Analysis and Optimization of the TWINKLE Factoring Device

Arjen K. Lenstra[1] and Adi Shamir[2]

[1] Citibank, N.A., 1 North Gate Road, Mendham, NJ 07945-3104, U.S.A.
arjen.lenstra@citicorp.com

[2] Computer Science Department, The Weizmann Institute, Rehovot 76100, Israel
shamir@wisdom.weizmann.ac.il

Abstract. We describe an enhanced version of the TWINKLE factoring device and analyse to what extent it can be expected to speed up the sieving step of the Quadratic Sieve and Number Field Sieve factoring algorithms. The bottom line of our analysis is that the TWINKLE-assisted factorization of 768-bit numbers is difficult but doable in about 9 months (including the sieving and matrix parts) by a large organization which can use 80,000 standard Pentium II PC's and 5,000 TWINKLE devices.

1 Introduction

The TWINKLE device is an optoelectronic device which is designed to speed up the sieving operation in the Quadratic Sieve (QS) and Number Field Sieve (NFS) integer factoring algorithms by using arrays of light emitting diodes (LED's) which blink at various rates (cf. [7]). The main purpose of this paper is to carry out a detailed and realistic analysis of the expected behavior of a TWINKLE-assisted factoring attempt on inputs whose binary sizes are 384, 512, and 768 bits. In particular, we describe the optimal choice of the many parameters involved in such factoring attempts, and identify several areas in which the original TWINKLE design leads to computational bottlenecks. We then propose enhanced hardware and algorithmic designs which eliminate these bottlenecks, and make such factorizations more feasible.

This paper is organized as follows. In Section 2 we briefly review the original TWINKLE design from [7]. In Section 3 we discuss the applicability of the original TWINKLE design to 384-bit numbers using the QS algorithm. In the remainder of the paper we concentrate on how TWINKLE may be used for the sieving step of the NFS for the factorization of 512-bit and 768-bit numbers. In Section 4 we briefly sketch the required NFS background, and in Section 5 we discuss the sieving step of the NFS in more detail. In Section 6 we present a number of hardware enhancements of the TWINKLE device. In Section 7 we describe how the NFS sieving step may be carried out on the modified TWINKLE device and we analyse its running time. In Section 8 we address the question what it is about TWINKLE that makes LED's necessary and comment upon the proposals to build a TWINKLE-like device using ordinary electronic circuitry.

© Springer-Verlag Berlin Heidelberg 2000

All PC running times referred to in this paper are actual measurements rather than asymptotic or conjectured running times. They are based on optimized implementations of the various algorithms on a 450MHz Pentium II PC with 192 megabytes of RAM. However, the TWINKLE device itself has not been built so far, and thus its feasibility, performance, and cost are open to debate.

2 The Original TWINKLE Design

We recall the basics of the TWINKLE device as described in [7]. The most time consuming step of most modern factoring algorithm is the 'sieving step': for many pairs of integers (p, r) in succession add an approximation of $\log(p)$ to location $r + kp$ of a sieve interval (initialized as zeros) for all integers k such that $r + kp$ is in the interval, and report all j for which location j exceeds a certain threshold. Standard implementations use space (memory) to represent the interval, and time (clock cycles) to loop over the (p, r) pairs. The TWINKLE device reverses the roles of space and time: it uses space to represent the (p, r) pairs, and time to loop over the sieve interval. This goes as follows.

The TWINKLE device is a cylinder of 6 inch diameter and 10 inch height. The bottom consists of a single wafer of GaAs, containing one 'cell' for each different p. Each cell contains an LED, a photodetector, an A register representing the value of p, for each r corresponding to that p a B register initially loaded with a representation of r, and wiring. The top of the cylinder contains a summing photodetector that measures the total light intensity emitted by the bottom LED's and a clocking LED that distributes the clock signal by flashing at a fixed clock rate. As clock signal j is received by a cell's photodetector, the values of the B registers are decremented by one, if a resulting value represents zero the cell's LED flashes with intensity proportional to $\log(p)$, and the A register is copied to the B register representing zero. If the total light intensity detected by the top photodetector exceeds a certain threshold the value of j is reported. Further details of the original TWINKLE design, such as how integers are represented and decremented and how the optical delays are handled, can be found in [7].

The bottom wafer as proposed in [7] contains 10^5 cells with one A and two B registers each, and is clocked at 10 GHz. Since it takes a single clock cycle to sum the values corresponding to a sieve location and to report the location if necessary, this would correspond to a QS implementation that takes 10 milliseconds to inspect 10^8 integers for smoothness with respect to the primes up to $3 * 10^6$.

3 Analysis of TWINKLE-Assisted 384-Bit QS Factorizations

The original description of the TWINKLE device in [7] is geared towards the QS factoring algorithm (cf. [6]). In this section we analyse the effectiveness of the original TWINKLE device for the factorization of 384-bit numbers using the QS.

Although 384-bit numbers are unlikely to be the moduli in RSA cryptosystems, their quick factorization may be useful in various number theoretic subroutines (e.g., when we try to complete the factorization of a large number of randomly generated values after we eliminate their smooth parts). The factorization of such numbers is not particularly difficult - it can be carried out in a few months on a single PC. Our goal is simply to find out the improvement ratio between TWINKLE-assisted factorizations and PC-based factorizations. There are two reasons why such an analysis can be interesting:

– There is a great deal of experimental data on the optimal choice of parameters and the actual running time when factoring numbers of this size, and thus the comparison can be based on harder data.
– It enables us to examine the specific issues related to the implementation of the QS algorithm on the TWINKLE device. The QS and NFS algorithms exhibit quite different properties when implemented on the TWINKLE device, but for larger input sizes the QS algorithm is simply not competitive with the NFS algorithm.

We assume that the reader is familiar with the general outline of the QS algorithm (see, e.g. [7]). A sequence of quadratic polynomials $f_1, f_2, \ldots$ is generated that depends on the number to be factored and the length $2 * \mathbf{A}$ of the sieve interval. For $i = 1, 2, \ldots$ in succession the roots of the f_i modulo the primes in the factor base are computed and the values $f_i(x)$ for $-\mathbf{A} \leq x < \mathbf{A}$ are sieved to test them for smoothness. The resulting smooth values have to be post-processed, which consists of trial division possibly followed by the computation of the decomposition of the resulting cofactor. For the QS algorithm the post-processing step is negligible compared to the polynomial generation and root computation. When doing the actual sieving on the TWINKLE device, polynomial generation, root finding, and post-processing have to be carried out by one or more auxiliary PC's. The sequence of polynomials can be generated using the ordinary Multiple Polynomial variant (MPQS) or using the Self Initializing variant (SIQS).

Based on actual data, the factorization of a 384-bit number with the QS algorithm on a single 450 MHz Pentium II requires:

– About 9 months when running the QS algorithm with optimal parameters: 186,000 primes in the factor base and $2 * \mathbf{A}$ ≈1,600,000 (SIQS) or $2 * \mathbf{A}$ ≈16,000,000 (MPQS).
– About 14 months when running the QS algorithm with the suboptimal choices used in the original TWINKLE design (cf. [7]): 100,000 primes in the factor base and a sieving interval of length $2 * \mathbf{A}$ = 100,000,000.

For 384-bit inputs, there is little difference between the running times of MPQS and SIQS, but SIQS can compute the roots of the f_i faster, which is a significant advantage in TWINKLE-assisted factorizations. For the optimal choice of parameters, a PC implementation spends about 25% of the time on polynomial selection and root finding, but for the original choice (which we shall assume from now on) this fraction drops to 0.6% (about 0.09 seconds per polynomial on a PC). We consider two possible scenarios:

1. A TWINKLE device running at the maximum possible speed of 10 GHz. Each sieving interval of length 100,000,000 can be scanned in 0.01 seconds (cf. [7]). The total running time of the TWINKLE device is about 11 hours, and 9 (= 0.09/0.01) PC's are needed to generate the polynomials and to compute their roots. These 9 PC's can execute a conventional QS factorization with optimal parameters in about a month, and thus the achievable improvement ratio is approximately $30 * 24/11 \approx 65$.
2. A TWINKLE device running at the minimum recommended speed of 1 GHz (cf. 6.1). Scanning a single interval takes 0.1 seconds, and the whole scanning phase takes 110 hours or about 4.5 days. However, in this case we need only one PC to support the TWINKLE device. Thus we have to compare this execution time to the 9 months required by a single PC implementation of QS with optimal parameters. The relevant improvement ratio is thus $9 * 30 * 24/110 \approx 59$.

The surprising conclusion is that we get about the same improvement ratio regardless of whether we run the TWINKLE device at 10 GHz or at 1 GHz, since the computational bottleneck is in the supporting PC's. As described in 6.1, a 1 GHz TWINKLE is much easier to design and operate, and can make the whole idea much more practical.

The improvement ratio of about 60 refers only to application of the QS because a 384-bit number can be factored in about 2 months on a PC using the NFS (this figure is based on extrapolation of the results from [1]).

We next consider the problem of factoring 512-bit numbers, which are typical RSA keys in E-commerce applications. For this size the QS is not competitive with the asymptotically faster NFS so we concentrate on the NFS in the remainder of this article.

4 Number Field Sieve

The Number Field Sieve integer factorization algorithm consists of four main steps:

- Polynomial selection;
- Sieving;
- Matrix processing;
- Algebraic square root computation.

We briefly describe these steps as far as relevant for the description of the TWINKLE device. Let n be the number to be factored. For ease of exposition we assume that n is a 512-bit number.

4.1 Polynomial Selection

In the first step of the NFS factorization of n two polynomials of degrees 5 and 1 with a common root modulo n are selected:

$$f_1(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \in \mathbf{Z}[x]$$

and

$$f_2(x) = x - m \in \mathbf{Z}[x] \ ,$$

where $f_1(m) \equiv 0 \bmod n$. It is advantageous if the a_i and m are small in absolute value and if f_1 has relatively many roots modulo small primes. The best known method to find such polynomials (cf. [5]) produces an m that is of the same order of magnitude as $n^{1/6}$ and a polynomial f_1 that is *skew*, i.e., $|a_5| \ll |a_4| \ll |a_3| \ll |a_2| \ll |a_1| \ll |a_0|$. The *skewness ratio* s of f_1 approximates the average ratio $|a_i|/|a_{i+1}|$. A realistic value for s is 10^4. The bivariate, homogeneous, integral polynomials F_1 and F_2 are defined as

$$F_1(x, y) = y^5 * f_1(x/y) \text{ and } F_2(x, y) = y * f_2(x/y) \ .$$

Everything related to f_1 or F_1 is referred to as the *algebraic side*, as opposed to the *rational side* for f_2 or F_2.

4.2 Sieving

In the second step, the sieving step, *relations* are sought. These are coprime pairs of integers (a, b) such that $b > 0$ and both $F_1(a, b)$ and $F_2(a, b)$ are smooth, where smoothness of $F_1(a, b)$ and $F_2(a, b)$ is defined as follows:

– $F_1(a, b)$ factors over the primes $< 2^{24}$, except for possibly three primes $< 10^9$;
– $F_2(a, b)$ factors over the primes $< 2^{24}$, except for possibly two primes $< 10^9$.

Thus, three large primes are allowed on the algebraic side, but only two large primes are allowed on the rational side. There are about one million primes $< 2^{24}$, more precisely $\pi(2^{24})$ =1,077,871.

Candidate pairs (a, b) are located by twice sieving with the primes $< 2^{24}$ over the rectangular region $-\mathbf{A} \leq a < \mathbf{A}, 0 < b \leq \mathbf{A}/s$, for a large $\mathbf{A}$ that is specified in 5.5. The sieving region is skew with skewness ratio s. The resulting candidate pairs are trial divided to inspect if they indeed lead to relations. How the sieving and the trial division may be carried out is addressed in the next section. It is the most time consuming step of the NFS, and it is the step for which we want to use the TWINKLE device.

4.3 Matrix Processing

Each relation a, b gives rise to a vector of exponents corresponding to the multiplicities of the primes in the factorizations of $F_1(a, b)$ and $F_2(a, b)$. It may be expected that there are many linear dependencies among these vectors after about 80 to 100 million relations have been found. Dependencies modulo 2 among the vectors are determined in the matrix processing step. How the matrix step is carried out is described in the literature referred to in [1].

4.4 Algebraic Square Root Computation

Each dependency modulo 2 leads with probability at least one half to the factorization of n in the last step of the NFS. References describing how this is done can be found in [1].

5 Sieving

So far two distinct sieving methods have been used in NFS implementations: *line sieving* and *special q sieving*. Line sieving works by sieving over the a-interval $[-\mathbf{A}, \mathbf{A})$ for each $b = 1, 2, 3, \ldots$ consecutively, until enough relations have been found. Special q sieving works by repeatedly picking an appropriate prime q between 2^{24} and, approximately, $5 * 10^8$, and by restricting the sieving to the pairs (a, b) for which q divides $F_1(a, b)$, until enough unique relations have been found. Note that a relation is found at most once by line sieving but that it may be found up to three times by special q sieving because each of the at most three algebraic large primes may be used as special q. Both sieving methods may be used, simultaneously or separately, for a single factorization. We describe both methods in more detail, paying special attention to a property of special q sieving that is not generally appreciated and that turns out to be beneficial for TWINKLE-assisted NFS sieving.

5.1 Factor Bases and Sieving Thresholds

Let for $i = 1, 2$

$$P_i = \{(p, r) : f_i(r) \equiv 0 \bmod p, \ \ p \text{ prime}, \quad p < 2^{24}, \ \ 0 \leq r < p\} \ .$$

The set P_2, the *rational factor base*, consists of the pairs $(p, m \bmod p)$ for all primes $p < 2^{24}$ and is trivial to compute. For the computation of P_1, the *algebraic factor base*, the roots of $f_1 \bmod p$ have to be determined for all primes $p < 2^{24}$. The number of times a particular prime $p < 2^{24}$ occurs in a (p, r) pair in P_1 may be 0, 1, 2, 3, 4, or 5. The sets P_1 and P_2 are computed once. Let T_1 and T_2 be two threshold values.

5.2 Line Sieving

For $b = 1, 2, 3, \ldots$ in succession do the following.

> For $i = 1, 2$ in succession, initialize the sieve locations $S_i(a)$ to zero for $-\mathbf{A} \leq a < \mathbf{A}$, and next for all (p, r) in P_i replace $S_i(br + kp)$ by $S_i(br + kp) + \log(p)$ for all integers k such that $br + kp \in [-\mathbf{A}, \mathbf{A})$. Finally, for all a such that $\gcd(a, b) = 1$, $S_1(a) > T_1$, and $S_2(a) > T_2$ inspect if both $F_1(a, b)$ and $F_2(a, b)$ are smooth.

For a fixed b-value the a-interval $-\mathbf{A} \leq a < \mathbf{A}$ is referred to as a *line*. Note that line sieving uses $\#P_1 + \#P_2$ arithmetic progressions per line, i.e., per b-value.

5.3 Special Sieving

For pairs (q, r_q) with $f_1(r_q) \equiv 0 \bmod q$, q prime, and $2^{24} < q < 10^9$ do the following.

> Let L_q be the lattice spanned by the two 2-dimensional vectors $(q, 0)^T$ and $(r_q, 1)^T$. Sieving over $L_q \cap \{(a, b)^T : -\mathbf{A} \le a < \mathbf{A}, 0 < b \le \mathbf{A}/s\}$, i.e., 'small' (a, b) pairs for which q divides $F_1(a, b)$, is approximated as follows. Find a basis $(x_1, y_1)^T$, $(x_2, y_2)^T$ for L_q for which $|x_i| \approx |s * y_i|$. Let V_q be the subset of L_q consisting of the vectors $d * (x_1, y_1)^T + e * (x_2, y_2)^T$ for integers d, e with $-\mathbf{A}/(s*q)^{1/2} \le d < \mathbf{A}/(s*q)^{1/2}$, $0 < e \le \mathbf{A}/(s*q)^{1/2}$ (although in practice one sieves over the same d, e values for all pairs (q, r_q)). For $i = 1, 2$ in succession, initialize the sieve locations $S_i(v)$ to zero for all v in V_q, and next for all (p, r) in P_i replace $S_i(v)$ by $S_i(v) + \log(p)$ for all v in V_q that can be written as an integer linear combination of $(p, 0)^T$ and $(r, 1)^T$. Finally, for all $v = (a, b)^T$ in V_q for which $\gcd(a, b) = 1$, $S_1(v) > T_1 - \log(q)$, and $S_2(v) > T_2$ inspect if both $F_1(a, b)$ and $F_2(a, b)$ are smooth.

In this case a line is the d-interval for a fixed e-value. It follows from the asymptotic values of $\mathbf{A}$ and p (cf. [3]) that a particular line (e-value) is not hit (in the d-interval) by the majority of pairs (p, r). Using arithmetic progressions for all (p, r) pairs per e-value would therefore increase the asymptotic running time of NFS, i.e., it is too expensive to visit all $\mathbf{A}/(s*q)^{1/2}$ e-values for all $(p, r) \in P_1 \cup P_2$. Instead, per (p, r) only $O(\mathbf{A}^2/(s * q * p))$ steps may be performed for the sieving over V_q. In [2] this problem is adequately solved by *lattice sieving* for each (p, r) pair, as proposed by Pollard (cf. his second article in [3]). Although the TWINKLE device may appear to solve the problem by processing all (p, r) pairs simultaneously for each sieve location, per line the initial B registers still have to be loaded for each (p, r) pair, which is obviously too expensive. This problem and its consequences are discussed in more detail in 7.2.

5.4 Trial Divisions

For reasonable choices of T_1 and T_2 the number of pairs (a, b) for which $F_1(a, b)$ and $F_2(a, b)$ have to be inspected for smoothness is so large that straightforward trial division with the primes in P_1 and P_2 would take considerably more time than the sieving. Trial division is therefore too time consuming. Instead, in PC implementations the primes dividing $F_1(a, b)$ and $F_2(a, b)$ are found by first repeating the sieving step in a somewhat altered fashion, next performing divisions of $F_1(a, b)$ and $F_2(a, b)$ by the primes found by this resieving, and finally factoring the resulting cofactors if they are composite and sufficiently small. For line sieving the F_1-cofactor may have to be factored into three factors, whereas for special q sieving two factors suffice. Cofactor factorization is thus substantially easier for special q sieving than for line sieving. For the line sieving in [1] this problem was avoided by using a substantially larger algebraic factor base and by allowing only two additional (large) primes in the factorization of $F_1(a, b)$.

This is illustrated by the following actual timings. For smoothness as defined here, the divisions and cofactor factorizations for line sieving cost about 0.7 seconds on a PC per resulting relation, including the time spent on the false reports but not including the time spent on the resieving. For special q sieving it goes more than 6 times faster, i.e., about 0.1 seconds per resulting relation, due to the easier factorizations and considerably smaller number of false reports. Between 80 to 100 million relations are needed (cf. 4.3), so that with line sieving only one may expect to spend more than two years on a single PC to process the reports resulting from sieving and resieving. For special q sieving this can be reduced to about 5 months.

Therefore, a line sieving TWINKLE device needs to be supported by about $7 * 10^7$ seconds on a PC to perform the divisions by the primes found by the resieving plus the cofactor factorizations. That is about 12.5% of the total PC sieving time reported in [1]. For a special q sieving TWINKLE device other considerations come into play, as shown in 7.2.

5.5 Sieving Regions

For a 512-bit n sufficiently many good pairs (a, b) can be expected if special q sieving is done for $-2^{12} \leq d < 2^{12}$ and $0 < e \leq$5,000, for all (q, r_q) pairs with $2^{24} < q < 5*10^8$. A gross over-estimate for the required line sieving effort follows by taking $q = 5 * 10^8$ and $\mathbf{A} = 2^{12} * (s * q)^{1/2}$. Thus $\mathbf{A} = 9 * 10^9$, i.e., $9 * 10^5$ lines of length $1.8 * 10^{10}$ each, should suffice for line sieving. The number of points sieved would be about $17 * 10^{15}$ for line sieving, but only about 10^{15} for special q sieving (where we use that $\pi(5 * 10^8) =$ 26,355,867). PC implementations of the NFS exclude the 'even, even' locations from the sieve, so on PC's the numbers of points sieved are 25% lower.

6 Hardware Enhancements

In this section we address most of the potential problems in the original TWINKLE paper which were pointed out by hardware designers and factoring experts. The result is a simpler, better, and more practical factoring device. More particularly, we address the following issues:

1. Clock rate;
2. Power consumption;
3. Avoiding trial division;
4. Using separate algebraic and rational LED's;
5. Geometric considerations.

The type of sieving is left unspecified (cf. Section 7). We assume familiarity with the design from [7] as reviewed in Section 2: cells with A registers for the primes, B registers for the counters, a photodetector for the clock signal, and LED's for the flashing.

6.1 Clock Rate

In [7] Shamir assumed that a TWINKLE device can be designed to run at 10 GHz. This speed represents the limit to which currently available GaAs technology can be pushed. However, it is not clear that our ability to produce at great cost a single laser diode which can switch at such speeds can be duplicated in mass produced wafer scale designs. Such a clock rate should thus be viewed as ambitious and speculative, but not as revolutionary as the construction of a quantum factoring computer.

On the other hand, devices made with the slower CMOS technology already run at clock rates exceeding 700 MHz. We can thus reasonably assume that a TWINKLE device built today can run at clock rates exceeding 1 GHz, and that a TWINKLE device built 5 to 10 years from now can run at clock rates approaching 10 GHz. However, as demonstrated in Section 3 the speed issue can be irrelevant since the achievable speedup ratio can be independent of the actual speed of the TWINKLE device.

6.2 Power Consumption

Several experienced hardware designers objected to the original TWINKLE design, claiming that it would consume too much power, which could lead to a wafer meltdown. Since the power consumption grows linearly with the clock rate, a 10-fold reduction of the recommended clock rate can greatly reduce this problem.

Even greater power reduction can be obtained by using a different cell design. The total power consumption of all the LED's is negligible, since at most a few hundred out of the 100,000 LED's can flash at any given time. The total power consumption of all A registers is also negligible, since they change their state only once per sieving interval. Almost all the power consumed by the wafer is used to change the state of the bits in the B registers which count the number of clock cycles. The original TWINKLE design implemented the counters as linear feedback shift registers. Such a counter design eliminates the carry propagation problem and makes the flashes highly synchronized, but it consumes a lot of power since each bit in the counter changes state every second clock cycle on average.

To reduce the power consumption, we now propose a different design. It is based on an asynchronous ripple counter in which the clock signal is fed only to the least significant bit, and the ith bit changes state only once every 2^i clock cycles. As a result, most of the bits in the counter can operate at slow speed, and the average power consumption is a small constant which is independent of the length of the counter.

The LED can be flashed when the most significant bit changes state from 0 to 1. This eliminates the tree of AND's in the original design, but it can take a long time (several clock cycles) for the clock to ripple through the register when state "0111...111" changes to "1000...000". A 10% difference in the switching speeds of two counters can lead to flashes which are almost a full clock cycle apart, leading to incorrect results. A simple solution to this problem is based on the observation that the timing of the least significant bits is likely to be much more precise than the timing of the most significant bits. Assume that the maximum propagation delay across the register is between 0 and 15 clock cycles. We derive the flashing signal by AND'ing the most significant bit and the fifth least significant bit. Regardless of when the former bit turns to "1", the flash will occur when the latter is turned to "1". Since we reload the B register (in parallel) shortly afterwards, this AND condition will not reoccur until the end of the next cycle.

6.3 Avoiding Trial Division

The analog nature of the TWINKLE device implies that each reported smoothness event has to be confirmed and turned into an actual vector of prime exponents. The original TWINKLE design assumed that such events will be so rare that the host PC will use trial division with 100,000 possible primes to accomplish this. For the QS algorithm this assumption is correct, as mentioned in Section 3. However, as mentioned in 5.4 this is a potential bottleneck for the NFS.

In this section we describe a small modification of the TWINKLE design which can greatly simplify this task. The basic idea is to use the optical photodetector in order to detect that a large number of primes seem to divide the current value, and to use the parallel electronic I/O lines on the wafer to report their identities with a proper encoding technique. The PC only has to perform trial division by about 50 known primes rather than trial division by all primes in the factor bases. The I/O lines are used to load the A and B registers for the new sieving interval, and are idle during the actual sieving. However, these are long high capacitance wires which cannot continuously report the identity of the flashing LED's at each clock cycle. The solution is to make sure that reports will be generated only when the photodetector senses a possible smoothness event, and only by the approximately 50 relevant cells.

To achieve this, we add an optical feedback path from the photodetector to the cells. When the light measured by the photodetector exceeds the threshold, it flashes a query LED placed next to it (and opposite the wafer). Each cell has an additional photodetector for the query LED. When this query LED is sensed, each cell checks whether it flashed its own LED a certain number of clock cycles ago (depending on the total delay along the optical path), and if so, reports its identity on the I/O lines.

The simplest way of implementing this idea is to separate the flashing of the LED and the reloading of the counter in each cell. Assume for example that each B register is a ripple counter which flashes its LED when it reaches state "10. . . 010000" (cf. 6.2). It continues to count upwards, reports its identity if the query LED is sensed AND its state is "10. . . 011000", and reloads itself from register A when its state reaches "10. . . 011001". The value of the A register has to be augmented to compensate for this delay, and different wavelengths have to be used for the various LED's and photodetectors to avoid confusion between the various optical functions.

6.4 Using Separate Algebraic and Rational LED's

In the QS about half the primes up to a bound $\mathbf{B}$ do not yield arithmetic progressions, and the other half generate two distinct arithmetic progressions. This implies that in the original TWINKLE design a single cell contains one A register for a prime, two B registers for the arithmetic progressions, and one LED that flashes if either B register enters a special state. For QS with factor base bound $\mathbf{B}$ one may therefore expect $\pi(\mathbf{B})$ arithmetic progressions generated by $\pi(\mathbf{B})/2$ cells with 3 registers (a single A and two B), and one LED per cell.

NFS requires a different cell design. If distinct cells are feasible we show that the same average number of registers per cell (namely 3) can be achieved as in QS. Let $\mathbf{B} = 2^{24}$ (cf. 4.2). All primes less than $\mathbf{B}$ are potential divisors of $F_2(a,b)$. Thus, at least $\pi(\mathbf{B})$ different cells, each with at least an A register, a B register, and an LED, are needed for the resulting $\pi(\mathbf{B})$ rational arithmetic progressions. For $F_1(a,b)$ the number of arithmetic progressions required for a certain prime $p < \mathbf{B}$ depends on the number of distinct roots of $f_1 \bmod p$. On average one may expect that for

- 11/30 of the primes $f_1 \bmod p$ does not have a root;
- 3/8 of the primes $f_1 \bmod p$ has a single root;
- 1/6 of the primes $f_1 \bmod p$ has two distinct roots;
- 1/12 of the primes $f_1 \bmod p$ has three distinct roots; and
- 1/120 of the primes $f_1 \bmod p$ has five distinct roots.

(Note that $11/30+3/8+1/6+1/12+1/120 = (44+45+20+10+1)/120 = 1$.) Let $p < \mathbf{B}$ be a prime for which $f_1 \bmod p$ has d distinct roots. This p requires $d+1$ distinct arithmetic progressions which can be taken care of by a single cell with $d+2$ registers: a single A register for p, a single B register for the rational arithmetic progression, and d different B registers for the d distinct algebraic arithmetic progressions. Here we use that unless n has a small factor, p cannot divide both $F_1(a,b)$ and $F_2(a,b)$, so that the rational arithmetic progression is different from the algebraic ones. This leads to a total of $\pi(\mathbf{B})$ cells: $11*\pi(\mathbf{B})/30$ with 2 registers, $3*\pi(\mathbf{B})/8$ with 3 registers, $\pi(\mathbf{B})/6$ with 4 registers, $\pi(\mathbf{B})/12$ with 5 registers, and $\pi(\mathbf{B})/120$ with 7 registers. The total number of registers is $(2*11/30+3*3/8+4/6+5/12+7/120)*\pi(\mathbf{B}) = (88+135+80+50+7)*\pi(\mathbf{B})/120 = 3*\pi(\mathbf{B})$. The expected number of arithmetic progressions equals $\pi(\mathbf{B})+(3/8+2/6+3/12+5/120)*\pi(\mathbf{B}) = 2*\pi(\mathbf{B})$. Thus, for NFS with factor base bounds $\mathbf{B}$ one may expect $2*\pi(\mathbf{B})$ arithmetic progressions generated by $\pi(\mathbf{B})$ cells with on average 3 registers, which is not much different from QS. The numbers of LED's per cell is discussed below.

The simplest approach to simultaneous algebraic and rational sieving would be to let the rational B register and the algebraic B registers in a particular cell share the same LED. In the terminology of 5.2 and 5.3 this would mean that the dual condition "$S_1(x) > T_1(-\log(q))$ and $S_2(x) > T_2$" is replaced by the single condition "$S_1(x) + S_2(x) > T_1(-\log(q)) + T_2$". Extensive software experiments using this simplification were not encouraging, as it leads to too many false reports (with the original T_i's) or too many missed pairs (with adapted T_i's). Nevertheless, for TWINKLE it may be worth trying this approach. It would lead to a single LED per cell.

A more promising approach would be to have the algebraic flashes on the odd beat and the rational flashes on the even beat. This can easily be realized by storing $2p$ instead of p in the A registers and by changing the values initially stored in the B registers in the obvious way. If the photodetector detects a pair of consecutive odd and even high intensities a *report* occurs, i.e., a good pair may have been found. This approach still requires a single LED per cell, but it has the disadvantage that it takes two clock cycles to process a single sieve location.

Another approach would be to use LED's of different colours for algebraic and rational flashes. The algebraic LED flashes if either of the algebraic B registers is in a special state, and the rational LED flashes if the rational B register is in a special state. A report occurs if two photodetectors for the two different frequencies simultaneously detect a high intensity. In this approach all cells have a rational LED and 19/30 of the cells have an algebraic LED as well, for a total of $49 * \pi(\mathbf{B})/30$ LED's, which is almost 5/3 LED's per cell on average. The advantage of this approach, which we assume in the sequel, is that processing a single sieve location takes a single clock cycle, as in the original TWINKLE design. Note that it requires yet another different wavelength to avoid confusion with other optical signals.

6.5 Geometric Considerations

The geometry of the TWINKLE design described in [7] was based on the operational requirement that the time delay along the optical paths (from the clocking LED to all the cells on the flat wafer, and from these cells back to the summing photodetector) should be as uniform as possible. The recommended design placed the wafer at one face of a cylindrical tube, the photodetector at the center of the opposite face, and several synchronized clocking LED's around the perimeter of this face. This physical design reduced but did not eliminate the time difference between various optical paths in the tube. As a result, the tube had to be quite long, and thus the LED's on the wafer had to be made stronger, bigger, and more power consuming.

A greatly improved design (which was independently discovered by several researchers in private communication) places both the clocking LED and the photodetector at the center of one face of the cylinder, and at the focal point of a convex lens placed inside the cylinder between its two faces. Since all the relevant light rays (in both directions) between the lens and the wafer are parallel along the cylinder, all the wave fronts (which are perpendicular to the light rays) are flat and parallel to the wafer, and thus the time delay from the clocking LED to any point in the wafer and from there back to the photodetector is exactly the same. In addition, all the light gathered by the lens is concentrated on the small face of the photodetector, and thus the LED's on the wafer can be made smaller and weaker.

7 Analysis of TWINKLE-Assisted NFS Factorizations

7.1 Line Sieving for 512-Bit Numbers

To simplify the analysis, we assume for the moment that the factor base size is irrelevant. Under this assumption, the device as described in [7] and with the modifications from Section 6, can straightforwardly be used to perform line sieving for an NFS 512-bit factorization. The primes p for the (p,r) in P_2 are loaded once in the A registers, with the exact distribution over the different types of cells determined by the number of roots of $f_1 \bmod p$, as implied by the description in 6.4. For the first line ($b = 1$) the B register corresponding to a pair $(p,r) \in P_1 \cup P_2$ is initialized as $r + \mathbf{A} - p * [(r+\mathbf{A})/p]$, where $\mathbf{A} = 9 * 10^9$. The initial B-value for the next line follows by adding r to the initial value for the current line and taking the result modulo p. Thus, computation of the two million initial B register values for the next line can be done on an ordinary PC in less time than it takes TWINKLE to sieve the current line (see below). As shown in 5.5, a total of $\mathbf{A}/s = 9 * 10^5$ lines of length $2 * \mathbf{A} = 1.8 * 10^{10}$ each should suffice. As in Section 3 we consider two possible scenarios:

1. A modified TWINKLE device running at the maximum possible speed of 10 GHz. Each sieving interval of length $1.8 * 10^{10}$ can be scanned in 1.8 seconds. Reloading the B registers can be done in 0.02 seconds (cf. [7]) when done sequentially for all registers, or in 0.002 seconds when done in 10-fold parallelism, and can thus be neglected. All $9 * 10^5$ lines can be processed in approximately $1.8 * 9 * 10^5$ seconds, which is less than 3 weeks. A speed-up by 25% can be obtained by excluding 'even, even' locations from the sieve (cf. 5.5). This improvement is not reflected in our TWINKLE running time estimates but is included in the running times from [1]. The 3 week estimate includes the time for reloading and resieving, but does not include the time to do the actual divisions and cofactor factorizations. The latter can be done in $1.8*9*10^5$ seconds by about 43 loosely coupled PC's, as estimated in 5.4, and one additional PC is needed to compute the root updates. A total of 44 PC's would be able to do the sieving step in about 21 weeks (using special q sieving, cf. [1]). The improvement ratio is about a factor 8.
2. A modified TWINKLE device running at the minimum recommended speed of 1 GHz (cf. 6.1). Each sieving interval of length $1.8*10^{10}$ can be scanned in 18 seconds and all $9*10^5$ lines can be processed in approximately $18*9*10^5$ seconds, which is less than 27 weeks. It follows from 5.4 that 5 auxiliary PC's suffice for all auxiliary computations (divisions, cofactor decompositions, and root updates). A total of 5 PC's would be able to do the sieving step in about 186 weeks, and the improvement ratio we obtain is about a factor 7.

Thus, as in Section 3, we get about the same improvement ratio regardless of the clock rate of the TWINKLE device. This is due to the fact that the computational bottleneck is in the supporting PC's. Note that the improvement ratio is close to the maximum attainable ratio implied by the last paragraph of 5.4.

The factor base sizes specified in 4.2 imply that the TWINKLE device, using the cell design as in 6.4, would contain about 10 wafers of more or less the same size as the wafers described in [7]. For that reason we now concentrate on how special q sieving may be used for TWINKLE-assisted factorizations of 512-bit and 768-bit numbers.

7.2 Special Sieving for 512-Bit Numbers

Naïve implementation of special q sieving on a modified TWINKLE device is not promising. Despite the fact that a total of only 10^{15} sieve locations (cf. 5.5) have to be processed (which can, at 10 GHz, be done in less than 28 hours, including the resieving), the B registers have to be reloaded every $2 * 2^{12} =$ 8,192 sieve locations (cf. 5.5). Even with 10-fold parallelized reloading this adds $0.002 * (10^{15}/8{,}192) = 2.4 * 10^8$ seconds, i.e., almost 8 years, to the sieving time, without even considering how a PC is supposed to prepare the required data in 0.8 microseconds (the sieving time per line). As noted in 5.3, this problem is due to the fact that in special q sieving one cannot touch all factor base elements for all lines without violating the NFS running time.

A much better solution is obtained by radically changing the approach, and to make use of the fact that the majority of the factor base elements does not hit a particular line. Of the 2 million (p, r) pairs with $p > 2*2^{12}$ on average only about 10^4 hit a particular line, and if it hits, it hits just once. It follows that on average $2 * \pi(8{,}192) + 10^4$ pairs must be considered per line, and that roughly $2 * 10^4$ cells suffice if the same cell is associated with different p's (of approximately the same size) for different lines. Of these cells $2 * \pi(8{,}192)$ are as usual with fixed A registers, variable B registers, and proper arithmetic progressions. The other cells, however, need B registers only, assuming that their (variable) primes can be derived from their location if a report occurs. This follows from the fact that there is just a single hit, which implies that there is no true arithmetic progression to sieve with, and that the step size p is not needed. A clear advantage is that it simplifies the design of the TWINKLE device considerably, because only a single wafer with about $2 * 10^4$ cells would suffice. And most cells are even simpler than usual since they contain just one B register, two photodetectors, and a single rational or algebraic LED (split evenly among the cells). Note that the TWINKLE device would not actually be sieving for the primes $> 8{,}192$ but act as an accumulator of logarithms of primes corresponding to identical B values.

We analyze the resulting speed for this modified and simplified TWINKLE device running at the maximum possible speed of 10 GHz. The number of sieve locations per special q is $8,192*5,000$ which can be scanned in 4 milliseconds. Per line about $2*\pi(8{,}192)+10^4$ values have to be reloaded. This can be done in 0.12 milliseconds. Thus, the auxiliary PC's have $5{,}000*0.00012 + 0.004 = 0.6$ seconds to prepare the list of register values ordered according to the lines where they should be used. Obviously, the PC's should also not touch each line per (p, r) pair, so they will have to use some type of lattice technique to process each (p, r) pair. The lattice siever from [2] that was used in [1] takes about 8.8 seconds to provide the required list. Thus, one may expect that about $8.8/0.6 \approx 15$ PC's are required to prepare the line data for a TWINKLE device. It follows that a single modified and simplified TWINKLE device supported by about 15 PC's can do the special q NFS sieving for a 512-bit number in $(\pi(5*10^8)-\pi(2^{24}))*0.6$ seconds, i.e., about half a year. To keep up with the actual divisions and cofactor factorizations at 0.1 seconds per resulting relation (cf. 5.4) for 100 million relations (cf. 4.3), a single PC suffices. A total of 16 PC's would be able to do the sieving in slightly more than a year (cf. [1]), and the total improvement ratio is about 2.3. But note that the auxiliary PC's require only a modest amount of memory, whereas the PC's running the special q siever from [2] need to be able to allocate 64 megabytes of RAM to run efficiently. The same analysis holds when the TWINKLE device runs at the minimum recommended speed of 1 GHz.

The single wafer required for this modified and simplified TWINKLE device is much smaller than the one proposed in [7]. From our analysis it looks as if loading the new line data is the big bottleneck for special q sieving on the TWINKLE device. If that can be done x times faster, the TWINKLE device will run about x times faster. But the comparison to PC's would not be affected, because also x times more PC's would be needed to prepare the line data. So, from that point of view the data loading time is not a bottleneck, and we conclude that the PC support required for the preparation of the list of line data has a similar (and even stronger) effect on TWINKLE-assisted special q sieving as the post-processing PC-support has for TWINKLE-assisted line sieving.

7.3 Special q Sieving for 768-Bit Numbers

Based on the asymptotic running time of the NFS, it may be expected that 768-bit numbers are at most 5,000 times harder to factor than 512-bit numbers. The size of the total sieving region grows proportional to the running time, and the factor base sizes and size of sieving region per special q grow proportional to the squareroot of the running time. Based on the figures from [1] we expect that 90,000 PC's with huge RAM's of about 5 gigabytes per PC can do the special q sieving in about a year. Based on extrapolation of the results from [1] we expect that one terabyte of disk space (about 50 standard PC hard disks costing a total of about \$10,000) would suffice to store the data resulting from the sieving step.

Using well known structured Gaussian elimination methods that require only sequential disk-access to the data, a matrix of less than half a billion rows and columns and on average less than 100 entries per row can be built, requiring less than 200 gigabytes of disk space. Extrapolation of existing PC implementations of the block Lanczos algorithm suggests that this still relatively sparse matrix can be processed in less than 4,000 years on a single PC, using a blocking factor of 32. Preliminary results of block Lanczos parallelization seem to indicate that k-fold parallelization leads to a $(k/3)$-fold speed-up, where so far no values of $k > 16$ have been used (cf. [4]). Assuming that this parallelization scales up to larger k, application of these preliminary results with $k = 5 * 4 * 4,000 =$ 80,000 and a conservative (80,000/5)-fold speed-up leads to the estimate that 80,000 PC's can do the matrix step in 3 months, when they are connected to a sufficiently fast network. Each of the PC clients would need only a few megabytes of RAM to store only a small fraction of the matrix. Unlike other figures in this paper, this estimate has not been confirmed by an actual implementation, and we stress that it is based on the assumption that the parallelization from [4] scales reasonably well. Note that future 64-bit PC's can use a blocking factor of 64, thereby halving the number of Lanczos iterations and substantially reducing the time required. Another way to parallelize block Lanczos that may be worth considering is to replace each PC client by clusters of, say, t PC clients, thereby further reducing the number of Lanczos iterations by a factor t.

We now consider how the simplified design from 7.2 scales up to 768-bit numbers. The total sieving time increases by a factor of about 5,000 to approximately 2,500 years. The factor base sizes increase by a factor 70, but so does the size of the sieving region per special q, so the same number of supporting PC's will be able to prepare the required lists of line data, per TWINKLE device. The wafer size would increase by a factor less than 9, and thus become comparable to the size proposed in [7]. We can thus conclude that about 5,000 modified and simplified TWINKLE devices supported by about 80,000 PC's can do the sieving step for a 768-bit number in about half a year. With the above estimate for the matrix step we arrive at the estimate given in the abstract.

PC's with 5 gigabyte RAM's which are needed to run the special q siever in standard NFS factorizations are highly specialized: Only a negligible number of such machines exist, and they have very few other applications. On the other hand, the auxiliary PC's in TWINKLE-assisted factorizations do not need exceptionally large memories, and thus it is possible to timeshare standard PC's which are used for other purposes in the organization (or over the Internet) during daytime. Large memories are also not needed for parallelized block Lanczos implementations. Since the 80,000 PC's are likely to be more expensive than the 5,000 TWINKLE devices, their free availability can dramatically reduce the cost of the hardware, and make a TWINKLE-assisted attack on a 768-bit RSA modulus much more feasible than a pure PC-based attack that uses dedicated PC's with huge memories.

8 TWINKLE without Optoelectronics

After the publication of the original TWINKLE paper, several alternative implementations were proposed by various researchers. The main theme of the modified designs was to replace the optoelectronic adder by an electronic adder of one of the following types:

1. An analog adder, in which each cell adds some current to a common line. An event is registered whenever the total current is high enough.
2. A digital adder, in which a tree of local 2-way adders adds the binary numbers which represent the contributions of the various cells.

3. A one dimensional systolic array, in which each cell increments one in p numbers passing through it for some p. The sequence of numbers “falling off” the end of the array is scanned for high entries.

The analog adder is likely to be too slow to react to rapidly changing signals due to the high capacitance of the tree of wires. The digital adder tree is faster, but each adder is likely to use a larger area and more power than a single LED which is dark most of the time. In addition, large adder trees are not fault tolerant, since a dead adder can eliminate the contributions of all the cells in its subtree. Similarly, a systolic array requires complex bypass mechanisms to overcome the dead or unreliable cells along it, since each number should pass through all the cells.

A purely electronic design may look more attractive than an optoelectronic design, since it is slightly easier to design and somewhat cheaper to manufacture. However, this is not likely to be a major consideration in large scale factoring efforts by large organizations, and in most respects it makes the design less efficient: Gallium Arsenide technology is faster than silicon technology, LED’s are smaller than adders, independent cells are more fault tolerant than interconnected cells, and ultraprecise timing is easier to achieve with optics than with electronics.

9 Conclusion

From our analysis we conclude that both the original TWINKLE device as proposed in [7] and the variant that runs at one tenth of the speed can be expected to achieve a substantial speed-up over a PC implementation for the QS-factorization of 384-bit numbers. We described a modified version of the TWINKLE device that is better suited for the implementation of the NFS factoring algorithm than the original design. We found that 768-bit RSA moduli are more vulnerable to NFS attacks by our improved TWINKLE design than by current PC implementations.

References

1. S. Cavallar, B. Dodson, A.K. Lenstra, W. Lioen, P.L. Montgomery, B. Murphy, H.J.J. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, P. Zimmermann, Factorization of a 512-bit RSA modulus, these proceedings.
2. R. Golliver, A.K. Lenstra, K.S. McCurley, Lattice sieving and trial division, Proceedings of ANTS-I, Lecture Notes in Computer Science 877, Springer, pp 18-27.
3. A.K. Lenstra, H.W. Lenstra, Jr., The development of the number field sieve, Lecture Notes in Mathematics 1554, Springer, 1993.
4. P.L. Montgomery, Parallel block Lanczos, presentation at the RSA 2000 conference, Jan 17, 2000.

5. P.L. Montgomery, B. Murphy, Improved polynomial selection for the number field sieve. Extended abstract for the Conference on the Mathematics of Public-Key Cryptography, June 13-17, 1999, The Fields Institute, Toronto, Ontario, Canada.
6. C. Pomerance, The quadratic sieve factoring algorithm, Proceedings of Eurocrypt'84, Lecture Notes in Computer Science, Springer, pp 169-182.
7. A. Shamir, Factoring large numbers with the TWINKLE device, Proceedings of the CHES conference, Lecture Notes in Computer Science 1717, Springer, August 1999.