On the factorization of RSA-120

T. Denny[1], B. Dodson[2], A. K. Lenstra[3], M. S. Manasse[4]

[1] Lehrstuhl Prof. Buchmann, Fachbereich Informatik , Universität des Saarlandes, Postfach 1150, 66041 Saarbrücken, Germany
E-mail: `denny@cs.uni-sb.de`

[2] Department of Mathematics, Lehigh University, Bethlehem, PA 18015-3174, U.S.A
E-mail: `bad0@Lehigh.EDU`

[3] MRE-2Q334, Bellcore, 445 South Street, Morristown, NJ 07960, U.S.A
E-mail: `lenstra@bellcore.com`

[4] DEC SRC, 130 Lytton Avenue, Palo Alto, CA 94301, U.S.A
E-mail: `msm@src.dec.com`

Abstract. We present data concerning the factorization of the 120-digit number RSA-120, which we factored on July 9, 1993, using the quadratic sieve method. The factorization took approximately 825 MIPS years and was completed within three months real time. At the time of writing RSA-120 is the largest integer ever factored by a general purpose factoring algorithm. We also present some conservative extrapolations to estimate the difficulty of factoring even larger numbers, using either the quadratic sieve method or the number field sieve, and discuss the issue of the crossover point between these two methods.

On the factorization of RSA-120

Evaluation of integer factoring algorithms, both from a theoretical and practical point of view, is of great importance for anyone interested in the security of factoring-based public key cryptosystems. In this paper we concentrate on the practical aspects of factoring. Furthermore, we restrict ourselves to *general purpose* factoring algorithms, i.e., algorithms that do not rely on special properties the numbers to be factored or their factors might have. These are the algorithms that are most relevant for cryptanalysis.

Currently the two leading general purpose factoring algorithms are the *quadratic sieve* (QS) and the *number field sieve* (NFS), cf. [12] and [2]. Throughout this paper, NFS is the generalized version (from [2]) of the algorithm from [8]; the latter algorithm is much faster, but can only be applied to composites of a very special form, cf. [9]. Let

$$L_x[a,b] = \exp\left((b+o(1))(\log x)^a(\log\log x)^{1-a}\right)$$

for real a, b, x, and $x \to \infty$. To factor an odd integer $n > 1$ which is not a prime power, QS runs in time

$$L_n[1/2, 1], \qquad (1)$$

and NFS in

$$L_n[1/3, 1.923]. \tag{2}$$

It follows that NFS is asymptotically superior to QS, but for numbers in the 100 to 150-digit range it is not immediately obvious which of the two methods is faster. If the $o(1)$'s in both runtimes are set to zero, then the crossover occurs for n around 124 digits. This computation neglects many aspects affecting the practical runtimes of both methods, and is thus oversimplistic. Similarly, it does not make much sense to evaluate any of these expressions for some particular n (again with $o(1) = 0$), and to interpret the result as 'the number of instructions' needed to factor that n, as sometimes happens in the literature or on sci.crypt. They can be used, however, to predict how hard factoring m will be, when it is known how hard n is, and if n and m differ by not more than, say, 15 digits. Although such a prediction can be helpful, it is of only limited practical value, in particular if m is considerably larger than n or if n was already testing the limits of our capabilities. This is caused by the fact that the $o(1)$ is *not* a constant and by a variety of other issues that will be discussed below.

The QS-factorization of a 116-digit factor of $10^{142}+1$ was the largest general purpose factorization reported in the literature [11]. As mentioned in [11] various parameters for that factorization were deliberately chosen suboptimally, in order to keep memory requirements acceptable to the contributors of cycles (the authors of [11] use their 'electronic mail' approach (cf. [10]) to get the cycles necessary for this factorization, so they had every reason to avoid complaints from prospective contributors). The factorization was completed in approximately 400 MIPS years, which corresponds to roughly $400 \cdot 10^6 \cdot 365 \cdot 24 \cdot 3600 \approx 1.3 \cdot 10^{16}$ instructions. Application of the ill-advised practice described above to this 116-digit n would lead to an estimate of $L_n[1/2, 1] \approx 5.7 \cdot 10^{16}$ instructions, which is off by only a factor 4.4, and thereby unusually accurate.

In the present paper we consider the 120-digit number RSA-120. Until June 9, 1993, RSA-120 was the smallest unfactored number on the 'RSA challenge list,' which is a list of composite numbers of d digits, for $d = 100, 110, 120, \ldots,$ 490, 500. This list was compiled by RSA Data Security Corporation Inc. in the following manner (cf. [14]):

> 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.

RSA-100 was factored in April 1991 by the third and fourth author into two 50-digit primes, and RSA-110 was factored in April 1992 by the third author into two 55-digit primes [4]. Here we discuss some of the data that we gathered during our QS-factorization of RSA-120, and we present its two 60-digit prime factors. This factorization is a new general purpose factoring record, breaking the old record by 4 digits. A reader who wished to argue that the 116-digit record has not been broken would need to assert that there is no reason to believe the integrity of the above statement of RSA Data Security Corporation Inc., of its employees, or of the authors of this paper.

There are several points where our effort differs from the old 116-digit record. In the first place we did not employ the approach from [10] where an unknown number of anonymous volunteers on the internet contribute virtually all necessary cycles, all using the same program. Instead, we used *three* independently coded programs that ran at only four different sites, unevenly spread over two continents: the first author used his QS-implementation on workstations at the university of Saarbrücken [3], the other authors used the program from [10; 11] on workstations at Lehigh University, Bellcore, and DEC SRC, and the third author used his SIMD QS-implementation on Bellcore's massively parallel machine (MasPar), as described in [4].[5]

Secondly, we did not impose artificial restrictions on any of the parameters that have to be chosen, as the authors of [11] readily admit to have done (for the reasons mentioned above). As a consequence, the memory demands of our programs, as well as the further storage requirements, vastly exceeded those of any previous factoring efforts of which we are aware. This includes factorizations using the number field sieve applied to composites of a very special form, a detailed account of which can be found in [1; 8; 9]. While we have made every attempt to minimize the total effort to factor RSA-120 we also gathered experience with the much larger programs and files that one would have to deal with when attempting to factor the 512-bit moduli used for security in cryptosystems such as RSA. We note that, as a practical matter, many people would regard 1024-bit moduli as the minimum necessary for longer term security; and that our result provides a benchmark that indicates how far our best current efforts are from being able to factor such 308-digit numbers.

As will be shown in the next section our factorization of RSA-120 took at most 825 MIPS years. As mentioned above, we tried to minimize this runtime, so we do not expect that RSA-120 could have been factored by QS in substantially less time. Extrapolation of the 400 MIPS years for the 116-digit number from [11] to RSA-120, using (1), gives approximately 950 MIPS years; here we use the fact that the multiplier for the 116-digit number was 1, but that we used 7 for RSA-120, i.e., we factored $7 \cdot$RSA-120. (The use of multipliers is one of the reasons why extrapolation of QS-runtimes may be unreliable.) This shows that extrapolation based on (1) can give reasonably accurate results. When we use (1) to extrapolate the 825 MIPS years for RSA-120 to the runtime that would be needed for the QS-factorization of a 129-digit number, we would get approximately 5000 MIPS years. At the time of writing a group of people on the internet is actually working

[5] When we were almost finished we were joined by Walter Lioen and Herman te Riele from the Centre for Mathematics and Computer Science (CWI) in Amsterdam, The Netherlands, who were using their QS-implementation on workstations at the CWI. We gratefully acknowledge their assistance.

on the QS-factorization of a 129-digit number,[6] following the approach and using an adapted version of the code of [10]. Current predictions indicate that it will take between 5000 and 6000 MIPS years to complete this factorization, which is close to our prediction based on extrapolation.

An important issue in the study of the practical behavior of factoring algorithm is the crossover point between QS and NFS. Experiments indicate that the NFS-implementation from [1] would need at least 300 MIPS years for a general 110-digit number. Optimistic extrapolation to RSA-120, using (2), suggests that it would take at least 1300 MIPS years using NFS; similarly, a 129-digit number would take at least 5000 MIPS years. These figures suggest that the crossover point between QS and NFS lies beyond 130 digits.

Unlike QS, however, NFS is a very new algorithm: hardly anybody has much practical experience using it on numbers with more than 110 digits, and most implementations are still in an experimental stage. At the time of writing the third author is working on various improvements of his NFS-implementation that increase the sieving and trial division speed, and that at the same time improve the yield by including relations with triple and quadruple large primes (cf. [8: 7.3]). While it is still too early to present runtime estimates for this new NFS implementation, experiments suggest that the expected NFS runtimes given above can be substantially improved and that the crossover point lies closer to 125 than to 130 digits.

In the remainder of this paper we present some of the data and statistics that we have gathered during our factorization of RSA-120, aimed at a reader with a reasonable background in current factoring terminology. We also present the factorization of RSA-120.

QS-data for RSA-120

We present QS-data related to the following number:

$$\text{RSA-120} = 2270104812954373633342599609474936688958753364660847800$$
$$38173258247009162675779735389791151574049166747880487470296548479,$$

where the first line consists of the first 55 digits, and the second gives the last 65 digits (cf. [14]). We used 7 as multiplier, and a factor base size of 245 810. These choices are based on several experimental runs with these and other choices. For the 116-digit factorization of $n = (10^{142}+1)/(101 \cdot 569 \cdot 7669 \cdot 380623849488714809)$ reported in [11] multiplier 1 and a factor base size of 120 000 were used, but the authors of [11] remark that 160 000 would have been a better choice. Because the factor base size is supposed to grow with the square root of the runtime, this remark would indicate that

[6] The number they are trying to factor is the 129-digit RSA challenge number that was published in Martin Gardner's column in the August 1977 issue of Scientific American.

$$160000 \cdot \sqrt{\frac{L_m[1/2,1]}{L_n[1/2,1]}} \approx 246264$$

(with $m = 7 \cdot$ RSA-120) would be a good choice for RSA-120, and this is indeed close to our choice. The memory requirements of our programs varied between 4 and 7 megabytes. As far as we know this is considerably more than was ever used before in a large factoring project using workstations; the reason that we could afford this 'luxury' is that most new workstations, at least most of the machines that we were using, are normally equipped with 16 or more megabytes of main memory. On such machines our program fitted comfortably alongside space consuming standard software packages (windows, etc.), something which is much harder on machines with only 8 megabytes or less.

Sieving stage. We used the 'double large prime variation' of QS from [11], but we allowed large primes up to 2^{30} which is an order of magnitude larger than 10^8—more or less the customary bound.[7] As a result we collected far more data than would have followed from a straightforward extrapolation of the data from the 116-digit factorization. For that factorization 1.25 million relations involving one or two large primes were sufficient to generate 120000 combinations. For RSA-120 we collected 5105500 relations (1175252338 bytes of data), 48665 of which were full relations, 884323 were partial relations involving a single large prime, and 4172512 were partial relations involving two large primes.[8] The 5056835 partial relations generated 203557 combinations, where only 653899 of the partials actually occurred in at least one combination. This led to a total of $48665 + 203557 = 252222$ relations, which is considerably more than the $\approx 245810 + 10$ relations that we would need to factor RSA-120. The lengths of the relations in terms of the original relations are given in the table.

length	number of combinations	length	number of combinations
1	48665	12	646
2	41493	13	303
3	42958	14	173
4	37018	15	86
5	29910	16	48
6	20753	17	22
7	13821	18	6
8	7934	19	5
9	4587	20	1
10	2484	21	3
11	1305	22	1

[7] Among the 1149564 relations contributed by the first author there were 16203 single large prime relations with large prime between 2^{30} and 2^{31}.

[8] The second author contributed 30.3%, Bellcore's MasPar 26.7%, the first author 22.5%, and workstations at Bellcore, DEC SRC, and the CWI the remaining 20.5%.

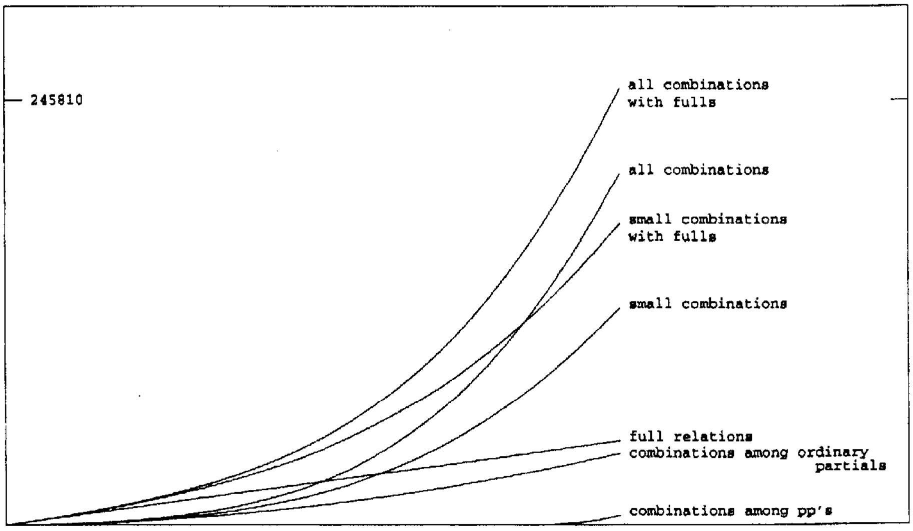

Fig. 1. Progress in the sieving stage

The total number of polynomials generated during the sieving stage was approximately 13 million, and the total number of sieve reports (i.e., actual trial divisions carried out) was approximately 100 million.

The figure presents the progress (as a function of computing time) of the relation collecting stage. The full and the partial relations behaved as usual almost as linear functions of time; the partials are not given in the figure. The upper curve, the sum of the full relations and the independent combinations among all partials, hit the 245810 mark on June 7, after 82 days of sieving. The 'small combinations' gives the growth of the number of combinations among all partials with large prime(s) less than 10^8. From this curve and the curve giving its sum with the fulls, it follows that using large primes larger than 10^8 saved us approximately two weeks of sieving; although considerably more disk space is needed to store the huge amounts of data, it follows that it is indeed a good idea to relax the usual bound of 10^8 on the large primes.

The 'combinations among ordinary partials' gives the growth of the number of combinations in the ordinary single large prime variation. These combinations of ordinary partials are known to accumulate according to a quadratic curve, and we determine the leading term of this quadratic below. The reader might observe that this leading term is sufficiently small that the factorization was completed before the quadratic term becomes visible in the picture. The lowest curve, the 'combination among pp's', is merely a curiosity, as it gives the number of cycles among the partial relations involving two large primes, excluding the partials with a single large prime. These cycles were never actually constructed, because there is no reason at all to exclude the ordinary partials, but we counted their lengths: two cycles of length 3 and 5432 cycles of lengths ranging from 22 to 155, with an average length of 95.5.

Not given in the figure are the curves for the number of combinations among partials with large primes $< i \cdot 10^8$, for $i = 2, 3, \ldots, 10$. For $i = 10$ that curve is almost the same as the 'all combinations curve', because there were 202428 combinations among partials with large primes $< 10^9$, which is only 1129 less than the total of 203557 combinations. For $i = 2, 3, \ldots, 9$ the combination curve for $i \cdot 10^8$ lies approximately halfway between the curves for $(i-1) \cdot 10^8$ and 10^9. All cycles were counted and built using the methods from [11].

Analysis of sieving. The main reason to present all these numbers and curves is to gain a better understanding of the behavior of the double large prime variation, and in particular to be able to predict how much sieving is needed before the factorization can be completed. A theoretical estimate of the number of combinations among the ordinary partial relations is given in [11: Section 3]: if there are t ordinary partial relations, and if p_q is the probability that a large prime $q \in Q$ occurs in such a relation, where Q is the set of large primes that can possibly occur in these relations, then the number of combinations is approximately $c \cdot t^2$, for $c = (\sum_{q \in Q} p_q^2)/2$.

To apply this estimate, let $Q = \{q : q \text{ prime}, 7216241 < q < 2^{30}, (\frac{m}{q}) = 1\}$, where 7216241 is the largest element in our factor base, and $m = 7 \cdot$ RSA-120. For an initial estimate of p_q one might assume that a particular large prime q occurs with probability inversely proportional to q, since $1/q$ of all numbers are multiples of q. This assumption neglects the fact that after the large prime q is removed from the number, the resulting co-factor is (almost) smooth, i.e., factors over the factor base. Since smaller numbers are more likely to be smooth than larger numbers, this may make the occurrence of larger large primes more likely. On the other hand, due to the way the sieving process works, partial relations with smaller large primes are easier to find than those with larger large primes. To take these considerations into account, we take as our model that q occurs with probability proportional to $1/q^\alpha$, for some positive $\alpha < 1$. For the 868120 ordinary partials with large prime q in Q (where $884323 - 868120 = 16203$ had large prime $> 2^{30}$), we found that $\alpha = 0.79$. This leads to $c = 5.832 \cdot 10^{-8}$, and an estimate of $c \cdot 867268^2 = 43866$ combinations. This is reasonably close to the 41490 combinations that we actually got ($41493 - 41490 = 3$ combinations of length two were among the 16203 ordinary partials with large prime $> 2^{30}$ referred to above), and we may conclude that the number of combinations of length two can fairly well be predicted, as soon as α can be estimated reliably enough given some initial collection of ordinary partials. Although other QS-factorizations lead to very similar curves, even two factorizations with the same factor base size and $\#Q$ may have entirely different α's. The curves for one of them may therefore lie much 'higher' or 'lower' than the curves for the other. This is another reason why extrapolation of QS-runtimes may be unreliable.

A theoretical analysis of the expected number of combinations of length more than two is much harder and has to our knowledge not yet been given. Consequently, we do not know of a better way to predict the yield than to plot the initial parts of the curves and extrapolate them in some reasonable way. The third author has all large prime data (138 995 818 bytes) available for anyone who wants to analyse these data further, compute the corresponding α (we only computed the α corresponding to the single large prime relations), or attack this problem in any other way that might take the guess-work out of future QS-runtime predictions.

We used two methods to estimate the runtime of the sieving stage. From a combination of all log-files of all our runs on DEC5000/240 workstations, we found that 5 105 500 relations could have been found on a single DEC5000/240 workstation in 33.02 years. Because a DEC5000/240 is a 25 MIPS machine, it follows that sieving took approximately 825 MIPS years. Another estimate is based on the MasPar runtime. Using only 3/4 of the full memory, the MasPar produced on average 480 full relations per CPU-day. This implies that 101.4 CPU-days would produce 48 665 full relations, and since the MasPar is rated at approximately 3000 MIPS, we derive an estimate of $(3000 \cdot 101.4)/365 \approx 830$ MIPS years. This is too pessimistic because using the full MasPar would produce more fulls per day, and because the partials produced by the MasPar had on average smaller large primes than the partials produced by the other machines, and are therefore more likely to be combined with other partials.

We have carried out extensive tests with other choices of the factor base size, both before we started and after we had finished the sieving step. According to our computations a factor base size of 240 000 would have required one percent more time, 230 000 four percent, and 220 000 more than seven percent. We did not attempt to analyse the effects of a factor base that is even larger than 245 810.

Matrix reduction. As a result of the sieving stage we got a 252222×245810 bit-matrix. To find dependencies modulo two among its rows the third author used the technique described in [9] with the extension from [1]: structured Gaussian elimination (cf. [5; 13]) followed by the incremental version from [1] of the MasPar dense matrix eliminator described in [6]. Structured Gauss managed to reduce the size of the matrix to 89304×89088. This took 15 hours on a Sparc10 workstation, 10 of which were needed to build the 994 489 344 byte dense matrix, in 47 separate files of more than 21MB each. This dense matrix was smaller than expected (and smaller than one of the dense matrices from [1]) and just fitted in core on the MasPar. Reading it into core took 13 minutes, finding dependencies took 4 CPU-hours. The second dependency produced the two 60-digit prime factors of RSA-120:

$$327414555693498015751146303749141488063642403240171463406883$$

and

$$693342667110830181197325401899700641361965863127336680673013.$$

Acknowledgments. Acknowledgments are due to Paul Leyland for his helpful comments.

References

1. Bernstein, D. J., Lenstra, A. K.: A general number field sieve implementation. 103–126 in [7]
2. Buhler, J. P., Lenstra, Jr., H. W., Pomerance, C.: Factoring integers with the number field sieve. 50–94 in [7]
3. Buchmann, J. A., Denny, T. F.: An implementation of the multiple polynomial quadratic sieve, University of Saarbrücken (to appear)
4. Dixon, B., Lenstra, A. K.: Factoring integers using SIMD sieves. Advances in Cryptology, Eurocrypt '93, Lecture Notes in Comput. Sci. (to appear)
5. LaMacchia, B. A., Odlyzko, A. M.: Computation of discrete logarithms in prime fields. Designs, Codes and Cryptography 1 (1991) 47–62
6. Lenstra, A. K.: Massively parallel computing and factoring. Proceedings Latin'92, Lecture Notes in Comput. Sci. **583** (1992) 344–355
7. Lenstra, A. K., Lenstra, Jr. H. W. (eds): The development of the number field sieve. Lecture Notes in Math. **1554**, Springer-Verlag, Berlin, 1993
8. Lenstra, A. K., Lenstra, Jr., H. W., Manasse, M. S., Pollard, J. M. : The number field sieve. 11–42 in [7]
9. Lenstra, A. K., Lenstra, Jr., H. W., Manasse, M. S., Pollard, J. M.: The factorization of the ninth Fermat number. Math. Comp. **61** (1993) 319–349
10. Lenstra, A. K., Manasse, M. S.: Factoring by electronic mail. Advances in Cryptology, Eurocrypt '89, Lecture Notes in Comput. Sci. **434** (1990) 355–371
11. Lenstra, A. K., Manasse, M. S.: Factoring with two large primes. Math. Comp. (to appear); extended abstract in: Advances in Cryptology, Eurocrypt '90, Lecture Notes in Comput. Sci. **473** (1991) 72–82
12. Pomerance, C.: The quadratic sieve factoring algorithm. Lecture Notes in Comput. Sci. **209** (1985) 169–182
13. Pomerance, C., Smith, J. W.: Reduction of huge, sparse matrices over finite fields via created catastrophes. Experiment. Math. **1** (1992) 89–94
14. RSA Data Security Corporation Inc., sci.crypt, May 18, 1991; information available by sending electronic mail to challenge-rsa-list@rsa.com