Factorization of RSA–140 Using the Number Field Sieve ⋆

Stefania Cavallar[3], Bruce Dodson[5], Arjen Lenstra[1], Paul Leyland[6], Walter Lioen[3], Peter L. Montgomery[7], Brian Murphy[2], Herman te Riele[3], and Paul Zimmermann[4]

[1] Citibank, Parsippany, NJ, USA
arjen.lenstra@citicorp.com
[2] Computer Sciences Laboratory, The Australian National University
Canberra ACT 0200, Australia
murphy@cslab.anu.edu.au
[3] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
{cavallar,walter,herman}@cwi.nl
[4] Inria Lorraine and Loria, Nancy, France
Paul.Zimmermann@loria.fr
[5] Lehigh University, Bethlehem, PA, USA
bad0@Lehigh.edu
[6] Microsoft Research Ltd, Cambridge, UK
pleyland@microsoft.com
[7] 780 Las Colindas Road, San Rafael, CA 94903–2346 USA
Microsoft Research and CWI
pmontgom@cwi.nl

Abstract. On February 2, 1999, we completed the factorization of the 140–digit number RSA–140 with the help of the Number Field Sieve factoring method (NFS). This is a new general factoring record. The previous record was established on April 10, 1996 by the factorization of the 130–digit number RSA–130, also with the help of NFS. The amount of computing time spent on RSA–140 was roughly twice that needed for RSA–130, about half of what could be expected from a straightforward extrapolation of the computing time spent on factoring RSA–130. The speed-up can be attributed to a new polynomial selection method for NFS which will be sketched in this paper.
The implications of the new polynomial selection method for factoring a 512–bit RSA modulus are discussed and it is concluded that 512–bit (= 155–digit) RSA moduli are easily and realistically within reach of factoring efforts similar to the one presented here.

1 Introduction

Factoring large numbers is an old and fascinating métier in number theory which has become important for cryptographic applications after the birth, in 1977, of

⋆ Paper 030, accepted to appear in the Proceedings of Asiacrypt '99, Singapore, November 14–18, 1999. URL: http://www.comp.nus.edu.sg/~asia99 .

the public-key cryptosystem RSA [22]. Since then, people have started to keep track of the largest (difficult) numbers factored so far, and reports of new records were invariably presented at cryptographic conferences. We mention Eurocrypt '89 (C100[1] [14]), Eurocrypt '90 (C107 and C116 [15]), Crypto '93 (C120, [8]), Asiacrypt '94 (C129, [1]) and Asiacrypt '96 (C130, [6]). The 130–digit number was factored with help of the *Number Field Sieve* method (NFS), the others were factored using the *Quadratic Sieve* method (QS).

For information about QS, see [21]. For information about NFS, see [13]. For additional information, implementations and previous large NFS factorizations, see [9–12].

In this paper, we report on the factoring of RSA–140 by NFS and the implications for RSA. The number RSA–140 was taken from the RSA Challenge list [23]. In Sect. 2 we estimate how far we are now from factoring a 512–bit RSA modulus. In Sect. 3, we sketch the new polynomial selection method for NFS and we give the details of our computations which resulted in the factorization of RSA–140.

2 How far are we from factoring a 512–bit RSA modulus?

RSA is widely used today. We quote from RSA Laboratories' *Frequently Asked Questions about today's Cryptography 4.0* (`http://www.rsa.com/rsalabs/faq/html/3-1-9.html`):

```
Question 3.1.9.
Is RSA currently in use?

     RSA is  currently  used in a wide  variety  of  products,
     platforms, and  industries  around the world. It is found
     in many commercial software products and is planned to be
     in many more. RSA is built into current operating systems
     by Microsoft,  Apple,  Sun, and  Novell. In hardware, RSA
     can be found in secure  telephones, on  Ethernet  network
     cards, and on smart cards. In addition, RSA is
     incorporated into all of the major  protocols for  secure
     Internet  communications, including S/MIME  (see Question
     5.1.1), SSL (see Question 5.1.2), and S/WAN (see Question
     5.1.3). It is also used internally in many  institutions,
     including branches of the U.S. government, major
     corporations, national laboratories, and universities.

     At the time of this  publication, RSA technology is
     licensed by about 350 companies.  The estimated installed
     base of RSA  encryption  engines is  around  300 million,
     making it by far the most widely used public-key
```

[1] By "Cxxx" we denote a composite number having xxx decimal digits.

`cryptosystem in the world. This figure is expected to grow rapidly as the Internet and the World Wide Web expand.`

The best size for an RSA key depends on the security needs of the user and on how long the data needs to be protected. At present, information of very high value is protected by 512–bit RSA keys. For example, CREST [7] is a system developed by the Bank of England and used to register all the transfers of stocks and shares listed in the United Kingdom. The transactions are protected using 512–bit RSA keys. Allegedly, 512–bit RSA keys protect 95% of today's E-commerce on the Internet [24].

The amount of CPU time spent to factor RSA–140 is estimated to be only twice that used for the factorization of RSA–130, whereas on the basis of the heuristic complexity formula [3] for factoring large N by NFS:

$$\mathcal{O}\left(\exp\left((1.923+o(1))(\log N)^{1/3}(\log\log N)^{2/3}\right)\right),$$

one would expect an increase in the computing time by a factor close to four. This has been made possible by algorithmic improvements (mainly in the polynomial generation step [18], and to a lesser extent in the sieving step and the filter step of NFS), and by the relative increase in memory speed of the workstations and PCs used in this project.

After the completion of RSA–140, we completely factored the 211–digit number $10^{211}-1$ with the Special Number Field Sieve (SNFS) at the expense of slightly more computational effort than we needed for RSA–140. We notice that the polynomial selection stage is easy for $10^{211}-1$. Calendar time was about two months. This result means a new factoring record for SNFS (see `ftp://ftp.cwi.nl/pub/herman/NFSrecords/SNFS-211`). The previous SNFS record was the 186–digit number $32633^{41}-1$ (see `ftp://ftp.cwi.nl/pub/herman/NFSrecords/SNFS-186`).

Experiments indicate that the approach used for the factorization of RSA–140 may be applied to RSA–155 as well. Estimates based on these experiments suggest that the total effort involved in a 512–bit factorization (RSA–155 is a 512–bit number) would require only a fraction of the computing time that has been estimated in the literature so far. Also, there is every reason to expect that the matrix size, until quite recently believed to be the main stumbling block for a 512–bit factorization using NFS, will turn out to be quite manageable. As a result 512–bit RSA moduli do, in our opinion, not offer more than marginal security, and should no longer be used in any serious application.

3 Factoring RSA–140

We assume that the reader is familiar with NFS [13], but for convenience we briefly describe the method here. Let N be the number we wish to factor, known to be composite. There are four main steps in NFS: polynomial selection, sieving, linear algebra, and square root.

In the *polynomial selection step*, two irreducible polynomials $f_1(x)$ and $f_2(x)$ with a common root m mod N are selected having as many as practically possible smooth values over a given factor base.

In the *sieving step* which is by far the most time-consuming step of NFS, pairs (a, b) are found with $\gcd(a, b) = 1$ such that both

$$b^{\deg(f_1)} f_1(a/b) \text{ and } b^{\deg(f_2)} f_2(a/b)$$

are smooth over given factor bases, i.e., factor completely over the factor bases. Such a pair (a, b) is called a *relation*. The purpose of this step is to collect so many relations that several subsets S of them can be found with the property that a product taken over S yields an expression of the form

$$X^2 \equiv Y^2 \pmod{N}. \tag{1}$$

For approximately half of these subsets, computing $\gcd(X - Y, N)$ yields a non-trivial factor of N (if N has exactly two distinct factors).

In the *linear algebra step*, the relations found are first filtered with the purpose of eliminating duplicate relations and relations in which a prime or prime ideal occurs which does not occur in any other relation. If a prime ideal occurs in exactly two or three relations, these relations are combined into one or two (respectively) so-called relation-sets. These relation-sets form the columns of a very large sparse matrix over $\mathcal{F}_2$. With help of an iterative block Lanczos algorithm a few dependencies are found in this matrix. This is the main and most time- and space-consuming part of the linear algebra step.

In the *square root step*, the square root of an algebraic number of the form

$$\prod_{(a,b)\in S} (a - b\alpha)$$

is computed, where α is a root of one of the polynomials $f_1(x), f_2(x)$, and where a, b and the cardinality of the set S are all a few million. The norms of all $(a - b\alpha)$'s are smooth. This leads to a congruence of the form (1).

In the next four subsections, we describe these four steps, as carried out for the factorization of RSA–140. We pay most attention to the polynomial selection step because, here, new ideas have been incorporated which led to a reduction of the expected – and actual – sieving time for RSA–140 (extrapolated from the RSA–130 sieving time) by a factor of 2.

3.1 Polynomial selection

For number field sieve factorizations we use two polynomials $f_1, f_2 \in \mathbb{Z}[x]$ with, amongst other things, a common root m mod N. For integers as large as RSA–140, a modified base-m method is the best method we know of choosing these polynomials. Montgomery's "two-quadratics" method [11] is the only known alternative, and it is unsuitable for numbers this large. With the base-m method, we fix a degree d (here $d = 5$) then seek $m \approx N^{1/(d+1)}$ and a polynomial f_1 of degree d for which

$$f_1(m) \equiv 0 \pmod{N}. \tag{2}$$

The polynomial f_1 descends from the base-m representation of N. Indeed, we begin with $f_1(x) = \sum_{i=0}^{d} a_i x^i$ where the a_i are the coefficients of the base-m representation, adjusted so that $-m/2 \leq a_i < m/2$.

Sieving occurs over the homogeneous polynomials $F_1(x, y) = y^d f_1(x/y)$ and $F_2(x, y) = x - my$. The aim for polynomial selection is to choose f_1 and m such that the values $F_1(a, b)$ and $F_2(a, b)$ are simultaneously smooth at many coprime integer pairs (a, b) in the sieving region.

We consider this problem in two stages; first we must decide what to look for, then we must decide how to look for it. The first stage requires some understanding of polynomial yield; the second requires techniques for generating polynomials with good yield. In this paper we seek only to outline our techniques. Full details will be published at a later date.

Polynomial yield. The *yield* of a polynomial $F(x, y)$ refers to the number of smooth (or almost smooth) values it produces in its sieve region. Ultimately of course we seek a *pair* of polynomials F_1, F_2 with good yield. Since F_2 is linear, all primes are roots of F_2, so the difficult polynomial is the non-linear F_1. Hence, initially, we speak only of the yield of F_1.

There are two factors which influence the yield of F_1. These are discussed in a preliminary manner in [19]. We call the factors *size* and *root properties*. Choosing good F_1 requires choosing F_1 with a good *combination* of size and root properties.

By *size* we refer to the magnitude of the values taken by F_1. It has always been well understood that size affects the yield of F_1. Indeed previous approaches to polynomial selection have sought polynomials whose size is smallest (for example, [6]).

The influence of root properties however, has not previously been either well understood or adequately exploited. By *root properties* we refer to the extent to which the distribution of the roots of F_1 modulo small p^k, for p prime and $k \geq 1$, affects the likelihood of F_1 values being smooth. In short, if F_1 has many roots modulo small p^k, the values taken by F_1 "behave" as if they are much smaller than they actually are. That is, on average, the likelihood of F_1-values being smooth is increased. We are able to exploit this property to the extent that F_1 values behave as if they are as little as 1/1000 their actual value. We estimate this property alone increases yield by a factor of four due (by comparison to sieving over random integers of the same size).

Generating polynomials with good yield. We consider this problem in two stages. In the first stage we generate a large sample of good polynomials. Although each polynomial generated has a good combination of size and root properties, there remains significant variation in the yield across the sample.

Moreover, there are still far too many polynomials to conduct sieving experiments on each one. Thus in the second stage we identify *without* sieving, the best polynomials in the sample. The few polynomials surviving this process are then subjected to sieving experiments.

Consider the first stage. We concentrate on so-called *skewed* polynomials, that is, polynomials whose first few coefficients (a_5, a_4 and a_3) are small compared to m, and whose last few coefficients (a_2, a_1 and a_0) may be large compared to m. In fact usually $|a_5| < |a_4| < \ldots < |a_0|$. To compensate for the last few coefficients being large, we sieve over a region much longer in x than y. We take the region to be a rectangle whose length-to-height ratio is s.

Notice that any base-m polynomial may be re-written so that sieving occurs over a rectangle of skewness s. Let $m = O(N^{1/(d+1)})$ giving an unmodified base-m polynomial F_1 with coefficients also $O(N^{1/(d+1)})$. The expected sieve region for F_1 is a "square" given by $\{(x, y) : -M \leq x \leq M \text{ and } 1 \leq y \leq M\}$ for some M. For some (possibly non-integer) $s \in \mathbb{R}$ let $x' = x/\sqrt{s}$, $y' = y\sqrt{s}$ and $m' = ms$. The polynomials $F_1(x', y')$ and $F_2(x', y')$ with common root m', considered over a rectangle of skewness s and area $2M^2$, have the same norms as F_1 and F_2 over the original square region. Such a skewing process can be worthwhile to increase the efficiency of sieving.

However, we have additional methods for constructing *highly* skewed polynomials with good yields. Hence, beyond simply skewing the region on unmodified base-m polynomials, we focus on polynomials which are themselves intrinsically skewed. The search begins by isolating skewed polynomials which are unusually small over a rectangle of some skewness s and which have better than average root properties. The first quality comes from a numerical optimization procedure which fits a sieve region to each polynomial. The second quality comes from choosing (small) leading coefficients divisible by many small p^k.

We then exploit the skewness to seek adjustments to f_1 which cause it to have exceptionally good root properties, *without* destroying the qualities mentioned above. We can make any adjustment to f_1 as long as we preserve (2). We make what we call a *rotation* by P for some polynomial $P(x)$. That is, we let

$$f_{1,P}(x) = f_1(x) + P(x) \cdot (x - m)$$

where $P \in \mathbb{Z}[x]$ has degree small compared to d. Presently we use only linear $P(x) = j_1 x - j_0$ with j_1 and j_0 small compared to a_2 and a_1 respectively. We use a sieve-like procedure to identify pairs (j_1, j_0) which cause $f_{1,P}$ to have exceptionally good root properties mod small p^k. At the end of this procedure (with $p^k < 1000$ say) we have a large set of candidate polynomials.

Consider then the second stage of the process, where we isolate without sieving the polynomials with highest yield. Notice that as a result of looking at a large range of a_d the values of m may vary significantly across the sample. At this stage it is crucial then to consider *both* F_1 and F_2 in the rating procedure. Indeed, the values s vary across the sample too.

We use a quantitative estimate of the effect of the root properties of each polynomial. We factor this parameter into estimates of smoothness probabilities for F_1 and F_2 across a region of skewness s. It is not necessary to estimate the yield across the region, simply to rank the polynomial pairs in the order in which we expect their yields to appear. Of course to avoid missing good polynomial pairs it is crucial that the metric so obtained be reliable.

At the conclusion of this procedure we perform short sieving experiments on the top-ranked candidates.

Results. Before discussing the RSA–140 polynomial selection results, we briefly consider the previous general factoring record, RSA–130 [6]. As a test, we repeated the search for RSA–130 polynomials and compared our findings to the polynomial used for the factorization. We searched for non-skewed polynomials only, since that is what was used for the RSA–130 factorization. Despite therefore finding fewer polynomials with exceptional root properties, we did, in a tiny fraction of the time spent on the RSA–130 polynomial search, find several small polynomials with good root properties. Our best RSA–130 polynomial has a yield approximately twice that of the polynomial used for the factorization. In essence, this demonstrates the benefit of knowing "what to look for".

The RSA–140 search however, further demonstrates the benefit of knowing "how to look for it". Here of course we exploit the skewness of the polynomials to obtain exceptional root properties.

Sieving experiments on the top RSA–140 candidates were conducted at CWI using line sieving. All pairs were sieved over regions of the same area, but skewed appropriately for each pair. Table 1 shows the relative yields of the top five candidate pairs, labeled A, . . . , E. These yields match closely the predictions of our pre-sieving yield estimate.

Table 1. Relative Yields of the top RSA–140 polynomials

Poly.	Rel. Yield
A	1.00
B	0.965
C	0.957
D	0.931
E	0.930

The chosen pair, pair A, is the following:

$$\begin{aligned}F_1(x,y) = \quad & 43\,96820\,82840\; x^5\\ & +39031\,56785\,38960\, y\, x^4\\ & -7387\,32529\,38929\,94572\, y^2x^3\\ & -190\,27153\,24374\,29887\,14824\, y^3x^2\\ & -6\,34410\,25694\,46461\,79139\,30613\, y^4x\\ & +31855\,39170\,71474\,35039\,22235\,07494\, y^5\end{aligned}$$

and

$$F_2(x, y) = x - 3\,44356\,57809\,24253\,69517\,79007\;\; y,$$

with $s \approx 4000$.

Consider F_1, F_2 with respect to size. We denote by a_{max} the largest $|a_i|$ for $i = 0, \ldots, d$. The un-skewed analogue, $F_1(63x,\, y/63)$, of $F_1(x, y)$ has

$$a_{max} \approx 5 \cdot 10^{20}.$$

A typical unmodified base-m polynomial has

$$a_{max} \approx 1/2N^{1/6} \approx 8 \cdot 10^{22}.$$

The un-skewed analogue, $F_2(63x,\, y/63)$, of $F_2(x, y)$ has

$$a_{max} \approx 3N^{1/6}.$$

Hence, compared to the typical case F_1 values have shrunk by a factor about 160 whilst F_2 values have grown by a factor of 3. F_1 has real roots x/y near -4936, 2414, and 4633.

Now consider F_1 with respect to root properties. Notice that a_5 factors as $2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 41 \cdot 29759$. Since also $4|a_4$ and $2|a_3$, $F_1(x, y)$ is divisible by 8 whenever y is even. $F_1(x, y)$ has at least three roots x/y modulo each prime from 3 to 17 (some of which are due to the factorization of the leading coefficient), and an additional 35 such roots modulo the 18 primes from 19 to 97.

We estimate that the yield of the pair F_1, F_2 is approximately eight times that of a skewed pair of average yield. Approximately a factor of four in that eight is due to the root properties, the rest to its size. We estimate the effort spent on the polynomial selection to be equivalent to 0.23 CPU years (approximately 60 MIPS-years). Searching longer may well have produced better polynomials, but we truncated the search to make use of idle time on workstations over the Christmas period (for sieving). We leave as a subject of further study the trade-off between polynomial search time and the corresponding saving in sieving time.

3.2 Sieving

Partially for comparison, two sieving methods were used: lattice sieving and line sieving. The line siever fixes a value of y (from $y = 1, 2, \ldots$ up to some bound) and finds values of x for which both $F_1(x, y)$ and $F_2(x, y)$ are smooth. The lattice siever fixes a prime q, called the special-q, which divides $F_1(x, y)$, and finds (x, y) pairs for which both $F_1(x, y)/q$ and $F_2(x, y)$ are smooth. This is carried out for many special-q's. Lattice sieving was introduced by Pollard [20] and the code we used is the implementation described in [12, 6], with some additions to handle skew sieving regions efficiently.

For the lattice sieving, a rational factor base of 250 000 elements (the primes $\leq$ 3 497 867) and an algebraic factor base of 800 000 elements (ideals of norm

$\leq 12\ 174\ 433$) were chosen. For the line sieving, larger factor base bounds were chosen, namely: a rational factor base consisting of the primes $< 8\ 000\ 000$ and an algebraic factor base with the primes $< 16\ 777\ 216 = 2^{24}$. For both sieves the large prime bounds were 500 000 000 for the rational primes and 1 000 000 000 for the algebraic primes. The lattice siever allowed two large primes on each side, in addition to the special-q input. The line siever allowed three large primes on the algebraic side (this was *two* for RSA–130) and two large primes on the rational side.

The special-q's in the lattice siever were taken from selected parts of the interval $[12\ 175\ 000, 91\ 000\ 000]$ and a total of 2 361 390 special-q's were handled. Lattice sieving ranged over a rectangle of 8192 by 4000 points per special-q, i.e., a total of about $7.7 \cdot 10^{13}$ points. Averaged over all the workstations and PCs on which the lattice siever was run, about 52 seconds were needed to handle one special-q and about 16 relations were found per special-q. So on average the lattice siever needed 3.25 CPU seconds to generate one relation.

Line sieving ranged over most of $|x| < 9\ 000\ 000\ 000$ and $1 \leq y \leq 70\ 000$, about $1.2 \cdot 10^{15}$ points. It would have been better to reduce the bound on x and raise the bound on y, in accordance with skewness 4000, but we overestimated the amount of line sieving needed. 30% of the relations found with the line-siever had three large primes. Averaged over all the workstations and PCs on which the line siever was run, it needed 5.1 CPU seconds to generate one relation.

A fair comparison of the performances of the lattice and the line siever is difficult for the following reasons: memory requirements of the two sievers are different; the efficiency of both sievers decreases – but probably not with the same "speed" – as the sieving time increases; the codes which we used for lattice and line sieving were optimized by different persons (Arjen Lenstra, resp. Peter Montgomery).

A total of 68 500 867 relations were generated, 56% of them with lattice sieving (indicated below by "LA"), 44% with line sieving (indicated by "LI").

Sieving was done at five different locations with the following contributions:

- 36.8 % Peter L. Montgomery, Stefania Cavallar, Herman J.J. te Riele, Walter M. Lioen (LI, LA at CWI, Amsterdam, The Netherlands)
- 28.8 % Paul C. Leyland (LA at Microsoft Research Ltd, Cambridge, UK)
- 26.6 % Bruce Dodson (LI, LA at Lehigh University, Bethlehem, PA, USA)
- 5.4 % Paul Zimmermann (LA at Médicis Center, Palaiseau, France)
- 2.5 % Arjen K. Lenstra (LA at Citibank, Parsippany, NJ, USA, and at the University of Sydney, Australia)

Sieving started the day before Christmas 1998 and was completed one month later. Sieving was done on about 125 SGI and Sun workstations running at 175 MHz on average, and on about 60 PCs running at 300 MHz on average. The total amount of CPU time spent on sieving was 8.9 CPU-years. We estimate this to be equivalent to 2000 MIPS years. For comparison, RSA–130 took about 1000 MIPS years. Practical experience we collected with factoring large RSA–numbers tells us that with a careful tuning of the parameters the sieving times may be reduced now to 1000 resp. 500 MIPS years. The relations were collected at CWI and required 3.7 Gbytes of disk storage.

3.3 Filtering and finding dependencies

The filtering of the data and the building of the matrix were carried out at CWI and took one calendar week.

Filtering. Not all the sieved relations were used for filtering since we had to start the huge job for finding dependencies at a convenient moment. We actually used 65.7M of the 68.5M relations as filter input.

First, the "raw" data from the different contributing sites were searched through for duplicates. This single-contributor cleaning removed 1.4M duplicates. Next, we collected all the relations and eliminated duplicates again. This time, 9.2M duplicates were found. The 1.4 + 9.2M duplicates came from machine and human error (e.g., the resumption of early aborted jobs resp. duplicate jobs), from the simultaneous use of the lattice and the line siever, and from the line siever and the lattice siever themselves.

In the filter steps which we describe next, we only considered prime ideals with norm larger than 10 million; in the sequel, we shall refer to these ideals as the *large prime ideals*. In the remaining 55.1M relations we counted 54.1M large prime ideals. We added 0.1M free relations (cf. [11, Sect. 4, pp. 234–235]). Taking into account another 1.3M prime ideals with norm *below* 10 million, it seemed that we did not have enough relations at this point. However, after we removed 28.5M so-called *singletons* (i.e., relations which contain a large prime ideal that does *not* appear in any other relation) we were left with 26.7M relations having 21.5M large prime ideals. So now we had more than enough relations compared with the total number of prime ideals. We deleted another 17.6M relations which were heuristically judged the least useful[2], or which became singletons after we had removed some other relations. We were finally left with 9.2M relations containing 7.8M large prime ideals. After this, relations with large prime ideals occurring twice were merged (6.0M relations left) and, finally, those occurring three times were merged (4.7M relations left).

Finding dependencies. The resulting matrix had 4 671 181 rows and 4 704 451 columns, and weight 151 141 999 (32.36 nonzeros per row). With the help of Peter Montgomery's Cray implementation of the block Lanczos algorithm (cf. [17]) it took almost 100 CPU-hours and 810 Mbytes of central memory on the Cray C916 at the SARA Amsterdam Academic Computer Center to find 64 dependencies among the rows of this matrix. Calendar time for this job was five days.

[2] The criterion used for this filter step will be described in a forthcoming report [4] .

3.4 The square root step

During February 1–2, 1999, four square root (cf. [16]) jobs were started in parallel on four different 250 MHz processors of CWI's SGI Origin 2000, each handling one dependency. Each had about 5 million (not necessarily distinct) $a - b\alpha$ terms in the product. After 14.2 CPU-hours, one of the four jobs stopped, giving the two prime factors of RSA–140. Two others also expired with the two prime factors after 19 CPU-hours (due to different input parameter choices). One of the four jobs expired with the trivial factors.

We found that the 140–digit number

RSA–140 =

$$2129024631825875754749788201627151749780670396327721627823338321538194\backslash$$
$$9984056495911366573853021918316783107387995317230889569230873441936471$$

can be written as the product of the two 70–digit primes:

$$p = 3398717423028438554530123627613875835633986495969597423490929302771479$$

and

$$q = 6264200187401285096151654948264442219302037178623509019111660653946049.$$

Primality of the factors was proved with the help of two different primality proving codes [2, 5]. The factorizations of $p \pm 1$ and $q \pm 1$ are given by

$p - 1 =$
$2{\cdot}7{\cdot}7649{\cdot}435653{\cdot}396004811{\cdot}183967535370446691250943879126698812223588425357931$
$p + 1 =$
$2^3 3^2 5{\cdot}13{\cdot}8429851{\cdot}33996935324034876299{\cdot}2534017077123864320746970114544624627539$

$q - 1 =$
$2^6 61 \cdot 135613 \cdot 3159671789 \cdot 3744661133861411144034292857028083085348933344798791$
$q + 1 =$
$2{\cdot}3{\cdot}5^2 389{\cdot}6781{\cdot}982954918150967{\cdot}1610636079665429174500735839132880759077996886 9$

Acknowledgements. Acknowledgements are due to the Dutch National Computing Facilities Foundation (NCF) for the use of the Cray C916 supercomputer at SARA, and to (in alphabetical order) CWI, Lehigh University, the Magma Group of John Cannon at the University of Sydney, the Médicis Center at École Polytechnique (Palaiseau, France), and Microsoft Research Ltd (Cambridge, UK), for the use of their computing resources.

References

1. D. Atkins, M. Graff, A.K. Lenstra, and P.C. Leyland. THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in Cryptology – Asiacrypt '94*, volume 917 of *Lecture Notes in Computer Science*, pages 265–277, Springer-Verlag, Berlin, 1995.

2. Wieb Bosma and Marc-Paul van der Hulst. *Primality proving with cyclotomy.* PhD thesis, University of Amsterdam, December 1990.
3. J.P. Buhler, H.W. Lenstra, Jr., and Carl Pomerance. Factoring integers with the number field sieve. Pages 50–94 in [13].
4. S. Cavallar. Strategies for filtering in the Number Field Sieve. In preparation.
5. H. Cohen and A.K. Lenstra. Implementation of a new primality test. *Mathematics of Computation*, 48:103–121, 1987.
6. James Cowie, Bruce Dodson, R.-Marije Elkenbracht-Huizing, Arjen K. Lenstra, Peter L. Montgomery, and Jörg Zayer. A world wide number field sieve factoring record: on to 512 bits. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – Asiacrypt '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 382–394, Springer-Verlag, Berlin, 1996.
7. CREST. Visit `http://www.crestco.co.uk/`.
8. T. Denny, B. Dodson, A.K. Lenstra, and M.S. Manasse, On the factorization of RSA–120. In D.R. Stinson, editor, *Advances in Cryptology – Crypto '93*, volume 773 of *Lecture Notes in Computer Science*, pages 166–174, Springer-Verlag, Berlin, 1994.
9. B. Dodson and A. K. Lenstra. NFS with four large primes: an explosive experiment. In D. Coppersmith, editor, *Advances in Cryptology – Crypto '95*, volume 963 of *Lecture Notes in Computer Science*, pages 372–385, Springer-Verlag, Berlin, 1995.
10. Marije Elkenbracht-Huizing. *Factoring integers with the number field sieve.* PhD thesis, Leiden University, May 1997.
11. R.-M. Elkenbracht-Huizing. An implementation of the number field sieve. *Experimental Mathematics*, 5:231–253, 1996.
12. R. Golliver, A.K. Lenstra, and K.S. McCurley. Lattice sieving and trial division. In Leonard M. Adleman and Ming-Deh Huang, editors, *Algorithmic Number Theory, (ANTS-I, Ithaca, NY, USA, May 1994)*, volume 877 of *Lecture Notes in Computer Science*, pages 18–27, Springer-Verlag, Berlin, 1994.
13. A.K. Lenstra and H.W. Lenstra, Jr., editors. *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1993.
14. A.K. Lenstra and M.S. Manasse. Factoring by Electronic Mail. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology – Eurocrypt '89*, volume 434 of *Lecture Notes in Computer Science*, pages 355–371. Springer-Verlag, Berlin, 1990.
15. A.K. Lenstra and M.S. Manasse. Factoring with two large primes. In I.B. Dåmgard, editor, *Advances in Cryptology – Eurocrypt '90*, volume 473 of *Lecture Notes in Computer Science*, pages 72–82. Springer-Verlag, Berlin, 1991.
16. Peter L. Montgomery. Square roots of products of algebraic numbers. In Walter Gautschi, editor, *Mathematics of Computation 1943–1993: a Half-Century of Computational Mathematics*, pages 567–571. Proceedings of Symposia in Applied Mathematics, American Mathematical Society, 1994.
17. Peter L. Montgomery. A block Lanczos algorithm for finding dependencies over $GF(2)$. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology – Eurocrypt '95*, volume 921 of *Lecture Notes in Computer Science*, pages 106–120, Springer-Verlag, Berlin, 1995.
18. Peter L. Montgomery and Brian 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.
19. B. Murphy. Modelling the Yield of Number Field Sieve Polynomials. J. Buhler, editor, *Algorithmic Number Theory, (Third International Symposium, ANTS-III,*

Portland, O, USA, June 1998), volume 1423 of *Lecture Notes in Computer Science*, pages 137–151, Springer-Verlag, Berlin, 1998.
20. J.M. Pollard. The lattice sieve. Pages 43–49 in [13].
21. Carl Pomerance. The Quadratic Sieve Factoring Algorithm. In T. Beth, N. Cot and I. Ingemarsson, editors, *Advances in Cryptology – Eurocrypt '84*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182, Springer-Verlag, New York, 1985.
22. R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21:120–126, 1978.
23. RSA Challenge Administrator. In order to obtain information about the RSA Factoring Challenge, send electronic mail to `challenge-info@rsa.com` and visit `http://www.rsa.com/rsalabs/html/factoring.html`.
24. A. Shamir. Factoring Large Numbers with the TWINKLE device. Manuscript, April, 1999.