Factoring Large Numbers with the TWINKLE Device
(Extended Abstract)

Adi Shamir

Computer Science Dept.
The Weizmann Institute
Rehovot 76100, Israel
shamir@wisdom.weizmann.ac.il

Abstract. The current record in factoring large RSA keys is the factorization of a 465 bit (140 digit) number achieved in February 1999 by running the Number Field Sieve on hundreds of workstations for several months. This paper describes a novel factoring apparatus which can accelerate known sieve-based factoring algorithms by several orders of magnitude. It is based on a very simple handheld optoelectronic device which can analyse 100,000,000 large integers, and determine in less than 10 milliseconds which ones factor completely over a prime base consisting of the first 200,000 prime numbers. The proposed apparatus can increase the size of factorable numbers by 100 to 200 bits, and in particular can make 512 bit RSA keys (which protect 95% of today's E-commerce on the Internet) very vulnerable.

Keywords: Cryptanalysis, Factoring, Sieving, Quadratic Sieve, Number Field Sieve, optical computing.

Ç.K. Koç and C. Paar (Eds.): CHES'99, LNCS 1717, pp. 2–12, 1999.

1 Introduction

The security of the RSA public key cryptosystem depends on the difficulty of factoring a large number n which is the product of two equal size primes p and q. This problem had been thoroughly investigated (especially over the last 25 years), and the last two breakthroughs were the invention of the Quadratic Sieve (QS) algorithm [P] in the early 1980's and the invention of the Number Field Sieve (NFS) algorithm [LLMP] in the early 1990's. The asymptotic time complexity of the QS algorithm is $O(e^{ln(n)^{1/2} ln(ln(n))^{1/2}})$, and the asymptotic time complexity of the NFS algorithm is $O(e^{1.92\ ln(n)^{1/3} ln(ln(n))^{2/3}})$. For numbers with up to about 350 bits the QS algorithm is faster due to its simplicity, but for larger numbers the NFS algorithm is faster due to its better asymptotic complexity.

The complexity of the NFS algorithm grows fairly slowly with the binary size of n. Denote the complexity of factoring a 465 bit number (which is the current record - see [R]) by X. Then the complexity of factoring numbers which are 100 bits longer is about 40X, the complexity of factoring numbers which are 150 bits longer is about 220X, and the complexity of factoring numbers which are 200 bits longer is about 1100X. Since the technique described in this paper can increase the efficiency of the NFS algorithm by two to three orders of magnitude, we expect it to increase the size of factorable numbers by 100 to 200 bits, or alternatively to make it possible to factor with a budget of one million dollars numbers which previously required hundreds of millions of dollars. The main practical significance of such an improvement is that it can make 512 bit numbers (which are the default setting of most Internet browsers in e-commerce applications, and the maximum size deemed exportable by the US government) easy to crack.

The new factoring technique is based on a novel optoelectronic device called TWINKLE. [1] Designing and constructing the first prototype of this device can cost hundreds of thousands of dollars, but the manufacturing cost of each additional device is about \$5,000. It can be combined with any sieve-based factoring algorithm, and in particular it can be used in both the QS and the NFS algoritms. It uses their basic mathematical structure and inherits their asymptotic complexity, but improves the practical efficiency of their sieving stage by a large constant factor. Since this is the most time consuming part of these algorithms, we get a major improvement in their total running time.

For the sake of simplicity, we describe in this extended abstract only the new implementation of the sieving stage in the simplest variant of the QS algorithm. Most of the new ideas apply equally well to improved variants of the QS algorithm and to the general NFS algorithm, but the details are more complicated, and will be described only in the full version of this paper.

[1] TWINKLE stands for "The Weizmann INstitute Key Locating Engine".

2 The QS Factoring Algorithm

Given the RSA number $n = pq$, the QS algorithm tries to construct two numbers y and z such that $y \neq \pm z \ (mod\ n)$ and $y^2 = z^2 \ (mod\ n)$. Knowledge of such a pair makes it easy to factor n since $gcd(y - z, n)$ is either p or q. To find such y and z, we generate a large number of values $y_1, y_2, \ldots, y_m$, compute each $y_i^2 \ (mod\ n)$, and try to factor it into a product of primes p_j from a prime base B consisting of the k smallest primes $p_1 = 2, p_2 = 3, \ldots, p_k$. Numbers $y_i^2 \ (mod\ n)$ which have such factorizations into $\prod_{j=1}^{k} p_j^{e_j}$ are called *smooth.* If the number of smooth modular squares found in such a way exceeds k, we can use Gauss elimination to find a subset of the vectors $(e_1, e_2, \ldots, e_k)$ of the prime multiplicities which is linearly dependent modulo 2. When the corresponding $y_i^2 \ (mod\ n)$ and their factorizations are multiplied, we get an equation of the form $\prod_{i=1}^{m} (y_i^2)^{b_i} = \prod_{j=1}^{k} p_j^{c_j} \ (mod\ n)$ where all the b_i's (which define the subset) are 0's and 1's and all the c_j's (which are the sums of the prime multiplicities) are even numbers. We can now get the desired equation $y^2 = z^2(\ mod\ n)$ by defining $y = \prod_{i=1}^{m} y_i^{b_i} \ (mod\ n)$ and $z = \prod_{j=1}^{k} p_j^{c_j/2} \ (mod\ n)$.

The key to the efficiency of the QS algorithm is the generation of many small modular squares whose smoothness is easy to test. Consider the simplest case in which we use the quadratic polynomial $f(x) = (a + x)^2 \ (mod\ n)$ where $a = \lfloor \sqrt(n) \rfloor$, and choose $y_i = a + i$ for $i = 1, 2, \ldots, m$. Then it is easy to see that for small m the corresponding $y_i^2 = f(i) \ (mod\ n)$ are half size modular squares which are much more likely to be smooth numbers than random modular squares.

The simplest way of testing the smoothness of the values in such a sequence is to perform trial division of each value in the sequence by each prime in the basis. Since the $f(i)$'s are hundreds of bits long, this is very slow.

The QS algorithm expresses all the generated $f(1), \ldots, f(m)$ in the non modular form $f(i) = (a + i)^2 - n$ (since m is small), and determines which of these values are divisible by p_j from the basis B by solving the quadratic modular equation $(a + i)^2 - n = 0 \ (mod\ p_j)$. This is easy, since the modulus p_j is quite small. [2]

The quadratic equation mod p_j will have either zero or two solutions d_i' and d_i". In the first case we can deduce that none of the $f(i)$'s will be divisible by p_j, and in the second case we can deduce that $f(i)$ will be divisible by p_j if and only if i belongs to the union of the two arithmetic progressions $p_j * r + d_j'$ and $p_j * r + d_j$" for $r \geq 0$.

The smoothness test in the QS algorithm is based on an array A of m counters, where the $i - th$ entry is associated with $f(i)$. The sieving algorithm zeroes all these counters, and then loops over the primes in the basis. For each prime p_j, and for each one of its two arithmetic progressions (if they exist), the algorithm scans the counter array, and adds the constant $log_2(p_j)$ to all the counters $A(i)$

[2] We ignore the issue of the divisibility of $f(i)$ by higher powers of p_j, since except for the smallest primes in the basis this is extremely unlikely, and we can explicitly add the powers of the first few primes to the basis without substantially increasing its size.

whose indices i belong to the arithmetic progression (there are about m/p_j such indices). At the end of this loop, the value of $A(i)$ describes the (approximate) binary length of the largest divisor of $f(i)$ which factors completely over the prime base B. The algorithm then scans the array, finds all the entries i for which $A(i)$ is close to the binary length of $f(i)$, tests that these $f(i)$'s are indeed smooth by trial division, and uses them in order to factor n.

Typical large scale factoring attacks with networks of PC's may use $m = 100,000,000$ and $k = 200,000$. The array requires 100 megabytes of RAM, and its counters can be accessed at the standard bus speed of 100 megahertz. [3] Just scanning such a huge array requires about one second. Well optimized implementations of the QS algorithm perform the sieving in 5 to 10 seconds, and find very few smooth numbers. They then choose a different quadratic polynomial $f'(x)$, and repeat the sieving run (on the same machine, or on a different machine working in parallel). This phase stops when a total of $k+1$ smooth modular squares are collected in all the sieving runs, and a single powerful computer performs the Gauss elimination algorithm and the actual factorization in a small fraction of the time which was devoted to the sieving.

In the next section we describe the new TWINKLE device, which is an ultrafast optical siever. It costs about the same as a powerful PC or a workstation, but can test the smoothness of 100,000,000 modular squares over a prime base of 200,000 primes in less than 0.01 seconds. This is 500 to 1000 times faster than the conventional sieving approach described above.

3 The TWINKLE Device

The TWINKLE device is a simple optoelectronic device which is housed in an opaque blackened cylinder whose diameter is about 6 inches and whose height is about 10 inches. The bottom of the cylinder consists of a large collection of LEDs (light emitting diodes) which twinkle at various frequencies, and the top of the cylinder contains a photodetector which measures the total amount of light emitted at any given moment by all the LEDs. The photodetector alerts a connected PC whenever this total exceeds a certain threshold. Such events are related to the detection of possibly smooth numbers, and their precise timing is the only output of the TWINKLE device. Since these events are extremely rare, the PC can leisurely translate the timing of each reported event to a candidate modular square, verify its smoothness via trial division, and use it in a conventional implementation of the QS or NFS algorithms in order to factor the input n.

The standard PC implementation of the sieving technique assigns modular squares to array elements (using space) and loops over the primes (using time). The TWINKLE device assigns primes to LEDs (using space) and loops over the

[3] Note that the faster cache memory is of little use, since the sieving process accesses arithmetic progressions of addresses with huge jumps, which create continuous cache misses.

modular squares (using time), which reverses their roles. This is schematically described in Fig. 1.

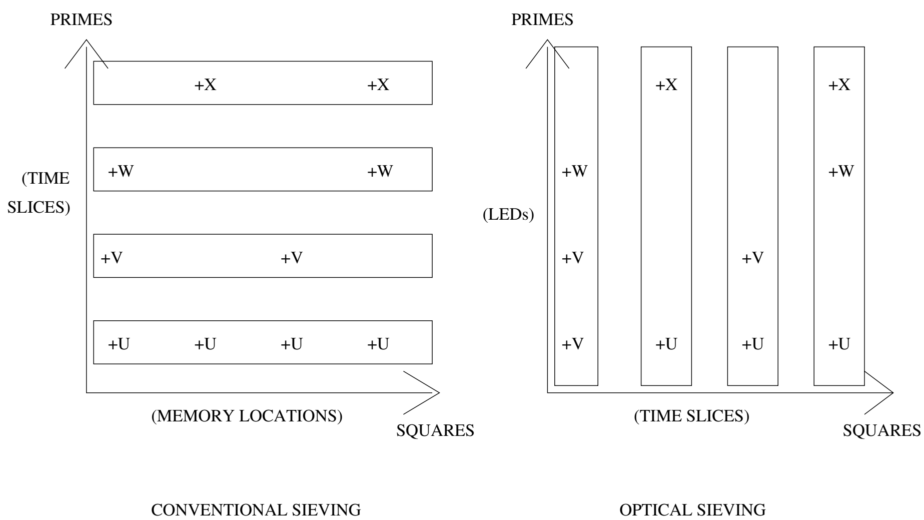

Fig. 1. Conventional vs. optical sieving: the boxed operations are carried out at the same time slice

Each LED is associated with some period p_j and delay d_j, and its only role is to light up for one clock cycle at times described by the arithmetic progression $p_j * r + d_j$ for $r \geq 0$. To mimic the QS sieving procedure, we have to use nonuniform LED intensities. In particular, we want the LED associated with prime p_j to generate light intensity proportional to $log_2(p_j)$ whenever it flashes, so that the total intensity measured by the photodetector at time i will correspond to the binary size of the largest smooth divisor of the $f(i)$ [4] We can achieve this by using an array of LEDs of different sizes or with different resistances. However, a simpler and more elegant solution to the problem is to construct a uniform array of identical LEDs, to assign similar sized primes to neighbouring LEDs, and to cover the LED array with a transparent filter with smoothly changing grey levels. [5] Note that the dynamic range of grey levels we have to use is quite limited, since the ratio of the logs of the largest and the smallest primes in a typical basis does not exceed 24:1.

To increase the sensitivity of the photodetector, we can place it behind a large lense which concentrates all the incoming light on its small surface area. The light

[4] Again, we ignore the issue of the divisibility of $f(i)$ by higher powers of the primes.
[5] For example, we can assign primes to LEDs in row major order and use a filter which is dark grey at the top and completely transparent at the bottom, or assign primes to LEDS in spiral order and use a filter which is darkest at its center.

intensity measurement is likely to be influenced by many sources of errors. For example, the grey levels of the filter are only approximations of the logs, and even uniformly designed LEDs may have actual intensities varying by 20% or more. We can improve the accuracy of the TWINKLE device by measuring the actual filtered intensity of each LED in the manufactured array, and assigning the sequence of primes to the various LEDs based on their sorted list of measured intensities. However, the QS and NFS factoring algorithms are very forgiving to such measurement errors, and in PC implementations they use crude integer approximations to the logs in order to speed up the computation. There are two possible types of errors: missed events and false alarms. To minimize the number of missed events we can set a slightly lower reporting threshold, and to eliminate the resultant false alarms we can later test all the reported events on a PC, in order to find the extremely rare real events among the rare false alarms. For typical values of the parameters, the average binary size of the smooth part of candidate values is about one tenth of their size, and only a tiny fraction of all candidate values have ratios exceeding one half. As a result, the desired events stand out very clearly as isolated narrow peaks which are about ten times higher than the background noise.

We claim that optical sieving is much better than conventional counter array sieving for the following reasons:

1. We can perform optical sieving at an extremely fast clock rate. Typical silicon RAM chips in standard PC's operate at about 100 megahertz. LEDs, on the other hand, are manufactured with a much faster Gallium Arsenide (GaAs) technology, and can be clocked at rates exceeding 10 gigahertz without difficulty. Commercially available LEDs and photodetectors are used to send 10 gigabits per second along fiber optic cables, and GaAs devices are widely used at similar clock rates as routers in high speed networks.
2. We can instantaneously add hundreds of thousands of optical contributions, if we do not need perfect accuracy. Building a digital adder with 200,000 inputs which computes their sum in a single clock cycle is completely unrealistic.
3. The optical technique does not need huge arrays of counters. Instead of using one memory cell per sieved value, we use one time slice per sieved value. Even with the declining cost of fast memories, time is cheaper than space.
4. In the optical technique do not have to scan the array at the beginning in order to zero it, and do not have to scan the array again at the end in order to find its high entries - both operations are done at no extra cost during the actual sieving.

In the remaining sections we flesh out the design of each cell and the architecture of the whole device. We based this design on many conversations with experienced GaAs chip designers, and used only commercially available technologies. We may be off by a small factor in some of our size speed and cost estimates, but we believe that the design is realistic, and that someone will try it out in the near future.

4 Cell Design

The LED array is implemented on a single wafer of GaAs. Each cell on this wafer contains one LED plus some circuitry which makes it flash for exactly one clock cycle every exactly p_j clock cycles with an initial delay of exactly d_j clock cycles. The high clock rate and extremely accurate timing requirements rule out analog control methods, and the unavoidable existence of bad cells in the wafer rules out a prewired assignment of primes to cells. Instead, we use identical cells throughout the wafer, and include in each cell two registers, A and B, which are loaded before the beginning of the sieving process with values corresponding to p_j and d_j, respectively. For a typical sieving run over $m = 100,000,000$ values, we need only $log_2(m) \approx 27$ bits in each one of these registers.

The structure of each cell (described in Fig. 2) is very simple. Instead of using counters (with their more complicated designs and additional carry-induced delays), we use register B as a maximal length shift register based on a single XOR of two of its bits. It is driven by the clock, and runs until it enters the special state in which all its bits are "1". When this is recognized by the AND of all the bits of register B, the LED flashes, and register B is reloaded with the contents of register A (which remains unchanged throughout the computation). The initial values loaded into registers A and B are not the binary representations of p_j and d_j, but the (easily computed) states of the shift register which are that many steps before the special state of all "1". That's the whole cell design!

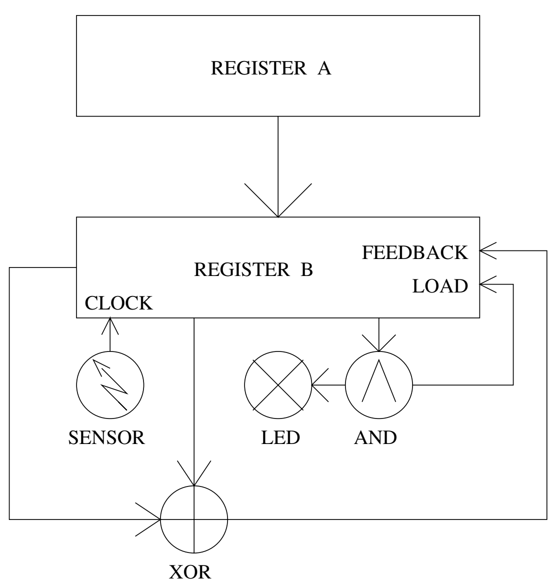

Fig. 2. A single cell in the array

An important issue in such a high speed device is clock synchronization. Each clock cycle lasts only 100 picoseconds, and all the light pulses must be synchronized to within a fraction of this interval in order to correctly sum their contributions. Distributing electrical clock pulses (which travel slowly over long, high capacity wires) at 10 gigahertz to thousands of cells all over the wafer without skewing their arrival times by more than 10-20 picoseconds seems to be a very difficult problem. We solve it by using another optical trick. Since it is easy to construct in GaAs technology a small photodetector in each cell, we use optical rather than electrical clock distribution: a strong LED placed opposite the wafer, which flashes at a fixed rate of 10 gigahertz, and its pulses are almost simultaneously picked up by the photodetectors in all the cells, and used to drive the shift registers in a synchronized way. Since light passes about 3 centimeters in 100 picoseconds, we just have to place the clocking LED and the summing photodetector sufficiently far away from the wafer to guarantee sufficiently similar optical delays to and from all the cells on the flat wafer. To avoid possible confusion between clock and data light pulses, we can use two different wavelengths for the two purposes.

Computing the AND of 27 inputs requires a tree of depth 3 of 3-input AND gates, which may be the slowest cell operation. To speed it up, we can use a systolic design which carries out the tree computation in 3 consecutive clock cycles. This delays the detection of the special state by 3 clock cycles but keeps all the flashing LEDs perfectly synchronized. To compensate for the late reloading of register B, we simply store a modified value of p_j in register A.

An improved cell design is based on the observation that about half the primes do not yield arithmetic progressions, whereas each prime in the other half yields two arithmetic progressions with the same period p_j. In standard PC implementations this has little effect, since we still have to scan on average one arithmetic progression per prime in the basis. However, in the TWINKLE design the two cells assigned to the same p_j can share the same A register (which never changes) to reload their separate B shift registers. In addition, the two cells can share the same LED and flash it with the OR of the two AND gates, since the two arithmetic progressions are always disjoint. We call such a combination a double cell, and use it to reduce the average number of registers per prime in the basis from 2 to 1.5. Since these registers occupy most of the area of the cell, this observation can increase the number of primes we can handle with a single wafer by almost 33%.

5 Wafer Design

We would like to justify our claim that a single wafer can handle a prime base of 200,000 primes (which is the actual size used in recent PC-based factorizations). A standard 6 inch wafer has a total usable area of about $16 * 10^9$ square microns. Commercially available LED arrays (such as the arrays sold by Oki Semiconductors to manufacturers of laser printers - see http://www.oki.co.jp/OKI/home/English/New/OKI-News/1998/z9819e.html for further details) have a linear density of 1200 LEDs per inch. At this density, each LED occupies a $20\mu \times 20\mu$ square with an area of $400\mu^2$, and we can fit about 40,000,000 LEDs on a single wafer. However, most of area of each double cell will be devoted to the three 27 bit registers. Crude conservative estimates indicate that we can very comfortably fit each one of these 81 bits into an area of $1,600\mu^2$ using commercially available GaAs technology. We can thus fit the whole double cell into an area of less than $160,000\mu^2$, and pack 100,000 double cells into a single wafer. Such a wafer will be able to sieve numbers over a prime base of 200,000 primes.

A simple reality check is based on the computation of the total amount of memory on the wafer. The 100,000 double cells contain $81 \times 100,000$ bits, or about one megabyte of memory. The other gates (XOR, AND) and diodes (LEDs, photodetectors) occupy a small additional area. This is a very modest goal for wafer scale designs.

The cost of manufacturing silicon wafers in a commercial FAB is about $1,500 per wafer, and the cost of manufacturing the more expensive GaAs wafers is about $5,000 per wafer (excluding design costs and assuming a reasonably large order of wafers). This is comparable to the cost of a strong workstation, but provides a sieving efficiency which is several hundred times higher.

The TWINKLE device does not have a yield problem, which plagues many other wafer-scale designs: During the sieving process each cell works completely independently, without receiving any inputs or sending any outputs to neighbouring cells. Even if 20% of the cells are found to be defective in postproduction inspection, we can use the remaining 80% of the cells. If necessary, we can place two or more wafers at the same distance opposite the same summing detector, in order to compensate for defective cells or to sieve over larger prime bases.

After determining the number of cells, we can consider the issue (which was ignored so far) of loading registers A and B in each cell with some precomputed data from a connected storage device. Silicon memory cannot operate at 10 gigahertz, and thus we have to slow down the clocking LEDs facing the GaAs wafer during the loading phase. The A registers which contain the primes assigned to each LED can be loaded only once after each powerup, but the B registers which contain the initial delays have to be loaded for each sieving run. The total size of the 200,000 B registers is about 675 kilobytes. Such a small amount of data can be kept in a standard type of silicon memory, and transfered to the wafer in 0.002 seconds on a 27 bit bus operating at 100 megahertz. This is one fifth the time required to carry out the actual sieving at the full 10 gigahertz clock rate, and thus it does not create a new speed bottleneck.

The proposed wafer design has just 31 external connections: Two for power, two for control, and 27 for the input bus. The four modes of operation induced by the two control wires consist of a test mode (in which the various LEDs are sequentially flashed to test their functionality and measure their light intensity), LOAD-A mode (in which the various A registers are sequentially loaded from the bus), LOAD-B mode (in which the various B registers are sequentially loaded from the bus), and sieving mode (in which all the shift registers are simultaneously clocked at 10 gigahertz). We can briefly freeze the optical clocking during mode changes in order to enable the slow electric control signals to propagate to all the cells on the wafer before we start operating in the new mode.

Another important factor in the wafer design is its total power consumption. Strong LEDs consume considerable amounts of power, and if a large number of LEDs on the wafer flash simultaneously, the excessive power consumption can skew the intensity of the flashes. However, each tested number can be divisible by at most several hundred primes fron the basis, and thus we have a small upper bound on the total power which can be consumed by all the LEDs at any given moment in the sieving process.

6 The Geometry of the TWINKLE Device

The TWINKLE device is housed in an opaque cylinder with the wafer at the bottom and the summing photodetector and clocking LED at the top. Its diameter is determined by the size of the wafer, which is about 6 inches. Its height is determined by the uniformity requirements of the length of the various optical paths.

To determine this height, we recall that light travels about 3 centimeters in a single clock cycle which lasts 100 picoseconds. To make sure that all the received light pulses are synchronized to within 15% of this duration, we want the length of the optical paths from the clocking LED to any point in the wafer and from there to the summing photodetector to vary by at most 0.5 centimeter. The simplest arrangement places both elements at the center of the top face of the cylinder, but this penalizes twice LEDs located at the rim compared to LEDs located at the center, and requires a cylinder whose length is about 110 centimeters. A better arrangement uses several clocking LEDs placed symmetrically around the rim of the top face, and a single photodetector at the center of this face. A simple geometric calculation shows that the required uniformity will be attained in a cylinder which is just 25 centimeters (10 inches) long.

7 Concluding Remarks

The idea of using physical devices in number theoretic computations is not new. D. H. Lehmer managed to factor (relatively small) numbers and solve other diophantine equations by pedalling on a device based on toothed wheels and bicycle chains of various lengths (a replica of this ingenious contraption from the 1920's is located at the Boston Computer Museum). His device even included a photodetector to alert the rider when the solution was found, but its mode of operation was of course completely different from our implementation of the quadratic sieve.

The TWINKLE device proposed in this paper demonstrates the incredible speed and almost unbounded parallelism which is offered by today's optoelectronic techniques. We believe that they will find many additional applications in cryptography and cryptanalysis.

Acknowledgements: I would like to thank Moty Heiblum and Vladimir Umanski for many useful discussions of GaAs technology.

References

[LLMP] A. K. Lenstra, H. W. Lenstra, M. S. Manasse, and J. M. Pollard, *The number field sieve*, Vol. 1554 of Lecture Notes in Mathematics, 11-42, Springer Verlag, 1993.

[P] C. Pomerance, *The quadratic sieve factoring algorithm*, Proceedings of EUROCRYPT 84 (LNCS 209), 169-182, 1985.

[R] Hermann J. J. te Riele, email announcement, February 4 1999, available at http://jya.com/rsa140.htm.