æ

THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE

Extended Abstract

Derek Atkins[1], Michael Graff[2], Arjen K. Lenstra[3], Paul C. Leyland[4]

[1] 12 Rindge Avenue, Cambridge, MA 02140, U. S. A.
E-mail: warlord@mit.edu
[2] Iowa State University, 215 Durham Center, Ames, IA 50010-2120, U. S. A.
E-mail: explorer@iastate.edu
[3] MRE-2Q334, Bellcore, 445 South Street, Morristown, NJ 07960, U. S. A.
E-mail: lenstra@bellcore.com
[4] Oxford University Computing Services, 13 Banbury Road, Oxford, OX2 6NN, U. K.
E-mail: pcl@ox.ac.uk

Abstract. We describe the computation which resulted in the title of this paper. Furthermore, we give an analysis of the data collected during this computation. From these data, we derive the important observation that in the final stages, the progress of the double large prime variation of the quadratic sieve integer factoring algorithm can more effectively be approximated by a quartic function of the time spent, than by the more familiar quadratic function. We also present, as an update to [15], some of our experiences with the management of a large computation distributed over the Internet. Based on this experience, we give some realistic estimates of the current readily available computational power of the Internet. We conclude that commonly-used 512-bit RSA moduli are vulnerable to any organization prepared to spend a few million dollars and to wait a few months.

1 Introduction

One of the earliest descriptions of the RSA public key cryptosystem appeared in Martin Gardner's column 'Mathematical Games' in the August 1977 issue of Scientific American. There, the inventors of RSA presented the following challenge. Let

$$\begin{aligned} r = {} & 1\ 1438\ 1625\ 7578\ 8886\ 7669\ 2357\ 7997\ 6146\ 6120\ 1021\ 8296\ 7212\ 4236\ 2562\ 5618\ 4293 \\ & 5706\ 9352\ 4573\ 3897\ 8305\ 9712\ 3563\ 9587\ 0505\ 8989\ 0751\ 4759\ 9290\ 0268\ 7954\ 3541 \end{aligned}$$

be the public modulus and let $e = 9007$ be the public exponent; find the plaintext English message which corresponds to the encrypted message

$$\begin{aligned} & 9686\ 9613\ 7546\ 2206\ 1477\ 1409\ 2225\ 4355\ 8829\ 0575\ 9991\ 1245\ 7431\ 9874\ 6951\ 2093 \\ & 0816\ 2982\ 2514\ 5708\ 3569\ 3147\ 6622\ 8839\ 8962\ 8013\ 3919\ 9055\ 1829\ 9451\ 5781\ 5154, \end{aligned}$$

where the original plaintext English message had been transformed into a decimal number using the transformation $\mathbf{A} = 01$, $\mathbf{B} = 02$, ..., $\mathbf{Z} = 26$, with 00 indicating a space between words. It is readily verified that application of this transformation to the title of this paper, followed by an eth powering modulo r, results in the encrypted message. Here we describe how we reversed this process and found the title of this paper, thereby solving the 'RSA-challenge' and winning the US$100 prize. The prize has been donated to the Free Software Foundation.

The modulus r has 129 decimal digits and is, according to [6], the product of a 64-digit prime p and a 65-digit prime q such that both $p-1$ and $q-1$ are relatively prime to e. The primes p and q were kept secret. It is well known that the encrypted message can be decrypted by computing (encrypted message)^d mod r where d satisfies $d \cdot e \equiv 1 \bmod (p-1)(q-1)$. Because of the condition on p and q, finding d is straightforward if $(p-1)(q-1)$ is known. But knowing $(p-1)(q-1)$ is equivalent to knowing the secret primes p and q (cf. [21]).[5] It follows that the encrypted message can be decrypted by factoring r.

In the full paper, we will describe in detail how we managed to factor r. This extended abstract consists of a discussion of some historical remarks on the supposed difficulty of factoring r (Section 2), an outline of the method that we used to factor r (Section 3), an overview of the analysis that will appear in the full paper (Section 4), a view behind the scenes of this world-wide computing effort (Section 5), and some concluding remarks (Section 6).

2 Predicting the difficulty of factoring r

Back in 1976, Richard Guy wrote 'I shall be surprised if anyone regularly factors numbers of size 10^{80} without special form during the present century' [8]. In 1977, Rivest estimated in [20] that factoring a 125-digit number which is the product of two 63-digit prime numbers would *require* at least 40 quadrillion years using the best factoring algorithm known, assuming that $a * b \,(\mathrm{mod}\, c)$ could be computed in 1 nanosecond, for 125-digit numbers a, b, and c.[6] Thus, it is not surprising that the inventors of RSA felt confident that 'with such a huge modulus the message will never be recovered' [23] and offered a $100 prize to the first successful decoder of the encrypted message. Interestingly, until the message was decoded, none of the parties involved remembered the expiration date of April 1, 1982, given in [20] but not in [6].

With 10^9 modular multiplications per second, about 1.3×10^{33} can be carried out in 40 quadrillion years. It is not clear why, back in 1977, it was believed that 1.3×10^{33} modular multiplications were required to factor a 125 or 126-digit number. Unfortunately, Rivest does not recall how he obtained his estimate. At that time, Pollard's 'rho method' was probably the fastest available factoring

---

[5] This does *not* imply that in order to decrypt the encrypted message it is necessary to find p and q but no faster way to decrypt an RSA-encryption has yet been published.

[6] A similar estimate can be found in [6].

method for which a run-time analysis had been published. To find a prime factor p it needs, on average, about $2\sqrt{p}$ iterations, where each iteration costs two modular multiplications. For a 63-digit prime factor this amounts to at most 1.3×10^{32} modular multiplications. This suggests that Rivest might have used the rho method but miscounted the number of zeros. In that case, the correct estimate should have been 4 quadrillion years, which still sounds sufficiently impressive to convince people of the infeasibility of breaking 125-digit RSA keys.

Since 1977 many new factoring algorithms have been invented and the availability and speed of computing resources has changed dramatically. Rivest's imaginary '1 nanosecond per modular multiplication' machine, however, still stretches the imagination. With pipelining and special purpose hardware with gate delays of about 10 picoseconds, it might just be feasible, even for 129-digit numbers. If we were to build such a machine, then we estimate that it has a fair chance to find the 64-digit factor of r after 5×10^{15} modular multiplications using the *elliptic curve method* (ECM). This would require two months: 240 quadrillion times faster than Rivest's estimate — progress that is due solely to improved factoring methods.

Today, finding 64-digit prime factors using ECM is believed to be infeasible. On a Sparc 10 workstation the 5×10^{15} modular multiplications on 129-digit numbers that would probably be needed for ECM would take well in excess of 15000 years. ECM uses mostly computations on local data which can be kept in the cache, and hardly any access to main memory. Consequently, the method runs x times faster on a machine whose 'mips rating' is x times better. Therefore, a statement of the kind 'finding a 64-digit prime factor of a 129-digit number using ECM would probably take more than half a million mips years' might make some sense. Here, we rate a Sparc 10 workstation at 35 mips; notice that this would imply a mips rating of about 3 million for Rivest's imaginary machine if it used ordinary 32-bit arithmetic.

For the quadratic sieve factoring algorithm (QS), the method that we used to factor r, run-times expressed in mips years are much harder to justify. Sieving algorithms spend most of their time accessing more or less random locations in memory. On most workstations, processor speed is poorly correlated with memory speed and, therefore, the mips rating of the processor is not a reliable measure of the expected performance of a sieving algorithm. To give some examples: on a Sparc 10 workstation QS would have taken about 120 years to factor r, which leads to an estimate of about 4000 mips years. On a Dec 5000/240, rated at 25 mips, we would have spent 237 years, which translates into almost 6000 mips years. On a Sun 3/50, however, rated at 1.5 mips, we would have spent about 1100 years but only 1700 mips years. Averaging over the machines that were used, and keeping in mind that we are indulging in energetic handwaving, we give a very approximate estimate of 4000 to 6000 mips years for the time spent on the factorization of r. If the ratio between memory and processor speed of Rivest's machine is not much worse than on current workstations, it could factor r in less than a day. Even though expressing run-times in mips years is far from ideal for memory-bound processes, it is common practice in factoring literature. It provides a convenient way to compare and predict run-times of factoring problems, and to check the accuracy of theoretical predictions.

Heuristically it can be argued that the expected run-time of QS behaves as

$$L(n) = \exp((1+o(1))\sqrt{\log n \log\log n})$$

for $n \to \infty$, where n is the number to be factored. Often, one simply ignores the $o(1)$, and interprets the resulting $L(n)$ as the number of operations needed to factor n. Since the $o(1)$ is neither 0 nor constant, this practice hardly makes sense.

This naive approach was first used in [21: Section IX.A], where $L(n)$, without the $o(1)$, was given as the run-time of Schroeppel's linear sieve factoring algorithm (cf. [22]). This is a big improvement compared with Rivest's slightly earlier estimate in [6]. The run-time from [21] did not include the matrix elimination step, because this step was considered to be trivial compared to the sieving step (cf. [22]). Including it would have increased the run-time to

$$\exp((u/2+o(1))\sqrt{\log n \log\log n})$$

(cf. [18: Theorem 6.1]), where a linear dependency in an $(s+1)\times s$ bit-matrix can be found in time s^u. At the time of the publication of [21], the best one could do was $u \approx 2.795$. Although for small n the matrix step is in practice negligible compared to the sieving step, this is not the case for the larger n in [21: Table 1], thus making these values strictly speaking (and disregarding the $o(1)$-issue) incorrect, for 1978. It was later shown that the sparseness of the matrix can be exploited and that $u = 2 + o(1)$ can indeed be used.

This does not imply that there are any significant digits in [21: Table 1]. The table illustrates, however, the approximate growth rate of $L(n)$, which *can* be used in an attempt to predict run-times: if factoring n_1 takes time t for some QS-implementation and $|\log(n_2/n_1)|$ is reasonably small, then factoring n_2 using the same implementation can be expected to take time approximately $t \cdot L(n_2)/L(n_1)$, where both $o(1)$'s are simply omitted.

Using this method it was estimated in [2] that factoring a 120-digit number using QS would take about 950 mips years, based on the observation from [16] that factoring a 116-digit number took about 400 mips years. The actual 120-digit factorization took about 825 mips years, slightly less than the prediction, due to several improvements in the program and intentional suboptimality of the parameter choices for the 116-digit factorization. Similar extrapolation of the 120-digit result to r would lead to about 4200 mips years, which is on the low side of the 4000 to 6000 mips years that we spent. This may be due to a combination of two factors. In the first place, the effect of the unknown behavior of the $o(1)$ in the 9 digit jump from 120 to 129 digits is significantly bigger than in the jump from 116 to 120 digits. Apparently, neglecting the $o(1)$ leads to an overly optimistic estimate. Secondly, the effects of small (and relatively slow) memories should be even worse for r than for the 120-digit factorization from [2].

The reason why these deliberations are considered to be useful is that they might help us to assess the security of widely-used 512-bit RSA moduli. Since $L(2^{512})/L(10^{129}) \approx 92$ (without the two $o(1)$'s), we find that factoring a 512-bit RSA modulus using our current QS-implementation can probably not be done in less than 500 000 mips years. An upper bound is harder to give, but a million mips years is probably the right order of magnitude.

We should stress that this estimate might not be a good indication of the difficulty to break a 512-bit RSA modulus. It tells us how hard it would probably be for a fairly well understood, reliable and by now rather ancient algorithm. The newer *number field sieve* factoring algorithm (NFS) (cf. [13]) is asymptotically superior to QS. Recent experiments have suggested that the crossover point between QS and NFS lies well below 116 digits (cf. [5; 7]), but it remains to be seen how well the memory requirements of NFS for 512-bit numbers can be handled. In any case, we suspect that with the factorization of r we have seen the last big QS-factorization. Furthermore, given how hard it was, 17 years ago, to make a decent prediction about the difficulty of factoring r, it is probably better to avoid making any firm prediction about 512-bit numbers.

3 Factoring r

We used the double large prime multiple polynomial variation of the quadratic sieve factoring method (QS) to factor r. The goal of QS is to find pairs of integers x, y such that $x^2 \equiv y^2 \bmod r$. Because r divides $x^2 - y^2 = (x-y)(x+y)$, we find that $r = \gcd(r, x-y)\gcd(r, x+y)$. If the pairs are generated in a random fashion, then there is a probability of $1/2$ that this factorization is non-trivial (because r has only two distinct prime factors) [3]. In QS, pairs x, y are constructed in two steps, the *sieving step* and the *matrix step* [12: 4.16; 16; 18; 24]. We give an outline of these steps as applied to r. In the full paper we will give more details.

3.1 The sieving step

First, we select a *multiplier*, a small positive integer m such that mr is a quadratic residue modulo many small primes (cf. [9: 4.5.4]). Next, we compute the *factor base* P, consisting of -1 and the primes $p \le B_1$ for which mr is a quadratic residue modulo p, for some bound B_1. In the sieving step we collect a set of *relations*: integer tuples $(v, q_1, q_2, (e_p)_{p\in P})$ such that

$$v^2 \equiv q_1 \cdot q_2 \cdot \prod_{p\in P} p^{e_p} \bmod r, \tag{3.2}$$

where each q_i is either 1 or a *large prime* $\in (B_1, B_2]$, for some *large prime bound* B_2. If $q_1 = q_2 = 1$ the relation is called *full*, if only one of the q_i equals 1 it is a *partial*, and a *double partial* otherwise. If the large primes match, the (double) partial relations can be combined into *cycles* [16]. The sieving step is complete as soon as #fulls+#cycles $>$ $\#P$.

We used $m = 5$ as multiplier. Asymptotically, the factor base size in QS applied to n behaves as $\sqrt{L(n)}$, for $n \to \infty$. A factor base size of 245 810 elements was close to optimal for the 120-digit number RSA-120 factored in [2], with multiplier 7. Therefore a factor base of $245\,810\sqrt{L(mr)/L(7 \cdot \text{RSA-120})} \approx 554\,933$ (omitting both $o(1)$'s) elements should be reasonable but on the small side (because the $o(1)$ grows with n). After experiments with other values, we settled for $\#P = 524\,339 = 51 + 2^{19}$ (and $B_1 = 16\,333\,609$). Our choice is certainly within 20% of optimal, and probably slightly suboptimal. We felt that this was less undesirable than the memory-related problems caused by an even larger choice.[7] We used $B_2 = 2^{30}$.

Based on our experience with many other factorizations, we expected that we would need about 110 000 fulls before we had enough cycles to complete the sieving step. Experiments indicated that this would take less than 6000 mips years. More than 8 million relations would have to be collected (since only 1 out of every ≈ 75 is full), using a program which would need approximately 10.5 MBytes to run. We opted for the idle-cycle collection method from [15]. This sounds easier than it was, even though we could re-use the software, because our effort would need an order of magnitude more time, memory, and disk space to run the sieving program and to store the relations. We give more details of the resource management in Section 5. We estimate that we had approximately 600 contributors using more than 1600 machines and producing about 80% of the relations. The other 20% was contributed by several MasPars running the program from [4].

On March 21 1994 we had about 8.25 million relations, with more than 108 000 fulls and 417 000 cycles. Because $108\,000 + 417\,000 > 524\,339 = \#P$, the 'cease and desist' message was mailed out on March 22. The final counts, on March 26, were: 112 011 fulls, 1 431 337 partials, and 6 881 138 double partials, for a total of 8 424 486 relations after less than 220 days of sieving. The (double) partials that could occur in cycles were written to tape by the first author and sent to the third (by regular overnight mail). They resulted in 457 455 cycles, constructed using the method from [16]. With the fulls, which were `ftp`-ed from `hot-spare.mit.edu` to `flash.bellcore.com` via `tbird.cc.iastate.edu`, this led to 569 466 sparse 524 339-dimensional vectors $(e_p)_{p \in P}$ as in (3.2).

3.3 The matrix step

Here we find linear dependencies modulo 2 among the $> \#P$ bit-vectors $(e_p \bmod 2)_{p \in P}$. Each dependency corresponds to a set $W = \{(w, (e_p)_{p \in P}) : w^2 \equiv \prod_{p \in P} p^{e_p} \bmod r\}$ for which $\sum_W (e_p)_{p \in P} = (2w_p)_{p \in P}$ for integers w_p. Consequently, $x \equiv \prod_W w \bmod r$ and $y \equiv \prod_{p \in P} p^{w_p} \bmod r$ satisfy $x^2 \equiv y^2 \bmod r$.

[7] Because $\#P \equiv 51 \bmod 2^{14}$ we could use the QS-implementation from [4] on a 16K MasPar massively parallel computer. The choice is on the high side for this implementation, which performed noticeably worse than expected based on the experience from [2].

To find dependencies among the rows of the $569\,466 \times 524\,339$ bit-matrix with, on average, ≈ 47 bits per row, we intended to use the same approach as in previous large scale factorizations. The largest matrix processed so far, however, had less than half the number of rows and columns ($252\,222 \times 245\,810$ with the same density, cf. [2]), making the present effort an order of magnitude more difficult. Using structured Gaussian elimination (cf. [10; 14; 19]) the matrix was reduced to a dense $188\,614 \times 188\,160$ bit-matrix. This took less than 12 CPU-hours on a Sparc 10 workstation, 97% of which was spent building the 4 436 201 280 byte dense matrix, in 268 separate files of about 16 MBytes each.

To find a dependency among the rows of the dense matrix, we used the incremental version from [1] of the MasPar dense matrix eliminator from [11]. The dense matrix was processed in 5 blocks. With a core size of 1GByte, 41 595 rows could be processed per block. Each new block was first eliminated with the pivots found in the previous blocks, then with the new pivots in the block itself, after which the result was written to disk. Each of the first four blocks resulted in a file of ≈ 980 MBytes. Notice that one wrong bit, either in the dense matrix or in the intermediate files, can render the entire computation worthless. For this reason, the dense matrix intentionally contained 268 spurious dependencies which were supposed to be found at regular intervals during the elimination process to signal possible errors.

The entire process took 45 hours on a 16K MasPar MP-1: 75 minutes to read the dense matrix, 100 minutes to write the first four blocks, and 42 hours for the eliminations. This is better than we had expected,[8] because, unlike previous eliminations, most pivot rows had to be read from disk before eliminating with them and they could not be stored in registers during the elimination. All 268 spurious dependencies were found at the right moment during the computation, and 205 'true' dependencies were found at the end. It took a few minutes to convert the 205 dependencies modulo 2 in the dense matrix to dependencies in the sparse one. Computing the pairs x, y with $x^2 \equiv y^2 \bmod r$ took a few minutes per dependency. The first three dependencies led to $r = 1 \cdot r$. At 18:15 UT on April 2 1994 the fourth one led to the factorization $r = p \cdot q$, with p the 64-digit prime

3490 5295 1084 7650 9491 4784 9619 9038 9813 3417 7646 3849 3387 8439 9082 0577

and q the 65-digit prime

3 2769 1329 9326 6709 5499 6198 8190 8344 6141 3177 6429 6799 2942 5397 9828 8533.

Five of the first ten dependencies were 'unlucky'. It might be interesting to note that $p - 1 = 2^5 \cdot 3^2 \cdot \bar{p}$ for a 62-digit prime $\bar{p} = 1 + 2 \cdot 7^2 \cdot \hat{p}$ and a 60-digit prime $\hat{p}$; and that $q - 1 = 2^2 \cdot 41 \cdot \bar{q}$ for a 63-digit prime $\bar{q} = 1 + 2^2 \cdot 53 \cdot \hat{q}$ and another 60-digit prime $\hat{q}$. We suspect that $p - 1$ and $q - 1$ were consciously chosen to have large prime factors, so that r would withstand a Pollard $p - 1$ factoring

[8] The 4 hours for the 245 810 column matrix from [2] extrapolate to $4 \cdot (524\,339/245\,810)^3 \approx 39$ hours.

attack. The large prime factors of $\bar{p}-1$ and $\bar{q}-1$ might have been based on the widespread belief that they would be necessary to prevent a decryption attempt using iterated encryption [17]. The resistance of r against a Pollard $p+1$ attack, which had not yet been published by 1977, was probably a coincidence: $p+1 = 2 \cdot 1\,376\,164\,939\,307\,949\,996\,650\,933 \cdot \mathrm{p}40$, and $q+1 = 2 \cdot 3^4 \cdot 11 \cdot 79 \cdot 197 \cdot 227 \cdot \mathrm{p}55$, with $\mathrm{p}j$ denoting a j-digit prime.

4 Analysis of the data collected

Let $Q = \{q : q \text{ prime}, B_1 < q \le B_2, \left(\frac{mr}{q}\right) = 1\}$ be the set of 26 679 473 large primes which can possibly occur in the (double) partial relations. Let $p(q)$ be the probability that $q \in Q$ occurs in a partial relation. In [16] it is shown that t partial relations can be expected to generate $c \cdot t^2$ cycles, where $c = (\sum_{q \in Q} p(q)^2)/2$. To be able to predict the number of cycles among partials it is therefore useful to get more insight in the behavior of $p(q)$. It has been suggested in [16] that $p(q)$ is proportional to $1/q^\alpha$, for some positive $\alpha < 1$. In the full paper we will analyse which α best fits our data, and discuss how well the theoretically estimated number of cycles among the partials agrees with our findings. Furthermore, we will present the α_1 and α_2 corresponding to $p_1(q)$ and $p_2(q)$, where $p_1(q)$ is the probability that $q \in Q$ occurs as smallest large prime in a double partial and $p_2(q)$ is the probability that it appears as largest large prime in a double partial relation.

Different factorizations with similar α's usually have similar cycle-yields, which makes the α's good indicators of the cycle-yield. Because the behavior of the α's can be derived long before the factorization is complete, this might be helpful to predict the cycle-yield in other factorizations. The behavior of the cycle-yield is of particular interest for future NFS factorizations where relations can have more than 2 large primes (cf. [7]). Understanding their cycle-yield is crucial to be able to predict the run-times (cf. [5]).

The estimates that were sent to the contributors (cf. Section 5) were based on extrapolations of the cycle-curve. Initially a quadratic curve gave a good fit, and was therefore used to extrapolate. We lost our confidence in quadratic extrapolation in late December 1993, when the extrapolation predicted that we would need more than 13 million relations. Fortunately, we noticed in January that a quadratic no longer gave a close fit, and that the cycle-curve was showing stronger than quadratic growth. Experiments indicated, quite unexpectedly, that a quartic function gave a very good approximation. However the residuals, although small, were quite well serially correlated, which probably implies that the cycle-curve does *not* behave as a quartic. Since the error terms were relatively small, and because the first few extrapolations turned out to be unusually accurate, we kept using the quartic anyway. We predicted in early February that 8.15 million relations would suffice. The quartic extrapolation proved to be very slightly optimistic — it turned out that we actually needed 8.2 million relations. In the full paper we will provide more details and graphs of the actual yields and of the quadratic and quartic extrapolations.

5 Resource management

To factor r we assembled the largest collaboration yet seen in computational number theory and, possibly, performed the largest single computation ever completed. In several important respects, the resources we had available were barely adequate for the task. Consequently, ingenuity and diplomacy were required for the successful completion of the project. Full details of the resource management aspects of the computation will be given in the full paper; here we give a summary of the salient points.

Based on experience with earlier factorizations, especially those in [15] and [2], we could make reasonably precise estimates of the disk space, amount of computation and memory usage which would be required. Before we started, we estimated that we would need about 8 million relations, each of which would take about 350 bytes to store. We knew that a few mips-millenia would be required. We also estimated that the factor base would need to have about 400 000 to 600 000 elements, implying that the sieving programs would take over 8 MBytes of active memory. We did not know, before we started, how many machines and of what power would be available to us but we did have some encouraging offers of support. It was clear, however, that an unusually large number of people and machines would be involved with the project.

The first action was to settle the size of the factor base. As explained in 3.1, a figure of about half a million primes was close to optimal. A somewhat larger number would have reduced the theoretical amount of computation required, but we were already using 10.5 MBytes of active virtual memory. Given that most of the machines on the Internet have less than 16 MBytes of physical memory, and many have 8 or less, even 10.5 MBytes is uncomfortably large and would seriously restrict the number of people capable of helping us.

Before going public, the first and fourth authors ported the software to as many different kinds of workstations and PCs as were available to us. The first author persuaded MIT to lend us a file server for the duration. This machine was a DEC 5000/240 with 32 MBytes RAM and two (later three) 975 MBytes disks. MIT systems staff also agreed to perform regular backups of the information on disk. For these reasons MIT was chosen as collection site.

Also before the computations started, the second and fourth authors wrote documents explaining what we were proposing to do, how we were going to do it, and how to get further information should the reader wish to join in. The second author set up several mailing lists and email aliases for communications to and from our workers and the four coordinators; he also took on the role of front-line contact person, dealing with virtually all the 600 contributors and many others who expressed an interest but did not join in for whatever reason.

The postings to potential contributors went out on August 19 1993 to the 'number theory net', on August 23 to the 'cypherpunks' and the 'PGP development group', and on August 24 to the following newsgroups: alt.hackers, alt.security, alt.security.pgp, alt.security.ripem, comp.arch, comp.security.misc, sci.crypt, and sci.math. The initial response to our postings and the number of incoming relations was sufficiently encouraging to conclude that the project would be feasible.

Once contributors started offering their services, we had to port our software to their platforms, if we had not already done so. Although one Unix box is very much like another, there are sufficient differences that we often had to ask our contributors to perform the port for us and to send back the modifications so that we could pass them on to others with the same platform. Our code was run successfully on machines as disparate as 16MHz 80386sx PCs and Cray C90s. An attempted port to a Thinking Machines CM-5 failed, but one US corporation managed to get the sieving code running on a couple of fax machines!

The relatively severe memory requirements and the commonness of 8 MBytes machines gave us a strong incentive to produce memory-frugal variants of the siever. By reducing the sieve batch size and by reading the roots of mr mod the p's from a file, we were able to reduce the space required from the default 10.5 MBytes to 6.5 MBytes, albeit at a cost of running over twice as slowly. However, half a computer is better than none and by allowing more people to take part, we shortened the overall time taken for the sieving phase. Even so, 6.5 MBytes does not leave much room for an operating system and most of our contributors with 8 MBytes machines were only able to run our code overnight.

Once we were up to speed we received on average about 40 thousand relations per day, with peaks of over 50 thousand during the holiday seasons. Daily, all newly received relations that satisfied (3.2) were sorted and merged with older data; faulty or duplicate relations were thrown out. This took at most a few hours per day, even when the accumulated data had reached a couple of GBytes. About one relation in ten thousand received was unacceptable in some way. Of these, many had suffered in transit through mail gateways but we also received a number of relations intended for another factoring-by-email computation being run at the same time — these relations were sent on to the correct destination, together with a polite note to suggest that their contributors be informed of their error.

About half-way through the computation, it became clear that we would run out of disk space if we continued to store the relations in plain text. Accordingly, the fourth author wrote a simple but efficient compressor, and the bulk of the data was kept in a more compact format. Each day's data was left as ASCII for convenience (fewer processing programs had to be re-written) and a combined uncompress-sort-merge-compress of the total data set performed every few weeks as the disk began to get uncomfortably full. Unfortunately, we did not notice the disk overflowing in early January 1994. About three weeks' data had been corrupted; fortunately, the outputs from the daily runs of the email processor were still on disk so we were able to recover relatively easily.

To monitor the progress, the fourth author adapted the cycle-counting software from [16] to the relatively small workstation we had at MIT. Towards the end of the project this became a challenging task, with several GBytes of compressed data; a graph containing several million nodes, edges and components and a few hundred thousand cycles; no disk space to decompress or to write intermediate files; and only a 25 mips processor with 32 MBytes of physical memory. Details will be given in the full paper.

We have already described in Section 4 how the quadratic extrapolation began to indicate that we would need many more relations than we had predicted at the start of the project. We became seriously concerned at this point that we would have difficulty in maintaining the interest of our contributors for several months longer than anticipated. Further, although we had enough disk space (just!) to process 9 million relations, an extra 50% would have caused us significant problems. Luckily, our discovery of the much more accurate quartic prediction was made before we told our contributors, saving us from worrying them with a false alarm.

The resulting counts were mailed to the contributors each month, along with an estimate of how much more would be needed. These reports were also posted to Usenet, as it was found that they were very effective at attracting new contributors. The estimates were based on extrapolation of the cycle-curve (cf. Section 4).

Some substantial effort had to be devoted to problem solving. As our program had not previously run on certain platforms, its portability had not been *proven*. Two examples will suffice. One unfortunate contributor discovered that his disk had filled with over 300 MBytes of error messages overnight after our program had run wild. We discovered rather bizarre behavior of a particular compiler for the Intel 80x86 — unless the value of a certain variable was printed out, the program would crash unpredictably.

6 Conclusion

Our Internet contributors were exclusively volunteers, donating spare computer time from overnight and weekend running. Although 600 people and 1600 machines constitute the largest *ad hoc* multiprocessor so far assembled, it is only a tiny fraction of the size and power of the entire Internet. It is difficult to get even a rough estimate of the potential power available. At the time of writing (October 1994) it is thought that there are between 3 and 4 million machines on the Internet; it is currently growing at about 10-20% per month. If we assume, conservatively, that there are 3 million machines, each of which can average 5 mips (i.e., a typical 10 mips workstation with 50% availability) the potential power available is around 1.5×10^7 mips — about five times the power of Rivest's hypothetical machine. In principle, the Internet could have factored r in three hours! The implementation details are left as an exercise for the reader.

Let us attempt to give a more plausible estimate. We believe that we could acquire 100 thousand machines without superhuman or unethical efforts. That is, we would *not* set free an Internet worm or virus to find resources for us. Many organizations have several thousand machines each on the net. Making use of their facilities would require skilful diplomacy, but should not be impossible. Assuming the 5 mips average power, and a one year elapsed time, it is not too unreasonable to embark on a project which would require half a million mips years. We conclude that 512-bit RSA keys are on the brink of vulnerability to a venerable but robust algorithm. However, we believe that solving the RSA-challenge is probably the last gasp of an elderly workhorse. A credible attack on a 512-bit key would almost certainly use an NFS implementation.

However, even using NFS to factor a 512-bit number would be a non-trivial undertaking. Without falling into the trap of making over-precise estimates of run-times, we can say that the resources required will be well in excess of what we used to factor a 426-bit key. If the open Internet were to be used, the resource management would be tricky. If an organization were to purchase the hardware necessary to give an average power of 500 000 mips, — 5000 processors, each rated at 100 mips — an outlay of several million dollars would be needed. Assuming that NFS can be implemented on those machines and assuming that the matrix step can be performed, 512-bit keys would take a few months to factor. Nonetheless, organizations exist with annual budgets in excess of ten million dollars and which might regard it to be cost-effective to break 512-bit RSA keys protecting information of particular interest.

Acknowledgments. In the first place we want to thank everyone who donated their time to this project. Here is a list of all contributors to the sieving step, in the following format: a number indicating the percentage of the contribution, followed by one or more names of the persons or institutions responsible for that contribution. Some contributors confirmed that they wanted to remain anonymous ('Confirmed Anonymous'), some contributors did not respond when we asked their permission to include their name in this paper ('Unclaimed'), and for some we only have an email address. We apologize for any omissions, or spelling errors.

8.348 Arjen K. Lenstra, 5.022 Magnus Y. Alvestad, 4.412 Brian A. LaMacchia, 4.126 Mats Lofkvist, 3.855 Tage Stabell-Kulo, 3.704 Clem Taylor, 3.579 Case Larsen, 3.295 Electronic Data Systems, Research & Development, 2.490 Harald Hanche-Olsen, 2.459 Germano Caronni, 1.988 Daniel R. Oelke, 1.883 Derek Atkins, 1.445 Edesio Costa e Silva, 1.250 Jonathan Bertoni, 1.219 Mark Shand, 1.161 Robin Humble, 1.148 David DiGiacomo, 1.115 Allan Sherman, 0.976 Robert J. Harley, 0.974 Brick Verser, 0.970 Lucien Van Elsen, 0.934 Andrew Loewenstern, 0.927 Paul Leyland, 0.819 Scott Logan, 0.810 Pekka Jarvelinen, 0.748 Michael Graff, 0.733 Unclaimed, 0.717 Shaun Case, Jim Garlick, 0.680 Teun Nijssen, 0.666 Walter Lioen, Herman te Riele, Dik Winter, 0.666 J.P.M. de Vreught, 0.616 Mark S. Manasse, 0.594 Unclaimed, 0.561 Steven Leikeim, 0.558 Bruce K. Martin Jr., 0.556 John ffitch, 0.538 Jack Repenning, 0.521 Jeffrey Blohm, 0.515 Jim Castleberry, 0.514 John Kohl, 0.497 Billy Barron, 0.472 Andrew Odlyzko, 0.471 Boris Hemkemeier, 0.468 Bruce Rossiter, 0.458 Nur Serinken, 0.445 Peter Conrad, 0.443 Ken Rose, 0.442 Christer Berg, 0.429 Bob Devine, 0.428 Dana Jacobsen, 0.423 olson@lxt.com, 0.414 Lee Nave, 0.409 Unclaimed, 0.370 Paul Hodgkinson, 0.367 Unclaimed, 0.365 Sylvestre Matuschka, 0.354 Bill Rea, 0.353 Confirmed Anonymous, 0.349 David Crowley, 0.347 Hugh D. Gamble, 0.333 Bruce Dodson, 0.333 Herb Savage, 0.326 Andrew Mossberg, 0.321 Eric Postpischil, 0.310 Unclaimed, 0.304 Daniel L. Carroll, 0.296 Unclaimed, 0.295 Ian T. M. Flanigan, 0.284 Confirmed Anonymous, 0.281 Mitchell Perilstein, 0.270 Confirmed Anonymous, 0.268 Eric Prestemon, 0.257 Michael J. Griffin, 0.257 Dan Nachbar, Applied Computing Systems Institute of Massachusetts, Center for Intelligent Information Retrieval, 0.253 Roland Kaltefleiter, 0.250 Richard A. Lethin, 0.248 Erik Schoenfelder, 0.245 Michael K. Cepek, 0.244 Alan Bawden, 0.243 Chris Claborne, AT&T Global Information Systems, 0.240 Dave Sill, 0.238 Jeff Hayward, 0.237 Mark Eichin, 0.236 Bill Broadley, 0.236 Confirmed Anonymous, 0.227 Lee Richardson, 0.223 Tomás Oliveira e Silva, 0.222 Theodore Ts'o, 0.214 R. Loader, 0.213

Mark V. Shaney, 0.212 Roger M. Levasseur, 0.212 Keith Bostic, 0.211 Peter Widow, 0.210 David Hollenberg, 0.198 Richard P. Brent, 0.198 James R. Fowler, 0.197 Leo Broukhis, 0.196 Bill Sommerfeld, 0.194 David Lee, 0.190 Unclaimed, 0.189 Chris Siebenmann, 0.189 Brian P. Fitzgerald, 0.186 David Buckley, 0.180 Jeff Woolsey, 0.174 Nick Holloway, 0.173 Brad Jones, 0.169 Andreas Steffen, 0.169 Serge Pachkovsky, 0.167 Peter Ilieve, 0.165 Stefan Gaertner, 0.165 Henri Cohen, 0.164 Tatu Mannisto, 0.160 Will Gilbert, 0.158 Andrew S. Krzywdzinski, 0.155 David J. Wright, 0.155 Henry Ellingworth, 0.154 root@cse.nau.edu, 0.151 Marc Brett, 0.150 Alan Barrett, 0.147 Scott Bennett, 0.146 anonymous grad student at New York University, 0.145 Mike Olson, 0.143 Alfred Wassermann, 0.142 Janne Himanka, 0.141 David L. Black, 0.136 Matthew Martin, 0.135 Jim McKim, 0.134 Craig Steinberger, 0.133 Ross Finlayson, 0.132 Confirmed Anonymous, 0.129 Narin Suphasindhu, 0.129 Woody Weaver, 0.129 Lloyd Miller, 0.127 Confirmed Anonymous, 0.126 Thomas Denny, 0.125 Roy Goodman, Digital Equipment Corporation, 0.124 Michael G. Reed, 0.123 Lance Cottrell, 0.123 David Comay, 0.123 Ronald M. Sorli, 0.123 Archie L. Cobbs, 0.121 Jim Hutchins, 0.120 Kelly Hall, 0.118 Unclaimed, 0.118 Allan Gottlieb, 0.116 Tomas Tengling, 0.116 Otto J. Makela, 0.114 Matthew Lyle, 0.113 Bret Ketchum, 0.113 Unclaimed, 0.111 Milt Mallory, Lincoln Ong, Ron Roberts, 0.111 Jeffrey I. Schiller, 0.110 Paul Rouse, 0.110 Mikko Siren, 0.105 John Oswalt, 0.102 Mike Chang, 0.101 Gary Oberbrunner, 0.097 Richard Doty, 0.095 Confirmed Anonymous, 0.095 Donald Nichols, 0.095 Bill Danielson, 0.094 Matthew Jackson, 0.094 Timothy A. Ramsey, 0.092 Unclaimed, 0.092 Olivier Bonaventure, 0.088 Felipe Castro A., 0.088 D.J. James, 0.087 Paul Zimmermann, 0.085 rps@ch.hp.com, 0.085 Hoyt A. Stearns jr., 0.083 Mansour Esmaili, 0.083 Durval Menezes (and others), 0.079 Dominik Gaillardetz, 0.077 Chuck Fee, 0.076 Unclaimed, 0.070 Robert Kulagowski, 0.069 Len Adleman, Dhiren Patel, 0.069 Geoffrey Baehr, 0.068 Jim Kirkpatrick, 0.068 Adam S. Nealis, 0.067 Frank Zizza, Samuel Hall, 0.066 Edward Mehlschau, 0.066 Richard Tobin, 0.064 Jonathan A. Jones, 0.060 Eng. Carmine Di Biase, 0.060 Aaron J. Heller, 0.060 Warren Gish, 0.058 Confirmed Anonymous, 0.058 Confirmed Anonymous, 0.057 Glenn Crocker, 0.057 Matthew J. Kidd, 0.056 Roy Kipp, 0.055 Renate Scheidler, 0.053 Jyri J. Virkki, 0.053 Michael D. Ernst, 0.053 Eric Johnson, 0.051 Jean-Pierre Szikora, 0.050 Jeff Murphy, 0.049 Gary A. Delong, 0.048 Wayne Schlitt, 0.048 Robin O'Leary, 0.048 Bryan Olson, 0.047 Greg Menounos, 0.047 Dave Madden, 0.046 Antonio Lioy, 0.046 Unclaimed, 0.046 Glenn Hollinger, 0.046 Jim Hickstein, 0.045 Confirmed Anonymous, 0.045 Unclaimed, 0.045 Mark Riordan, 0.045 Dirk Weigenand, 0.045 Richard Pinch, 0.044 Johannes Gronvall, 0.044 Russ Pagenkopf, 0.041 Robert Spellman, 0.040 Shawn Instenes, 0.038 Erol Basturk, 0.037 Christopher B. Moore, 0.037 Patrick Demichel, 0.036 Anish Mathuria, 0.036 Bob Clements, 0.036 Neil C. Kenneally, 0.036 Eugene W. Stark, 0.035 Unclaimed, 0.034 Marc Evans, The Destek Group, Inc., 0.034 Tom Epperly, 0.033 Unclaimed, 0.033 Peter Broadwell, 0.032 Geoffrey D. Bennett, 0.032 Daniel Tauber, 0.032 Mike Rose, 0.031 Christopher M. Conway, 0.031 James S. Vera, 0.031 Charles E. Spurgeon, 0.030 Manu Iyengar, 0.030 Peter Gutmann, 0.029 Bill Wade, 0.027 Pace Willisson, 0.027 Blair Zajac, 0.027 Larry Gadallah, 0.026 Tim Shepard, 0.025 Unclaimed, 0.025 Michael Brandmaier, 0.025 Andries E. Brouwer, 0.024 Alan Klietz, 0.024 Noam Shomron, 0.024 Thomas von Kamptz, 0.024 UBC IEEE Student Branch, 0.024 Unclaimed, 0.023 Carl Schott, 0.022 Paul J. Murphy, 0.022 Derek Clegg, 0.022 George M. Sipe, 0.021 Hugh D.R. Evans, 0.021 Confirmed Anonymous, 0.020 Chris Davey, 0.020 Dave Ihnat, 0.020 Unclaimed, 0.020 Alan Fothergill, 0.020 Unclaimed, 0.020 Tilman Enss, 0.019 Kevin Hood, 0.019 Steve Uurtamo, 0.019 Andres Kruse, NIKHEF, 0.018 Unclaimed, 0.018 Unclaimed, 0.018 Jeffrey C. Ollie, 0.018 Mike Griffin, 0.018 James Leinweber, 0.018 Christian Weisgerber, 0.017 Markus Lamminmaki, 0.016 Bob Proulx, 0.016 Mark Edward Stahl, 0.016 Uwe Hollerbach, 0.016 Matthias Rosenberger, 0.016 Stewart A. Levin, 0.015 Mike Pearce, 0.015 Cornelis van der Laan, 0.015 Unclaimed, 0.014 Robert A. Hayden, 0.013 Jerome Abela, 0.013 Kenneth S.A. Oksanen, 0.013 David Metcalfe, 0.013 Unclaimed, 0.013 Lance M. Cottrell, 0.013 Allen Condit, 0.012 Claude Barbe, 0.012 Unclaimed, 0.012 David L. Johnson, 0.012 Dave Mitchell, 0.011 Bob Kupiec, 0.011 Juergen Hannken-Illjes, 0.011 John Cownie, 0.011 Peter Clark, 0.011 Stefan Larsson, 0.011 Ken Hardy, 0.010 Nelson Bolyard, 0.010 Unclaimed, 0.010 Timo Suhonen, 0.010 Andrew Gray, 0.010 Unclaimed, 0.010 Malcolm Beattie, 0.009 James Beal, 0.009 Simon Foley, 0.009 Seth Finkelstein, 0.009 John Oswalt, 0.009 Unclaimed, 0.009 Richard G. von Blucher, 0.009 Don Hejna, 0.008 Craig Groeschel, 0.008 Per Bojsen, 0.008 Carl Witty, 0.008 D.J. James, 0.007 Patrick Wendt, 0.007 Unclaimed, 0.007 Jason Yanowitz, and 0.007 Mark Delany.

Acknowledgements are due to Bruce Dodson for factoring $p+1$ using Peter Montgomery's ECM program, to Allen McIntosh for his assistance with our initial curve-fitting experiments, to Ron Rivest for his permission to quote from [20], and to Rich Schroeppel for his helpful comments on the run-times in [21].

We are particularly grateful to Jeff Schiller at MIT, who made available to us computing resources, GBytes of disk space and frequent backups of our hard-earned data. The fourth author is especially thankful for being permitted to log in to `hot-spare.mit.edu` in order to monitor progress and to test and install new software to handle the ever-increasing torrent of incoming data.

References

1. D. J. Bernstein, A. K. Lenstra, *A general number field sieve implementation*, 103–126 in: [13].
2. 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.
3. J. D. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. **36** (1981) 255–260.
4. B. Dixon, A. K. Lenstra, *Factoring integers using SIMD sieves*, Advances in Cryptology, Eurocrypt '93, Lecture Notes in Comput. Sci. **765** (1994) 28–39.
5. B. Dodson, A. K. Lenstra, *NFS with four large primes: an explosive experiment*, in preparation.
6. M. Gardner, *Mathematical games, A new kind of cipher that would take millions of years to break*, Scientific American, August 1977, 120–124.
7. R. Golliver, A. K. Lenstra, K. S. McCurley, *Lattice sieving and trial division*, Algorithmic number theory symposium, Lecture Notes in Comput. Sci. **877** (1994) 18–27.
8. R. K. Guy, *How to factor a number*, Proc. Fifth Manitoba Conf. Numer. Math., Congressus Numerantium **16** (1976) 49–89.
9. D. E. Knuth, *The art of computer programming*, volume 2, *Seminumerical algorithms*, second edition, Addison-Wesley, Reading, Massachusetts, 1981.
10. B. A. LaMacchia, A. M. Odlyzko, *Computation of discrete logarithms in prime fields*, Designs, Codes and Cryptography **1** (1991) 47–62.
11. A. K. Lenstra, *Massively parallel computing and factoring*, Proceedings Latin'92, Lecture Notes in Comput. Sci. **583** (1992) 344–355.
12. 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.
13. A. K. Lenstra, H. W. Lenstra, Jr. (eds), *The development of the number field sieve*, Lecture Notes in Math. **1554**, Springer-Verlag, Berlin, 1993.
14. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, *The factorization of the ninth Fermat number*, Math. Comp. **61** (1993) 319–349.
15. A. K. Lenstra, M. S. Manasse, *Factoring by electronic mail*, Advances in Cryptology, Eurocrypt '89, Lecture Notes in Comput. Sci. **434** (1990) 355–371.
16. A. K. Lenstra, M. S. Manasse, *Factoring with two large primes*, Advances in Cryptology, Eurocrypt '90, Lecture Notes in Comput. Sci. **473** (1990) 72–82; Math. Comp., to appear.

17. U. M. Maurer, *Fast generation of prime numbers and secure public-key cryptographic parameters*, Journal of Cryptology, to appear.
18. C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, pp. 89–139 in: H. W. Lenstra, Jr., R. Tijdeman (eds), *Computational methods in number theory*, Math. Centre Tracts **154/155**, Mathematisch Centrum, Amsterdam, 1983.
19. C. Pomerance, J. W. Smith, *Reduction of huge, sparse matrices over finite fields via created catastrophes*, Experiment. Math. **1** (1992) 89–94.
20. R. L. Rivest, letter to Martin Gardner, 1977.
21. R. L. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM **21** (1978) 120–126.
22. R. C. Schroeppel, personal communication, May 1994.
23. A. Shamir, personal communication, April 1994.
24. R. D. Silverman, *The multiple polynomial quadratic sieve*, Math. Comp. **48** (1987) 329–339.

This article was processed by the authors using the TEX macro package from Springer-Verlag.