direction of polynomial-time factoring algorithms, in the above metric: the asymptotic expected run-time of NFS is $L_n[1/3,c]$ for some c with $1.53 \approx (32/9)^{1/3} \leq c \leq (64/9)^{1/3} \approx 1.92$ — a fast $c \approx 1.53$ for 'nice' numbers as $2^{2^9}+1$ and $2^{523}-1$, but a much slower $c \approx 1.92$ for general numbers, i.e., numbers without special form.

We sketch why NFS is more efficient for 'nice' numbers than for general ones. An informal description of NFS can be found in [28]; for full details we refer to the papers in [26]. Let d be some small integer: close to 5 when n has around 130 digits, but more generally of order $((\log n)/(\log\log n))^{1/3}$. Let $m \in \mathbf{Z}$ be close to a dth root of tn for some small $t \in \mathbf{Z}$, and let $tn = \sum_{j=0}^{d} f_j m^j$ with $|f_j| \leq m/2$; the f_j can be obtained by writing tn symmetrically in base m. Notice that m and the f_j are only $n^{o(1)}$. We say that n is 'nice' if the f_j can be bounded by some constant independent of n. For instance, for $n = 2^{2^9}+1$ we might take $t = 8$, $m = 2^{103}$, $f_0 = 8$, $f_5 = 1$ and $f_j = 0$ for $j = 1,2,3,4$.

The polynomial $f(X) = \sum_{j=0}^{d} f_j X^j \in \mathbf{Z}[X]$ may be assumed to be irreducible (or else we can easily derive a factorization of n). Let K be the number field $\mathbf{Q}[X]/f\mathbf{Q}[X]$, i.e., $K = \mathbf{Q}(\alpha)$ with $f(\alpha) = 0$. Because of the way m was chosen, the mapping ϕ from $\mathbf{Z}[\alpha]$ to $\mathbf{Z}/n\mathbf{Z}$ with $\phi(\alpha) = (m \bmod n)$ is a ring homomorphism. Suppose we have small integers a, b such that $a+b\alpha = \prod_{p\in P_1} p^{e_p(a,b)}$ for some algebraic factor base $P_1 \subset \mathbf{Z}[\alpha]$ consisting of elements of small norm, and $a + bm = \prod_{p\in P_2} p^{e_p(a,b)}$ for some integer factor base P_2. From $\phi(a+b\alpha) \equiv a + bm \bmod n$ it follows that

$$\prod_{p\in P_1} \phi(p)^{e_p(a,b)} \equiv \prod_{p\in P_2} p^{e_p(a,b)} \bmod n.$$

With $P = P_1 \cup P_2$ this is an identity of the form (2.1), for instance after dividing by the left hand side. Thus, in the matrix step more then $\#P_1 + \#P_2$ of such identities can be combined into a solution to $x^2 \equiv y^2 \bmod n$.

The algebraic integer $a + b\alpha$ can be factored over P_1 if and only if its norm $(-b)^d f(-a/b)$ can be factored into small primes. Because $f_j = n^{o(1)}$, the norm

is also only $n^{o(1)}$ for small a, b. Similarly, the integer $a + bm$ that has to factor over P_2 is only $n^{o(1)}$ for small a, b. Notice that the norm becomes more likely to factor if the f_j are only constants. This explains why NFS is so much better for 'nice' numbers.

The computation of $\phi(p)$ for $p \in P_1$ requires explicit expressions in $\mathbf{Z}[\alpha]$ for the elements of P_1. In exceptional circumstances it might be possible to find such expressions, for instance some 'nice' numbers might be so lucky, but in general this cannot be expected. We refer to [26] for ways around this problem.

NFS soon proved to be practical for 'nice' numbers, as shown by the 1990 factorization of the 148-digit composite factor of $2^{2^9} + 1$, a number that was believed to be out of reach for a long time to come [28]. It took much longer to get NFS to work for numbers without special form, among others because of the problem referred to above. In June 1994 the NFS-factorization of a general 105-digit number was announced [35], and a 116-digit factorization is forthcoming [17]; see also [4; 8]. Even though NFS is asymptotically superior to QS (and all its variations), it has long been unclear for what size numbers NFS could be expected to outperform QS in practice. This issue has not been settled yet, but the current belief, based on the latest implementations and experiments [17; 20] is that the crossover point lies around 115 digits, well below earlier guesses of 124 digits, 140 to 150 digits, or even 330 digits [1]. This would imply that the QS-factorization reported in [3] could have been found substantially faster if NFS had been used.

In March 1994 a polynomial-time factoring algorithm was found by Shor [49]. The catch is, however, that instead of using the 'traditional' von Neumann computer, Shor's method uses a 'quantum Turing machine'. So far, building such a machine seems to be infeasible, and simulating the algorithm on a regular Turing machine seems to require exponential time. The practical impact of the algorithm is therefore probably rather limited for the time being.

3 Distributed and parallelized factoring

The relation collection step and the matrix step of the general purpose factoring algorithms from the previous section have the same asymptotic run-time. In practice there is a huge difference between the two run-times. For example, for the factorizations in [13] and [3] the matrix step took only 1/500th of the effort spent on the relation collection. Most likely, this fraction will become even smaller with improved sparse matrix algorithms [36].

For the time being, the limits of our factoring capabilities therefore depend on what can be done in the relation collection step. Traditionally, relation collection was done on single processor mainframes or supercomputers. The 71-digit QS-record from [12], for instance, took 9.5 hours on a Cray X-MP. Getting access to large machines, however, is expensive, and sponsors who are willing to sink a fortune in the next factoring record are hard to find. In this situation where financial and organizational worries were harder to overcome than the techni-

cal problems, the factoring achievements no longer reflected our true factoring capabilities.

As explained in Section 2, the relation collection step of MPQS lends itself extremely well to a distributed implementation. A cost-effective way to do this was mentioned in [50] and is described in detail by Caron and Silverman in [9]. They present two methods to run MPQS on a local area network of workstations: the *embarrassingly parallel approach* and the *star configuration*. In the first each machine runs independently of all other machines, collecting relations for the same number, but using its own sub-sequence of polynomials. No inter-processor communication is required. Every now and then the relations from all machines have to be collected, and machines have to be restarted for each new number or after a crash. In the star configuration one host machine computes the sequence of polynomials and farms the polynomials out to each satellite processor that requests a task. No inter-satellite communication is required, but the host constantly monitors all satellites, and restarts them if necessary. In [9] the latter approach was preferred because it required less handholding than the first one where machines had to be restarted 'manually'. Notice, however, that with a crash of the host the entire star configuration dies. Furthermore, the number of satellites that can be served by one host is limited; it depends on the relative speed of generating versus processing the polynomials, and on the communication overhead. There is no limit on the number of machines that can be employed in the embarrassingly parallel approach.

Both approaches achieve an N-fold speed-up when run on N machines. They can both easily be set up in such a way that the relation collection does not interfere with the 'normal' operation of the satellite machines, so that they only use cycles that would otherwise have been idle.

Using the star configuration the general-purpose factoring records quickly raised from 71 to 87 digits — the latter on 8 to 10 Sun workstations in about 550 CPU hours per machine and at a fraction of the cost of a similar computation on a supercomputer.

An independent project started out as an attempt to replace the default screensaver program running on the workstations at DEC's Systems Research Center by a program that would still save screens, but that would also do something else that might be useful. Instead of spending all cycles on constantly redrawing the screen, it was proposed to run ECM combined with a less computationally intensive screensaver. This incorporation of ECM in the screensaver never materialized. What emerged instead was a star-like ECM-implementation, described in detail in [29]. Later, and still independently from [9], this work led to an embarrassingly parallel MPQS-implementation, also described in [29], that was supposed to be more general and easier to use than the one from [9].

It consists of several shell-scripts, a relation collection program, and some input files that depend on the number being factored. On each client that helps factoring, the shell-scripts monitor the relation collection program — providing it with inputs, pausing it when necessary, killing and restarting it if the number to be factored changes. They also make sure that the factoring process resumes

after a crash or a reboot of the client by putting themselves in the client's `at`-queue at regular intervals. The relation collection program generates a unique sub-sequence of polynomials and uses them to look for relations. Depending on the input file the resulting relations are either concatenated to some local file, or sent to the e-mail address of some central collection site. For numbers of up to 115 digits the relation collection program needs at most 3 MBytes to run; the 129-digit number from [3] required 8 MBytes.

Notice that there is no way to ensure that any remote client actually processes its inputs, or that it processes the inputs using the relation collection program that it is supposed to use — anyone running this software can change and/or corrupt any part of it as desired. There are two reasons why this liberal approach works. In the first place, the number of relations to be collected is at most, say, ten million. Each input, when run to completion, can in principle generate at least, say, 10^5 relations. Thus, it would suffice if about 100 inputs were completely processed. But there are millions of 'good' inputs, each with approximately the same yield. Therefore, even if only a small fraction of the inputs is only very partially processed, more than enough relations will be found.

Secondly, at the central collection site the newly received would-be relations can easily be checked for correctness using (2.1), put in some standard format, and sorted and merged with the older ones (thereby removing duplicates). Anything that gets accepted is useful, even though it might not have been generated by the original relation collection program. Intentionally or unintentionally corrupted data are simply thrown away.

The only remaining problem is to find people who would be interested to donate the otherwise idle cycles on their workstations to large factoring projects that might run for months. This turned out to be surprisingly easy: a few postings on various Internet newsgroups (like `sci.crypt` and `sci.math`) with a description of the project and instructions how to get the software sufficed. The question remains why so many people responded.

This implementation has become known as 'factoring by e-mail' because all inter-processor communication (including the distribution of the shell-scripts, the program, and the input files) can be handled by electronic mail. After the software has been distributed, the amount of communication depends on the number of relations sent by the clients to the central collection site. For the factorization from [3], one relation fits easily in 350 bytes. A Sparc 10 workstation produces at most about 200 relations per day, which implies that a Sparc 10 client sends at most 80 KBytes per day, say in 8 batches of 25 relations each. With 2000 of such clients the central site would get 160 MBytes per day, a load that can easily be handled by the Internet. If these numbers would get much larger,[4] however, it would be advisable to use more than one collection site for the e-mail relation collection and data checking, and to use `ftp` to collect the data at one central site.

[4] This can easily happen if numbers much smaller than the one from [3] are factored using the same size network, since data is generated more quickly for smaller numbers.

As mentioned in Section 2 'factoring by e-mail' resulted in a sequence of general purpose factoring records. The current champion is the 129-digit number from [3], for which all cycles were donated by volunteers on a loosely-coupled world-wide network of about 1600 machines.

A disadvantage of 'factoring by e-mail' is that it is hard to optimize the code, because there is no way to predict on what kind of platforms or under what operating systems volunteers will attempt to run it. During the design portability was a bigger concern than efficiency, for the obvious reason that it does not make sense to make the program 20% faster while at the same time restricting it to 50% of the otherwise available resources. In the version from [3] some attempts were made, using compile options, to distinguish between machine types, something that was not done in the original programs from [29]. Portability is not an issue in the other distributed QS-implementations reported in the literature, because they used either a single local area network [9], or a large collection of stand-alone (not networked) more or less identical PC's (using floppy disks to communicate) [2].

A similar 'factoring by e-mail' approach to NFS resulted in the factorization of the ninth Fermat number $2^{2^9} + 1$ (cf. [28]). Unlike QS the number of useful inputs for NFS is rather limited. The sieving program running on the remote clients therefore not only reported the relations it had found, but also which inputs it had completed. In this way inputs could be redistributed if they were not completed within, say, ten days after being given out.

Obviously, any method that can be implemented efficiently on an e-mail network of machines can also be made to run efficiently on any *Multiple Instruction Multiple Data* machine, irrespective of its topology, if the nodes have enough memory. It is not so obvious that *Single Instruction Multiple Data* (SIMD) machines can be used for the relation collection step of either QS or NFS. Successful implementations on a 128×128 toroidal mesh of small processors can be found in [15] (QS) and [4] (NFS). In both these implementations it is essential that the individual processors allow indirect addressing, which is not the case on all SIMD machines.

References

1. L. M. Adleman, *Factoring numbers using singular integers*, proc 23rd Annual ACM Symposium on Theory of Computing (STOC) (1991) 64–71
2. W. R. Alford, C. Pomerance, *Implementing the self initializing quadratic sieve on a distributed network*, manuscript, 1994
3. D. Atkins, M. Graff, A. K. Lenstra, P. C. Leyland, *THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE* (in preparation)
4. D. J. Bernstein, A. K. Lenstra, *A general number field sieve implementation*, 103–126 in: [26]
5. R. P. Brent, *Factorization of the eleventh Fermat number (preliminary report)*, Abstracts Amer. Math. Soc. **10** (1989) 89T-11-73

6. R. P. Brent, *Parallel algorithms for integer factorisation*, pp. 26–37 in: J. H. Loxton (ed.), *Number theory and cryptography*, London Math. Soc. Lecture Note Series **154**, Cambridge University Press, Cambridge, 1990
7. R. P. Brent, J. M. Pollard, *Factorization of the eighth Fermat number*, Math. Comp. **36** (1981) 627–630
8. J. Buchmann, J. Loho, J. Zayer, *An implementation of the general number field sieve*, Advances in Cryptology, Crypto '93, Lecture Notes in Comput. Sci. **773** (1994) 159–165.
9. T. R. Caron, R. D. Silverman, *Parallel implementation of the quadratic sieve*, The Journal of Supercomputing **1** (1988) 273–290
10. S. Coppersmith, *Solving linear equations over GF(2): block Lanczos algorithm*, Linear algebras and its applications **192** (1993) 33–60
11. S. Coppersmith, *Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm*, Math. Comp. **62** (1994) 333–350
12. J. A. Davis, D. B. Holdridge, *Factorization using the quadratic sieve algorithm*, Tech. Report SAND 83-1346, Sandia National Laboratories, Albuquerque, NM, 1983
13. T. Denny, B. Dodson, A. K. Lenstra, M. S. Manasse, *On the factorization of RSA-120*, Advances in Cryptology, Crypto '93, Lecture Notes in Comput. Sci. **773** (1994) 166–174
14. A. Díaz, M. Hitz, E. Kaltofen, A. Lobo, T. Valente, *Process scheduling in DCS and the large sparse linear systems challenge*, J. Symbolic Computation (submitted)
15. B. Dixon, A. K. Lenstra, *Factoring integers using SIMD sieves*, Advances in Cryptology, Eurocrypt '93, Lecture Notes in Comput. Sci. **765** (1994) 28–39
16. J. D. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. **36** (1981) 255–260
17. B. Dodson, A. K. Lenstra, NFS with four large primes: an explosive experiment (in preparation)
18. M. Gardner, *Mathematical games, A new kind of cipher that would take millions of years to break*, Scientific American, August 1977, 120–124
19. J. L. Gerver, *Factoring large numbers with a quadratic sieve*, Math. Comp. **36** (1983) 287–294
20. R. Golliver, A. K. Lenstra, K. S. McCurley, *Lattice sieving and trial division*, Algorithmic number theory symposium, proceedings, Cornell, 1994 (to appear)
21. R. K. Guy, *How to factor a number*, Proc. Fifth Manitoba Conf. Numer. Math., Congressus Numerantium **16** (1976) 49–89
22. G. H. Hardy, E. M. Wright, *An introduction to the theory of numbers*, Oxford Univ. Press, Oxford, 5th ed., 1979
23. E. Kaltofen, *Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems*, Math. Comp. (to appear)
24. B. A. LaMacchia, A. M. Odlyzko, *Computation of discrete logarithms in prime fields*, Designs, Codes and Cryptography **1** (1991) 47–62
25. A. K. Lenstra, H. W. Lenstra, Jr., *Algorithms in number theory*, Chapter 12 in: J. van Leeuwen (ed.), *Handbook of theoretical computer science*, Volume A, *Algorithms and complexity*, Elsevier, Amsterdam, 1990
26. A. K. Lenstra, H. W. Lenstra, Jr. (eds), *The development of the number field sieve*, Lecture Notes in Math. **1554**, Springer-Verlag, Berlin, 1993
27. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, *The number field sieve*, 11–42 in: [26]

28. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, *The factorization of the ninth Fermat number*, Math. Comp. **61** (1993) 319–349
29. A. K. Lenstra, M. S. Manasse, *Factoring by electronic mail*, Advances in Cryptology, Eurocrypt '89, Lecture Notes in Comput. Sci. **434** (1990) 355–371
30. A. K. Lenstra, M. S. Manasse, *Factoring with two large primes*, Advances in Cryptology, Eurocrypt '90, Lecture Notes in Comput. Sci., **473** (1990) 72–82; Math. Comp. (to appear)
31. H. W. Lenstra, Jr., *Factoring integers with elliptic curves*, Ann. of Math. **126** (1987) 649–673
32. H. W. Lenstra, Jr., C. Pomerance, *A rigorous time bound for factoring integers*, J. Amer. Math. Soc. **5** (1992) 483–516
33. H. W. Lenstra, Jr., R. Tijdeman (eds), *Computational methods in number theory*, Math. Centre Tracts **154/155**, Mathematisch Centrum, Amsterdam, 1984
34. E. Messmer, *Bellcore leads team effort to crack RSA encryption code*, Network World, May 2, 1994
35. P. L. Montgomery, *Record number field sieve factorizations*, announcement on `NmbrThry@VM1.NODAK.EDU`, July 12, 1994
36. P. L. Montgomery, *A block Lanczos algorithm for finding dependencies over GF(2)*, Draft manuscript, June 17, 1994
37. M. A. Morrison, J. Brillhart, *A method of factoring and the factorization of F_7*, Math. Comp. **29** (1975) 183–205
38. R. Peralta, *A quadratic sieve on the n-dimensional hypercube*, Advances in Cryptology, Crypto '92, Lecture Notes in Comput. Sci. **740** (1993) 324–332
39. J. M. Pollard, *Theorems on factorization and primality testing*, Proc. Cambr. Philos. Soc **76** (1974) 521–528
40. J. M. Pollard, *A Monte Carlo method for factorization*, BIT **15** (1975) 331–334
41. J. M. Pollard, *Factoring with cubic integers*, 4–10 in [26]
42. C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, pp. 89–139 in: [33]
43. C. Pomerance, J. W. Smith, *Reduction of huge, sparse matrices over finite fields via created catastrophes*, Experiment. Math. **1** (1992) 89–94
44. C. Pomerance, J. W. Smith, R. Tuler, *A pipe-line architecture for factoring large integers with the quadratic sieve algorithm*, SIAM J. Comput. **17** (1988) 387–403
45. R. L. Rivest, letter to Martin Gardner, 1977
46. R. L. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM **21** (1978) 120–126
47. R. J. Schoof, *Quadratic fields and factorization*, pp 235–286 in: [33]
48. R. C. Schroeppel, personal communication, May 1994
49. P. W. Shor, *Algorithms for quantum computation: Discrete log and factoring*, DIMACS Technical Report 94-37; to appear in Proc. 35th Symposium on Foundations of Computer Science, 1994
50. R. D. Silverman, *The multiple polynomial quadratic sieve*, Math. Comp. **48** (1987) 329–339
51. J. W. Smith, S. S. Wagstaff, Jr., *An extended precision operand computer*, Proc. 21st Southeast Region ACM Conf. (1983) 209–216
52. D. H. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory **32** (1986) 54–62