A World Wide Number Field Sieve Factoring Record: On to 512 Bits

James Cowie[1], Bruce Dodson[2], R. Marije Elkenbracht-Huizing[3], Arjen K. Lenstra[4], Peter L. Montgomery[5], Jörg Zayer[6]

[1] Cooperating Systems Corporation, 12 Hollywood Drive, Chestnut Hill, MA 02167, U. S. A. E-mail: cowie@mumonkan.cooperate.com
[2] Department of Mathematics, Lehigh University, Bethlehem, PA 18015-3174, U. S. A. E-mail: bad0@lehigh.edu
[3] Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. E-mail: marije@cwi.nl
[4] Citibank, N.A., 4 Sylvan Way, Parsippany, NJ 07054, U. S. A. E-mail: arjen.lenstra@citicorp.com
[5] 780 Las Colindas Road, San Rafael, CA 94903-2346, U. S. A. E-mail: pmontgom@cwi.nl
[6] Gartenstrasse 13, 66352 Dorf im Warndt, Germany. E-mail: j.zayer@ids-scheer.de

Abstract. We present data concerning the factorization of the 130-digit number RSA130 which we factored on April 10, 1996, using the Number Field Sieve factoring method. This factorization beats the 129-digit record that was set on April 2, 1994, by the Quadratic Sieve method. The amount of computer time spent on our new record factorization is only a fraction of what was spent on the previous record. We also discuss a World Wide Web interface to our sieving program that we have developed to facilitate contributing to the sieving stage of future large scale factoring efforts. These developments have a serious impact on the security of RSA public key cryptosystems with small moduli. We present a conservative extrapolation to estimate the difficulty of factoring 512-bit numbers.

1 Introduction

Over the past years several new record integer factorizations have been reported at cryptology conferences. At Eurocrypt'89 a record 100-digit factorization was announced; the result was obtained by running an existing factoring algorithm, the *Quadratic Sieve* method (QS), using a novel *factoring by email* approach (cf. [13]). At Eurocrypt'90 that approach was improved by a new *double large prime* variation of QS (cf. [14]): at the conference a record 107-digit factorization was reported, but the same approach led to a record 116-digit factorization in the journal version of the conference paper, later to a record 120-digit factorization at Crypto'93 (cf. [4]) and ultimately to a record 129-digit factorization at Asiacrypt'94 (cf. [1]).

In this paper we present a new 130-digit integer factoring record. Although our result may seem a marginal improvement over the previous record, the new record is in fact a dramatic step forward. In the first place, the record has been achieved by the *Number Field Sieve* factoring method (NFS) in only a fraction of the amount of computer time spent on the previous record. This shows that NFS is superior to QS for numbers of approximately 130 digits and larger, as far as current implementations are concerned. This had been suspected for a while (cf. [2]); experiments described in [6] suggest, however, that the crossover point between QS and NFS lies considerably lower than 130 digits.

In the second place, part of the work was carried out using a novel *World Wide Web interface* to our NFS implementation. Compared to the old factoring by email approach, the new interface makes it much easier to contribute to large factoring tasks: anybody with access to the World Wide Web is only a few mouse clicks away from becoming a factoring contributor. Because many processors on the World Wide Web have only limited memory, this required a new, more flexible NFS implementation, on top of the design and implementation of some necessary *Common Gateway Interface* scripts.

By using their Web browsers to complete fill-out forms, users invoke simple server-side CGI scripts written in Perl. A registration script, for example, creates appropriate entries in a distributed database of contributors, including a "privacy level" that allows contributors to request partial or total anonymity within the sieving effort. "Status" scripts allowed users to glimpse the latest snapshot of the sieving progress, and determine their standing within the "Hall of Fame" of major contributors. Still other Web pages and CGI scripts offered password-protected administrative services, input range checkout, relation set checkin, online tutorial information, and so forth.

Internet-distributed computing efforts are especially prone to transient connectivity failures; it is not uncommon for sites to disappear from view for minutes or hours at a time. A Web-based sieving effort centered around a central Web server would have imposed severe performance bottlenecks, as well as making the global sieving process prone to single-point failure. To promote scalability and fault-tolerance, additional CGI scripts made it possible to "check out" personalized copies of the Web server software, hyperlinked back to the originating server. These derivative subservers came complete with a large initial subrange of inputs, plus hierarchical protocols for refreshing the server's pool of inputs. As a result, only a prolonged failure (on the order of tens of hours) of the root of the server tree—the primary Web server at Cooperating Systems—would have brought the global sieving process to a halt.

We feel confident that a global Web-based computational environment, guaranteeing the anonymity of participants and based on "best-effort" coordination of contributed resources, can easily scale to hundreds of thousands of globally distributed participants. The practical consequences of these developments for the security of 512-bit RSA-moduli are interesting, as we show in Section 2. Section 3 gives background information on the 130-digit factorization. We conclude with a description of the Web-interface in Section 4.

2 On the security of 512-bit RSA-moduli

Both QS and NFS consist of three major steps: a *sieving step* to collect a set of data, a *matrix step* to find dependencies among the data, and a *final step* where the dependencies are used to derive a factorization. The final step of neither QS nor NFS is considered to be problematic: for QS because it is an entirely straightforward computation, for NFS because of the method developed by Peter L. Montgomery (cf. [15]). In this section we therefore restrict our attention to the sieving and matrix steps.

Let

$$L_x[u, v] = \exp(v(\ln x)^u(\ln \ln x)^{1-u}).$$

It is well known that the sieving and the matrix step of QS both run in heuristic expected time $L_n[1/2, 1 + o(1)]$, for $n \to \infty$, with n the number to be factored (cf. [11]). For numbers in our current range of interest, however, the sieving step takes considerably more time than the matrix step: for the factorization reported in [1] for instance less than 0.3% of the total effort was spent on the matrix step. Large scale QS factoring efforts have shown that the run time estimate $L_n[1/2, 1 + o(1)]$ with $o(1) = 0$ can be used for limited range extrapolations (cf. [1, 4, 14]). Thus, we may expect that the sieving step for the QS-factorization of a 512-bit RSA-modulus would require about one hundred times the effort spent on the 129-digit QS record.

The sieving and the matrix step of NFS both run in heuristic expected time $L_n[1/3, (64/9)^{1/3} + o(1)]$, for $n \to \infty$ (cf. [12]). Asymptotically this is substantially faster than QS. Although the sieving step is still the most time consuming part of the computation, for NFS the difference is much smaller than for QS: for a 119-digit NFS factorization reported in [6], the matrix step took about 7% of the entire computation. Assuming that the run time estimate $L_n[1/3, (64/9)^{1/3} + o(1)]$ can again, with $o(1) = 0$, be used for limited range extrapolations (our results combined with the ones from [6] support this assumption), we find that the sieving step for the NFS-factorization of a 512-bit RSA-modulus would require only about thirty times the effort spent on our current 130-digit NFS record. Since the sieving effort spent on our 130-digit record ould have been less than 15% (as shown below) of the effort spent on the preious 129-digit record, we conclude that NFS-sieving for a 512-bit number could ake less than 5 times the effort spent on the old 129-digit record, and less than one twentieth of the time required for QS. We stress that these comparisons only hold for current QS and NFS implementations—both might be improved in the future.

This is a disturbingly small security margin for 512-bit RSA-moduli. Several things have to be kept in mind, however. In the first place, this estimate addresses only the sieving step. Although we passed the crossover point between the sieving steps of QS and NFS, we have not reached it yet for the matrix steps—the NFS matrix for our 130-digit number had about seven times as many rows and columns as the QS matrix for the 129-digit record. The matrix step for a 512-bit NFS-factorization would without any doubt pose an interesting new challenge.

Right now this challenge looks hard, but certainly not unsurmountable. Secondly, the estimate does not incorporate possible improvements in the NFS-sieving step that are currently under consideration, and that could have a substantial impact on the sieving time. Finally, we want to reiterate that it is unwise to make predictions about the difficulty of factoring (cf. [1,9]).

3 Factoring RSA130

RSA130. Until April 10, 1996, RSA130 was the smallest unfactored number on the 'RSA challenge list' of d-digit composite numbers, for $d = 100, 110, 120, \ldots, 490, 500$. This list was compiled by RSA Data Security Corporation Inc. in the following manner (cf. [18]):

> Each RSA number is the product of two randomly chosen primes of approximately the same length. These primes were both chosen to be congruent to 2, modulo 3, so that the product could be used in an RSA public-key cryptosystem with public exponent 3. The primes were tested for primality using a probabilistic primality testing routine. After each product was computed, the primes were discarded, so no one—not even the employees of RSA Data Security—knows any product's factors.

RSA100 was factored in April 1991, RSA110 in April 1992 (cf. [5]), and RSA120 in June 1993 (cf. [4]). All these factorizations were achieved using the Quadratic Sieve factoring method. In this section we discuss our Number Field Sieve factorization of RSA130:

$$\begin{aligned}\text{RSA130} = {} & 18070\ 82088\ 68740\ 48059\ 51656\ 16440\ 59055\ 66278\ 10251\ 67694\ 01349\ 17012\ 70214 \\ & 50056\ 66254\ 02440\ 48387\ 34112\ 75908\ 12303\ 37178\ 18879\ 66563\ 18201\ 32148\ 80557.\end{aligned}$$

We assume that the reader is familiar with NFS (cf. [12]).

Polynomial selection. Let $n = \text{RSA130}$. The first step of NFS is to select at least two irreducible polynomials with integer coefficients and a common root modulo n, the number to be factored. Although using more than two polynomials may turn out to be more efficient (cf. [8]), we used only two, as in [6,10,12]. In those references the choice of polynomials is restricted to the case where the common root m is an integer close to $n^{1/(d+1)}$ for some small integer d (such as 4 or 5); the polynomials can then be chosen as $f_1(X) = X - m$ and $f_2(X) = \sum_{i=0}^{d} c_i X^i$, where $n = \sum_{i=0}^{d} c_i m^i$ with $-m/2 \leq c_i \leq m/2$ is a base m representation of n.

For $j = 1, 2$ and integers a, b, let $N_j(a,b) = f_j(a/b)b^{\text{degree}(f_j)} \in \mathbf{Z}$. *Relations*, the type of data we collect during the sieving, are defined as pairs of coprime integers a, b, with $b > 0$, such that $N_j(a,b)$ has only prime factors $\leq B_j$, for $j = 1, 2$, and for appropriately chosen bounds B_1 and B_2. Thus, f_1 and f_2 should be chosen such that the N_j's have a relatively high probability to yield relations. At the moment only ad hoc strategies are known, which are not proven to construct and select the best polynomials possible. For RSA130 we did the following. Scott Huddleston from Oregon State University provided us with a

Table 1

#	yield	norm	B: 25	10^2	10^3	10^4	10^5
			9	25	168	1229	9592
14	100.0%	299.1%(14)	9(3)	25(3)	174(2)	1246(3)	9714(2)
4	99.1%	231.1%(10)	11(1)	25(4)	192(1)	1287(2)	9751(1)
1	93.7%	100.0%(1)	7(7)	19(12)	174(3)	1187(10)	9595(7)
12	87.5%	320.2%(15)	5(10)	24(5)	158(10)	1214(6)	9496(14)
8	82.2%	262.5%(11)	8(4)	24(6)	156(11)	1205(7)	9531(11)
3	80.0%	104.7%(2)	7(8)	21(9)	170(4)	1172(14)	9520(12)
10	77.8%	222.1%(9)	6(9)	27(1)	162(7)	1225(5)	9689(4)
2	76.8%	122.7%(3)	8(5)	21(10)	163(5)	1177(13)	9603(6)
11	76.6%	275.2%(13)	5(11)	23(8)	163(6)	1318(1)	9573(8)
15	75.9%	183.7%(7)	8(6)	26(2)	159(9)	1146(15)	9535(9)
9	75.4%	274.2%(12)	10(2)	20(11)	143(13)	1186(11)	9417(15)
5	70.1%	159.8%(5)	4(12)	18(13)	160(8)	1195(8)	9511(13)
7	64.9%	152.0%(4)	3(13)	18(14)	132(15)	1185(12)	9708(3)
13	64.2%	183.2%(6)	3(14)	24(7)	139(14)	1242(4)	9605(5)
6	57.8%	221.4%(8)	3(15)	14(15)	151(12)	1191(9)	9534(10)

list of 15 pairs of polynomials as above with $d = 5$ (i.e., each pair consisting of one linear and one degree 5 polynomial), ranked from 1 (best) to 15 (worst) depending on his experimental "goodness" measure that depends on the average size of the N_j's over some sieving rectangle.

The ranking that is most relevant for practical purposes is the actual yield in the sieving step. This is however rather expensive to compute. We did sieving experiments with all 15 pairs (using 500 *special q*'s, see below), and found that the pair that was the second worst (number 14) in Huddleston's ranking was most productive. In Table 1 the yields of all pairs are given in the second column, sorted by their relative yield compared to pair number 14. The number in the first column refers to the ranking given by Huddleston. For 7 of the 15 we did more sieving (a total of 800 special q's), confirming the earlier ordering, but giving slightly different relative percentages.

Assuming that a ranking is a randomly chosen permutation of $\{1, 2, \ldots, 15\}$, we define the *correlation coefficient* of two rankings x and y as $(E(xy) - E(x)E(y))/(\sigma(x)\sigma(y))$, with E denoting the expected value, and

$$\sigma(x) = \sqrt{E(x^2) - E(x)^2}.$$

Note that for all rankings x we have that $E(x) = 8$ and $\sigma(x) = \sqrt{56/3}$. The correlation coefficient of Huddleston's and the sieving rank is 0.1. This indicates that Huddleston's goodness measure does not effectively predict the sieving yield. It might be interesting to have a closer look at some of the polynomials he rejected.

In Table 1 we give several other rankings that are easy to compute and that might be useful to predict the sieving yield. The *norm* of a pair $(X - m, \sum_{i=0}^{5} c_i X^i)$ is defined as $m \sum_{i=0}^{5} |c_i|$. In the past rankings based on the norm

have often been used. For the 15 candidates the norms relative to the pair with smallest norm are given in column 3, with the resulting norm rank in parentheses. The correlation coefficient of this norm rank and the sieving rank is -0.26. We conclude that the norm does not lead to a useful ranking. Columns 4 through 8 give the number of roots of the second polynomial of each pair modulo all primes $< B$, for $B = 25$, 10^2, 10^3, 10^4, and 10^5, with the resulting ranking between parentheses: more roots are supposedly better and therefore give a lower ranking (the second row contains the values of $\pi(B)$). The resulting correlation coefficients with the sieving rank are: 0.69, 0.54, 0.77, 0.29, and 0.24. Some of these rankings are well correlated with the sieving rank, but they are not reliable predictors: polynomial number 11, ranked 9th for sieving, ranks first for $B = 10^4$. The ranking induced by the average root ranking has correlation coefficient 0.75 with the sieving ranking.

We also found the much better ranking 2, 1, 5, 7, 10, 3, 4, 6, 11, 8, 9, 13, 14, 12, 15 (i.e., Huddleston's number 14 got ranked second, his number 4 got ranked first, etc., until his number 6 which got ranked last) with correlation coefficient 0.86 with the sieving rank. This ranking was obtained by combining the integral of $f_1 \cdot f_2$ over the sieving region with information about the number of real roots and roots modulo primes $< 10^4$, and by considering extreme residual values after sieving instead of average values. Details may be published at a later occasion. We leave the problem of polynomial selection for NFS as a subject for further study.

As a result of the sieving experiments, we decided to use the polynomial that ranked as number 14 on Huddleston's list: $d = 5$, $m = 125\,74411\,16841\,80059\,80468$, $f_1(X) = X - m$, and

$$\begin{aligned} f_2(X) = {} & 5748\,30224\,87384\,05200\,X^5 + 9882\,26191\,74822\,86102\,X^4 \\ & - 13392\,49938\,91281\,76685\,X^3 + 16875\,25245\,88776\,84989\,X^2 \\ & + 3759\,90017\,48552\,08738\,X - 46769\,93055\,39319\,05995. \end{aligned}$$

Sieving. To find relations we mostly [7] used *lattice sieving with sieving by vectors* as introduced in [17]. We followed the approach sketched and used in [10] and [6], with three important modifications. In the first place, unlike [17], but like [7] we allowed *special* q's in $N_2(a, b)$ that are larger than B_2. As usual, disjoint ranges of special q's were assigned to different processors, and for efficiency reasons the sizes of the q's depended on the amount of available memory. For implementation technical reasons the q's were bounded from above by $2 \cdot 36^5 \approx 1.2 \cdot 10^8$. Secondly, unlike [10] we allowed processors to use lower values for B_1 and B_2, depending on memory restrictions. In the third place, the physical size of the lattice sieving array depended on the size of the available memory. As a consequence of these changes, the sieving program could be distributed much easier over a variety of machines than the program from [10].

[7] We also did a relatively small amount of traditional line-by-line sieving.

Table 2

Memory	$q < B_2$ $q/10^6 \in$	$q \geq B_2$ $q/10^6 \geq$
4M		86
6M		43
8M	[0.5, 1.9]	29
10M	[1.9, 3.8]	21
12M	[3.8, 5.8]	17
14M	[5.8, 7.9]	14
16M	[7.9, 10.0]	12
$\geq$ 18M	[2, 11.4]	2

We describe this set-up in more detail. Let $B_1 = 3497867$ and $B_2 = 11380951$. The number of roots of f_j modulo the primes $\leq B_j$ is 250001 for $j = 1$ and 750001 for $j = 2$. All processors that have at least 18 megabytes available to the siever used this B_1 and B_2; all available memory that remained after storing all relevant factor base data for these B_1 and B_2 was used for the sieving array. As a result 18 megabyte machines used a sieve of at least 2 megabytes: about 2 megabytes if the special q is $> B_2$, more than 2 megabytes if $q < B_2$, because in that case only the (prime, root) pairs from the second factor base for which the prime is at most q need to be stored (so that any remaining memory could be assigned to the sieve array as well).

If less than 18 megabytes was available, we did the following. At least 2 megabytes was used for the sieve. If the special q is $< B_2$, the complete first factor base was used, and all (prime, root) pairs from the second factor base with the prime $\leq q$. This meant that not all $q < B_2$ could be used; Table 2 gives the ranges of q's less than B_2 that we used (and that fit) on machines of various sizes; the q's were also bounded from below to make sure that at least some q's were available for smaller machines. Special q's less than B_2 were not used on machines with at most 6 megabytes. If the special q is $\geq B_2$, only $[x \cdot 250001]$ pairs from the first factor base and $[x \cdot 750001]$ pairs from the second factor base were used, for the largest positive $x \leq 1$ such that all relevant data fit in memory. So, processors with small memories used rather small factor bases, and were therefore not very productive. We therefore reserved the smaller (and better) special q's ($\geq B_2$) for the larger machines; Table 2 gives the lower bounds that we used for $q \geq B_2$ on machines of various sizes. In both cases any remaining memory was used for the sieve as well. Using this set-up the more than 6 million special q's that were available could be distributed over the available processors (and memory) without running out of sieving tasks and without wasting good q's on small machines.

Sieving time. Because sieving was done on machines of many different sizes, and because the number of relations found per second strongly depends on the memory size, it is hard to estimate how much time was spent on the sieving step. We can say, however, that the first $\approx 25 \cdot 10^5$ special q's would have generated about $7 \cdot 10^7$ relations, with at most 20% duplicates (and therefore at least $56 \cdot 10^6$

unique relations, which sufficed; see below). Since we sieved over the rectangle $[-4096, 4095] \times [1, 4000]$ not including the 'even,even' locations, inspecting a total of $\approx 6 \cdot 10^{13}$ sieve locations would have sufficed, with approximately one of every 860000 locations producing a relation. On a Sparc 10 workstation with 24 megabytes available for the sieving process, this would have taken 16.5 years. For 32, 40, and 48 megabytes these timings improve to 15.4, 14.7, and 14.5 years, respectively, but for 20, 16, and 12 megabytes it deteriorates to 18.5, 25, and 32 years, respectively. On the same workstation sieving for the 129-digit QS record would have taken approximately 120 years (cf. [1]). All these timings are $\approx 24\%$ better on a 90Mhz Pentium PC. It is therefore fair to say that sieving for RSA130 could have been done in less than 15% of the time spent on the 129-digit number.

Cycles. As usual we refer to the relations introduced above as *full relations* and the relations with one or more *large primes* $> B_j$ in the factorization of $N_j(a, b)$ as *partial relations*. With $B_1 = 3497867$ and $B_2 = 11380951$ the number of full relations combined with the number of *cycles* (cf. [6]) among the partial relations should be comfortably more than $250001 + 750001 \approx 10^6$ before the factorization can be completed (cf. [12]). The number of cycles can be counted by finding the *useful* partial relations (i.e., the largest subset of the partials in which each large prime occurs at least twice), and by subtracting the resulting number of large primes from the number of usefuls. In Table 3 it can be seen how the number of usefuls and cycles at first slowly increases as a function of the number of partials (after all duplicates had been removed), and how first the number of usefuls and then the number of cycles suddenly grows quite rapidly, as illustrated by the average cycle length (#usefuls/#cycles) in the last column. This was expected based on the data from [6]. The number in row labeled i and column labeled j of Table 4 gives the number of relations with i large primes in $N_1(a, b)$ and j large primes in $N_2(a, b)$, in the final collection of 56515672 relations. The relations referred to in columns labeled 4, 5, and 6, and those in rows labeled 4 and 5 were found early in the sieving stage when one of the authors was using slightly larger B_1 and B_2 than given above. In uncompressed format, listing only the primes $> 2 \cdot 10^6$ per $N_i(a, b)$, the final collection took about 3.5 gigabytes of disk space.

To make the amounts of data more manageable, we first extracted 8426508 partial relations leading to the shortest 968737 cycles (note that $48400 + 968737 = 1017137 > 10^6$ as required). These data could have been used to build a matrix of 1017137 rows (corresponding to the fulls and the cycles) and 1000002 columns (corresponding to the primes $\leq B_i$ in $N_i(a, b)$). As shown in [6: Section 5], however, removing all large primes from the matrix makes it relatively dense and thus expensive to process the matrix. We therefore also included columns in the matrix for large primes that occur at least 4 times in the collection of 8426508 relations. This led to 826592 and 1678272 additional columns for large primes in $N_1(a, b)$ and $N_2(a, b)$, respectively, 1527391 new 'full relations' (i.e., partial relations with only large primes corresponding to the new columns), and 1946210 cycles among the remaining partial relations. The resulting matrix has

Table 3

Date	#fulls	#partials	#usefuls	#cycles	length
95 08 30	4427	2524973	365	179	2.0
95 09 20	21407	8288179	4021	1880	2.1
95 10 14	36434	16756214	14202	6365	2.2
95 11 05	41248	26872640	26450	10553	2.5
95 11 24	44031	35016001	37969	14177	2.7
95 12 10	45653	41319347	47660	16914	2.8
95 12 19	46648	45431262	8214349	224865	36.5
96 01 06	48211	53282421	11960120	972121	12.3
96 01 14	48400	56467272	18830237	2844859	6.6

Table 4

	0	1	2	3	4	5	6
0	48400	479737	1701253	1995537	6836	403	9
1	272793	2728107	9617073	11313254	39755	2212	44
2	336850	3328437	11520120	13030845	56146	3214	71
3	1056	9022	24455	0	0	0	0
4	3	9	31	0	0	0	0

$1000002+826592+1678272 = 3504866$ columns and $48400+1527391+1946210 = 3522001$ rows. Note that $3522001 - 3504866 = 1017137 - 1000002$, i.e., the larger matrix is over-square by the same amount as the original matrix, as expected.

A matrix with more than $3.5 \cdot 10^6$ rows and columns is rather large, which makes the matrix step difficult. Also, we may expect $\approx (8426508 + 48400)/2$ relations per dependency, which makes the final step quite expensive too. It is likely that if we had collected more relations, far fewer than 8426508 partial relations would have sufficed to generate the required $> 10^6 - 48400$ cycles, thus making the matrix and final steps easier. In future factorizations this strategy should probably be used to keep the matrix size within reasonable bounds. Also one could try the filtering strategy described in [7], which might give better results than the approach followed here.

The matrix step. After substituting *free relations* (cf. [12]) for the 7000 heaviest rows, and removing some of the excess heavy rows, the above 3522001×3504866 matrix resulted in a 3516502×3504823 bit matrix of total weight 138690744 (on average 39.4 entries per row). To find dependencies modulo 2 among the rows of this matrix we used the *blocked Lanczos* method from [16] on a Cray C-90 supercomputer. Blocked Lanczos consists of a sequence of multiplications of the matrix and its transpose with a *blocked vector*, i.e., b bit-vectors simultaneously, where b is an implementation dependent *blocking factor*. The number of multiplications needed is, approximately, $m/(b - 0.76)$ for an $m \times m$ bit matrix. The best configuration we managed to find on the Cray C-90 was $b = 64$ which resulted in 4.37 seconds per iteration.

As a result it took 67.5 CPU-hours (using a single processor) and 700 megabytes central memory to find 18 dependencies. Each dependency consisted

Table 5

	0	1	2	3	4	5	6
0	24242	154099	330738	255742	1054	52	1
1	75789	443647	885136	648148	2734	164	2
2	56326	300369	565605	389046	1923	131	4
3	182	776	1105	0	0	0	0
4	2	4	7	0	0	0	0

of ≈ 4140000 relations (≈ 3500 of which are free). For one of the dependencies, the breakdown of the large primes amongst its 4140328 (non-free) relations is given in Table 5; this dependency also contained 3506 free relations.

The final step. Let S be the set of relations in a dependency, let α_2 be such that $f_2(\alpha_2) = 0$, and let φ be the homomorphism from $\mathbf{Z}[\alpha_2]$ to $\mathbf{Z}/n\mathbf{Z}$ that maps α_2 to m modulo n. From the matrix step we know that $\gamma_1 = \prod_{(a,b)\in S}(a - bm)$ is a square in $\mathbf{Z}$, say β_1^2, and that $\gamma_2 = \prod_{(a,b)\in S}(a - b\alpha_2)$ is a square in $\mathbf{Z}[\alpha_2]$, say β_2^2. Furthermore, we have that $\gamma_1 \equiv \varphi(\gamma_2) \bmod n$. In the final step we compute $\beta_1 \bmod n$ and $\varphi(\beta_2)$ and attempt to factor n by computing $\gcd(\beta_1 \bmod n - \varphi(\beta_2), n)$.

Computing $\beta_1 \bmod n$ is straightforward, because the prime factorization of all $a - bm$'s is known. To compute $\varphi(\beta_2)$ we used the method from [15]. Instead of γ_2, we work with $\prod_{(a,b)\in S}(a - b\alpha_2)^{\pm 1}$, where the exponents are chosen as 1 or -1 in an attempt to maximize the amount of cancellation between the numerator and the denominator. Note that the -1's can easily be incorporated in the previous argument. The resulting fraction for the product would, when written out, have approximately 9.7 million decimal digits. After about 10^5 iterations, each of which reduced the numerator or denominator of the product by about 200 decimal digits, the algebraic square root was reduced to the trivial square root of 1. This took 49.5 CPU-hours on a single 150 MHz R4400SC processor of an SGI Challenge. The first two dependencies resulted in the trivial factorization. The third one produced:

RSA130 = 39685 99945 95974 54290 16112 61628 83786 06757 64491 12810 06483 25551 57243
· 45534 49864 67359 72188 40368 68972 74408 86435 63012 63205 06960 09990 44599.

4 The Web-interface

Previous factoring efforts used electronic mail as a medium for simple sieving task distribution and relation collection. Since the 129-digit factorization reported at Asiacrypt'94 (cf. [1]), however, Web browser technology has achieved universal penetration of the desktop. Using the Common Gateway Interface (CGI) for providing simple scripted executable content, we constructed a package of Perl scripts that automated most of the work involved in constructing an ad hoc, global-scale sieving workgroup.

Web-factoring volunteers commonly located our Web server by way of a commercial search engine. From one set of Web pages, they could then access a wide range of support services for the sieving step of the factorization: NFS software distribution, project documentation, anonymous user registration, dissemination of sieving tasks, collection of relations, relation archival services, and real-time sieving status reports. We also constructed CGI scripts to support cluster management, directing individual sieving workstations through appropriate day/night sleep cycles to minimize the impact on their owners. Once volunteers downloaded and built the GNFSD sieving software daemon, the daemon automatically became a Web client, using the HTTP protocol to GET q-values from, and POST the resulting relations back to, a CGI script on the Web server.

Three factors combined to make this approach succeed. First, the flexibility of our NFS implementation allowed even single workstations with 4 megabytes to perform useful work using small bounds B_1 and B_2 and a small sieve. Second, we supported anonymous registration—users could contribute their hardware resources to the sieving effort without revealing their identity to anyone other than the local server administrator. Anonymity is a critical prerequisite for altruism, in order to protect contributors from having their donated resources attacked or misused as their reward.

Finally, we recruited other sites to run the CGI script package locally, forming a hierarchical network of RSA130 Web servers. The root server, located at Cooperating Systems, was fed large consecutive ranges of q's by hand; automated scripts then managed the fragmentation and distribution of these large tasks to sieving clients and other Web servers at Boston University and the Northeastern Parallel Architecture Center at Syracuse University. Those remote servers in turn partitioned the incoming q-ranges into manageable tasks for their local single-workstation sievers. These NFS sieving processes interacted directly with CGI scripts on the remote Web server to return their relations and receive a new task to work on; this automated dialog between NFS sieving clients and their host Web server allowed sieving to proceed around the clock with minimal human intervention.

Next Steps. Since we began the Web-based factoring project, Web technologies have taken another major step forward: the widespread availability of Java as a tool for donation and coordination of globally distributed resources. Existing CGI scripts, written in Perl, invented cumbersome protocols for preserving state between successive CGI invocations. By contrast, a Java-based HTTP server can offer integrated Java services, route successive service requests to previous invocations, and dynamically extend its functionality by loading new services at runtime. CGI performance is vastly improved, since instead of forking a new process per CGI request and loading the appropriate interpreter executable, the Java interpreter is already in core and executing when requests for service arrive. As a result, we are currently working on a new revision of the factoring software framework, combining Java-based Web servers, extensible applet interfaces, and NFS sieving code integrated via Java's native method interface.

This software framework will provide many additional management services that are common to globally coordinated volunteer computations, cutting the time required to ramp up and field future factoring projects. These services will include hooks to public-key encryption for interserver communication (authenticating tasks, results, and software upgrades), anonymous user database management, scalable implementations of general-purpose hierarchical multiserver task queues, improved fault-tolerance via distributed persistent state, and Web-based cluster management tools.

Global Implications. We made no formal announcement of the availability of our early prototype Web software. Nonetheless, by virtual word-of-mouth we attracted the interest of browsers from over 500 different Internet hosts; 20 percent of those hosts later participated in the sieving stage. These ranged from SLIP-connected home computers to high-performance corporate workstation clusters, and literally covered the globe. Browsers from 28 countries, from AT (Austria) to ZA (South Africa), left their prints on the RSA130 project.

In future large scale factoring efforts, therefore, we predict that Web technologies will allow us to easily recruit the spare cycles and aggregate memory of individual computers world-wide. We have already demonstrated that using the Internet to provide anonymous point-and-click accessibility to large computational problems is an effective mechanism for tackling problems of communal significance. Meanwhile, steady algorithmic improvements are making our sieving algorithms more flexible and amenable to global participation. As we have seen in Section 2, these developments seriously affect the security of 512-bit RSA-moduli.

Acknowledgments. Acknowledgments are due to to Scott Huddleston and Oregon State University for providing us with 15 candidate pairs of polynomials, and to the contributors to the World Wide Web sieving project. We especially thank our partners at Boston University and the Northeastern Parallel Architecture Center at Syracuse University for their help debugging several prereleases of the Web software, and for useful feedback. This work was sponsored by the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF) for the use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO). The fourth author's work on this paper was done at Bellcore.

References

1. D. Atkins, M. Graff, A.K. Lenstra, and P.C. Leyland, *THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE*, Advances in Cryptology, Asiacrypt'94, Lecture Notes in Comput. Sci. **917** (1995), 265–277.
2. D. J. Bernstein, A. K. Lenstra, A general number field sieve implementation, 103–126 in: [12]

3. J. Buchmann, J. Loho, J. Zayer, *An implementation of the general number field sieve*, Advances in Cryptology, Crypto '93, Lecture Notes in Comput. Sci. **773** (1994) 159–165.
4. T. Denny, B. Dodson, A. K. Lenstra, M. S. Manasse, *On the factorization of RSA-120*, Advances in Cryptology, Crypto '93, Lecture Notes in Comput. Sci. **773** (1994) 166–174.
5. B. Dixon, A. K. Lenstra, *Factoring integers using SIMD sieves*, Advances in Cryptology, Eurocrypt '93, Lecture Notes in Comput. Sci. **765** (1994) 28–39.
6. B. Dodson, A. K. Lenstra, *NFS with four large primes: an explosive experiment*, Advances in Cryptology, Crypto '95, Lecture Notes in Comput. Sci. **963** (1995) 372–385.
7. R. M. Elkenbracht-Huizing, *An implementation of the number field sieve*, Technical Report NM-R9511, Centrum voor Wiskunde en Informatica, Amsterdam, 1995; to appear in Experimental Mathematics.
8. R. M. Elkenbracht-Huizing, *A multiple polynomial general number field sieve*, Proceedings ANTS II, to appear.
9. M. Gardner, *Mathematical games, A new kind of cipher that would take millions of years to break*, Scientific American, August 1977, 120–124.
10. R. Golliver, A. K. Lenstra, K. McCurley, *Lattice sieving and trial division*, ANTS '94, Lecture Notes in Comput. Sci. **877** (1994) 18–27.
11. A. K. Lenstra, H. W. Lenstra, Jr., *Algorithms in number theory*, Chapter 12 in: J. van Leeuwen (ed.), *Handbook of theoretical computer science*, Volume A, *Algorithms and complexity*, Elsevier, Amsterdam, 1990.
12. A. K. Lenstra, H. W. Lenstra, Jr. (eds), The development of the number field sieve, Lecture Notes in Math. **1554**, Springer-Verlag, Berlin, 1993.
13. A. K. Lenstra, M. S. Manasse, *Factoring by electronic mail*, Advances in Cryptology, Eurocrypt '89, Lecture Notes in Comput. Sci. **434** (1990) 355–371.
14. A. K. Lenstra, M. S. Manasse, *Factoring with two large primes*, Advances in Cryptology, Eurocrypt '90, Lecture Notes in Comput. Sci. **473** (1991) 72–82; Math. Comp., **63** (1994) 785–798.
15. P. L. Montgomery, *Square roots of products of algebraic numbers*, Proceedings of Symposia in Applied Mathematics, Mathematics of Computation 1943-1993, Vancouver, 1993, Walter Gautschi, ed.
16. P. L. Montgomery, *A block Lanczos algorithm for finding dependencies over GF(2)*, Advances in Cryptology, Eurocrypt'95, Lecture Notes in Comput. Sci. **921** (1995) 106–120.
17. J. M. Pollard, The lattice sieve, 43–49 in: [12].
18. RSA Data Security Corporation Inc., sci.crypt, May 18, 1991; information available by sending electronic mail to challenge-rsa-list@rsa.com.