Factoring with two large primes

(Extended Abstract)

Arjen K. Lenstra
Bell Communications Research, room 2Q334
435 South Street, Morristown, NJ 07960
email: lenstra@flash.bellcore.com

Mark S.Manasse
Digital Equipment Corporation, Systems Research Center
130 Lytton Avenue, Palo Alto, CA 94301
email: msm@src.dec.com

Factoring with two large primes

The study of integer factoring algorithms and the design of faster factoring algorithms is a subject of great importance in cryptology (cf. [1]), and a constant concern for cryptographers. In this paper we present a new technique that proved to be extremely useful, not only to achieve a considerable speed-up of an older and widely studied factoring algorithm, but also, and more importantly, to make practical application of a new factoring algorithm feasible. While this first application does not pose serious threats to factorization-based cryptosystems, the consequences of the second application could be very encouraging (from the cryptanalysts point of view).

The technique has led to various new factorization records. It took us 50 days to factor a 107 digit number using our new version of the multiple polynomial quadratic sieve, and 60 days to factor a 111 digit number. This is quite a bit faster than the 120 days we needed for our previous 106 digit record with the old version [8]. We factored a 138 digit number using a new special purpose factoring algorithm [7]. Combined with our new technique this took approximately 50 days; it would have been impossible without.

For the 107 and the 138 digit number reported above we used the network of approximately 300 CVAX processors at Digital Equipment Corporation's Systems Research Center (SRC). For the other numbers a substantial amount of the computation was carried out by the participants to our electronic mail factoring network, as reported in [8].

Let n be some large integer to be factored. In cryptanalytic applications it is usually known that n is the product of two unknown primes of approximately the same size. In other cases (cf. [2]) one first has to decide if n is prime or composite. It is well known

I.B. Damgard (Ed.): Advances in Cryptology - EUROCRYPT '90, LNCS 473, pp. 72-82, 1991.

that this is usually not hard to do: use a probabilistic compositeness test [6] to prove that n is composite, and *believe* that n is prime if several attempts to prove compositeness have failed. In the latter case it remains to *prove* the primality of n, a problem for which efficient algorithms have been designed and implemented [3, 5, 10]. In the former case we should remark that the compositeness proof for n does in general *not* provide information which makes it easier to find a non-trivial factorization of n.

Suppose that the compositeness of n has been established beyond doubt. How can we find a non-trivial factorization of n? Again, if n is the modulus in some cryptosystem, it will have two carefully constructed large factors of approximately the same size, but for the rest n, or its factors, will have no specific properties that could make factoring easier. In such cases one has to resort to a *general purpose* factoring algorithm, a factoring algorithm that works no matter how *difficult* the number might be and whose running time is solely determined by the size of n.

For other numbers one could try the various *special purpose* factoring algorithms. These come in essentially two flavors. In the first place there are the methods that make use of properties an unknown factor of n might have. Because the factors are unknown, success is uncertain. It explains however why care has to be taken when designing a difficult composite n: if for instance one of the primes p dividing n is such that $p\pm 1$ is built up from small primes only, then n can be factored quite easily. In case of failure of this type of special purpose algorithms, a general purpose method should be applied, unless the second type of special purpose algorithm can be applied. This concerns methods that make use of the special form that n might have. Their run time, however, does not depend on any properties of the factors of n. An example of such a method is the *number field sieve* [7], and will be discussed below.

For cryptanalytic applications the general purpose algorithms are the ones to be studied. Until the summer of 1989 the best practical general purpose factoring algorithm was one of the multiple polynomial variations of the *quadratic sieve* algorithm (mpqs, for short) [11, 12]. The heuristic expected run time of mpqs is given by

(1) $\exp((1+o(1))(\log n)^{1/2}(\log\log n)^{1/2})$.

It is the only general purpose algorithm by which integers of more than 100 digits have been factored: a record factorization of a 106 digit integer in April 1989 took four months and used impressive computational resources [8].

Many general purpose factoring algorithms, and mpqs is no exception, work in two stages. In the first stage one collects so-called *relations*, in the second stage one uses the relations to find solutions $x, y \in \mathbf{Z}$ to $x^2 \equiv y^2 \bmod n$. Under reasonable assumptions each solution has a good chance to lead to a factorization of n by computing $\gcd(n, x \pm y)$. For mpqs a relation is an expression of the form

(2) $\quad v^2 \equiv \prod p^{e_{vp}} \bmod n,$

where $e_{vp} \in \mathbf{Z}_{\geq 0}$ and the product ranges over the primes (including -1) in the factor base (i.e., the number -1 and the primes less than some bound B). In mpqs, relations are found by means of a process called *sieving* (cf. [11]). On a fixed number of processors the number of relations found after t units of time will behave as $c \cdot t$, for some positive constant c depending on n. For a typical 100 digit number the number of elements of the factor base would be set to 50,000.

If the number of relations (found in the first stage) is more than the number of elements of the factor base, then a dependency modulo 2 can be found among the exponent vectors (in the second stage). Each such dependency leads to a solution to $x^2 \equiv y^2 \bmod n$, and thus to a chance of factoring n.

For mpqs the two stages are in theory asymptotically equally hard (they both take time (1), see [6] for an analysis and further description of the algorithm). For numbers in our current range of interest however the run time is entirely dominated by the first stage: if relations could be found twice as fast, the algorithm would run twice as fast. Gaining a factor two in the run time means, roughly, that we can factor integers having three more digits (cf. [1]).

A considerable speed-up of the first stage of the basic mpqs method can indeed be achieved quite easily by using the *large prime variation*, an idea that is already quite old [11]. It appears that the sieving stage of the algorithm can easily be changed so that it not only finds relations of the form (2), but relations of the following form as well:

(3) $\quad v^2 \equiv q_v \prod p^{e_{vp}} \bmod n,$

where q_v (the *large prime*) is a prime not in the factor base, and less than the square of the largest prime in the factor base. We will distinguish between relations (2) and (3) by calling them *small* (2) and *partial* (3) relations. So, a small relation is for instance given by a number v such that the least absolute residue $v^2 \bmod n$ completely factors over the factor base; for a partial relation there may still be *one* too large prime in the factoriza-

tion. One would expect therefore that the sieving stage produces many more partials than smalls, and that is indeed what happens.

Unfortunately, however, the partials themselves are worthless for the rest of the factorization process, unlike the smalls. It is only by *combining* the partials that they can be made into something that is as useful as a small relation. Suppose that both u and w give rise to partial relations, and suppose that we are so lucky that $q_u = q_w$. Multiplication of the two relations then gives

$$(4) \qquad (u \cdot w)^2 \equiv q_u^2 \prod p^{e_{up}+e_{wp}} \bmod n,$$

a so-called *big* relation. Since $u \cdot w$ can be divided by q_u (modulo n, unless n and q_u are not co-prime), such a big relation is just as useful as a small relation.

How much luck is involved in finding a big relation? The birthday paradox tells us that even a moderate number of partials will already lead to relations with matching large primes, and therefore big relations. To give an example (cf. [8]): of the approximately 320,000 relations gathered for a certain 100 digit number, 20,500 were small, and the remaining approximately 300,000 partials gave rise to 29,500 big relations. For this same 100 digit number the progress of the total number of relations as a function of the time is illustrated in Figure 1. For other number the graphs of the numbers of small and big relations behave similarly.

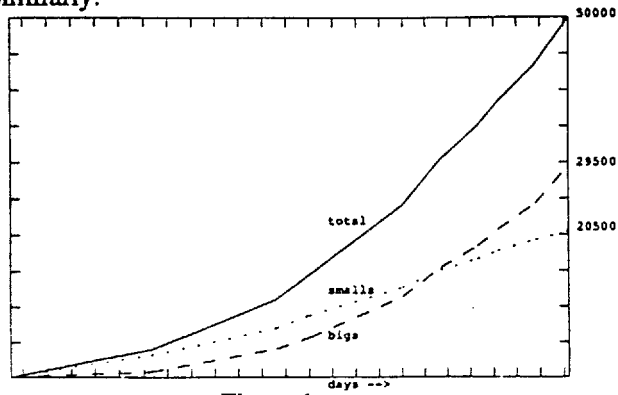

Figure 1

The large prime variation of mpqs affects the run time only in the $o(1)$ (cf. (1)). In practice it means a speed-up by a factor of approximately 2.5. Notice that it is straightforward to find the matching large primes: sort the partials according to their large prime, and match each pair of consecutive relations with the same large prime.

An obvious extension of the large prime variation is to allow *two* instead of only one large prime in a relation, i.e., a relation of the form

(5) $\quad v^2 \equiv q_{1v} q_{2v} \prod p^{e_{pv}} \bmod n,$

where the q_{iv} are primes not in the factor base. These relations will be called *partial-partials* (or *pp*'s for short). Notice that there should be far more pp's than partials. Like the partials, the pp's can be combined into relations that are just as useful as small or big relations. This can for instance be seen as follows. Identify each large prime with a vertex in a graph, and put an edge between two vertices each time they occur in the same partial-partial relation. A cycle in the resulting graph corresponds to a combination of pp's where the large primes occur an even number of times, which makes that combination useful for factoring. Notice that we are only interested in independent cycles, because dependent cycles would give rise to trivial dependencies in the matrix of exponents.

If the pp's are only combined among themselves, then this process can be seen as drawing random edges in a (big) graph. It is well known that it takes many edges before a cycle can be expected, so that the yield will be quite low. But if the pp's are combined with the partials the picture changes dramatically as can be seen in Figure 2, where the total number of combined relations as a function of the total number of partials and pp's is given for a 107 digit number we recently factored. The same algorithm can be used: view partials as pp's where one of the large primes equals 1, build the same graph, and look for independent cycles. Of course, in this way the big relations will be found as well. In the same picture we have given the number of big relations, i.e., combinations of two partials not involving pp's. Although the number of small relations is not a function of the number of pp's, the number of smalls and the total (i.e., smalls plus combinations) are given in Figure 2 as well. Compared to the ordinary large prime variation we achieved a speed-up of approximately a factor 2.5. The asymptotic run time for mpqs remains the same, i.e., using pp's only affects the $o(1)$ in (1). The factorization of the 107 digit number took approximately 50 days, on many fewer machines than the previous 106 digit record. We used a factor base of 65,000 elements.

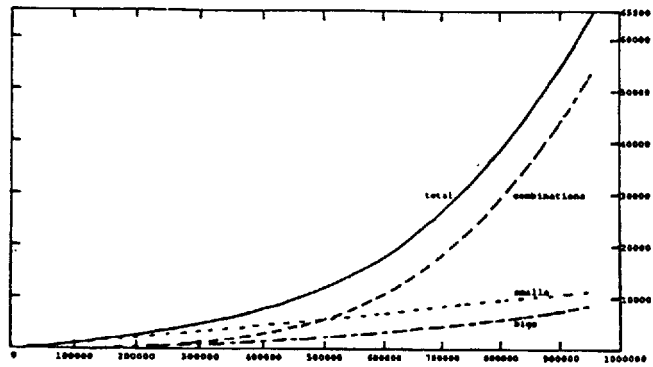

Figure 2

We got a similar picture for our current record general purpose factorization of a 111 digit composite factor of $2^{484}+1$:

$$2^{484}+1 = 17 * 353 * 209089 * 33186913 * 1251287137 * 2931542417 *$$
$$3860897986942821068655933036263824535533549879744 1 *$$
$$8469440919770574005769693908434732506225873994236085602665729.$$

The number we factored is the product of the last two factors. With a factor base consisting of 80,000 elements, we needed 14,300 small relations, and a total of 1,050,000 partials and pp's generating 66,100 cycles.

It can be seen in Figure 2 that combining partials and pp's only begins to pay off after finding an enormous number of relations. This could be one explanation for the fact that nobody used this method before: for smaller numbers fewer relations are needed, so that by the time the pp's have acquired enough weight, the other relations will already have done the job. For small numbers using pp's might even be counter-productive. This can be explained as follows. During the sieving stage the algorithm looks for so-called *reports*. For each report, the algorithm attempts to factor a certain number, using the primes in the factor base. If this attempt is successful, a small relation has been found. If the attempt fails, but the remaining cofactor is sufficiently small, then a partial relation has been found, because the cofactor is automatically prime. This implies that partial relations can be found at almost no extra cost; the only extra cost is caused by the fact that one has to allow for more reports to find more partials (and consequently more failures as well), but that will be made up for by the extra partials that will be found.

If one allows two large primes there should be even more reports, which makes the sieving stage slower, and there is the additional problem that the remaining cofactor must

be factored into two large primes (if it is not prime already, or too large to be interesting). So, one is faced with the problem of considerably more reports which are more expensive to process per report, and a considerable fraction of which will be worthless.

For small n there are many reports even if one allows only a single large prime. With two large primes processing reports would become a dominating term in the run time. Combined with the effect described above that pp's only start being useful if there are a lot of them, this implies that using two large primes makes the algorithm slower for relatively small n.

For larger n the sieving stage of the single large prime variation produces very few reports. There one can easily afford more (and more expensive) reports, without noticeable effect on the total sieving time. Because many relations are needed, one gets the chance to build a huge database of pp's, and consequently the algorithm will run faster. In the full version of this paper we will describe how we proceeded to find the combinations (i.e., the cycles in the graph), and how we coped with the gigantic amounts of data.

Notice that using two large primes in mpqs can be very advantageous, but that it also works perfectly well, though slower, without it. We now discuss a more important application of the same idea, and of the same cycle finder, that could have far-reaching consequences. This other application existed before the one described above, and actually made us realize that it would be a useful idea for mpqs as well.

In [7] a new factoring algorithm, the *number field sieve* (nfs), is presented. This algorithm is an example of the second type of special purpose algorithms as described above, because it only applies to n of the form $r^e - s$, where r is a small positive integer, and s is a non-zero integer of small absolute value. This is precisely the type of number that can be found in [2]. To factor an n of this form, the number field sieve runs in heuristic expected time

$$(6) \qquad \exp((c+o(1))(\log n)^{1/3}(\log\log n)^{2/3}),$$

with $c = 2(2/3)^{2/3} \simeq 1.526$. The algorithm has proved to be quite practical. Among others, we factored a 138 digit number that would have been absolutely impossible for mpqs. Using two large primes was of crucial importance to obtain this factorization.

The most exciting news about the nfs is that the algorithm can be generalized to integers of arbitrary form [4]. It is suspected [9] that the resulting general purpose factoring algorithm again runs in time (6), though with a slightly bigger value for c. The prac-

tical consequences remain to be seen. There is no doubt, however, that its chances of becoming practical are close to zero without the cycle finder.

For a description of the nfs we refer to [7]. Here we will only explain how two large primes can be used. Like mpqs, both the nfs and the generalized nfs consist of the two familiar stages, a relation collection stage, and an elimination stage. For nfs relations are expressions of the following form:

$$(7) \qquad \prod \phi(g^{v_g}) \equiv \prod p^{e_p} \bmod n .$$

The product at the left hand side ranges over elements g of some algebraic number field $K = \mathbb{Q}(\alpha)$ of norm equal to 1 or of prime norm $\leq B$, for some bound B, and the product at the right hand side ranges over the primes $\leq B$. The exponents v_g and e_p are integral, and ϕ is some homomorphism from $\mathbb{Z}[\alpha]$ to $\mathbb{Z}/n\mathbb{Z}$. We refer to [7] for the choice of K and ϕ. The form of n is important to be able to find a 'nice' number field. Relations are found by looking for coprime integers a and b such that the algebraic integer $a+b\alpha \in \mathbb{Z}[\alpha]$ and the integer $a+bm \in \mathbb{Z}$, with $m = \phi(\alpha)$, can be factored into small prime elements (and units) in $\mathbb{Z}[\alpha]$ and primes in $\mathbb{Z}$, respectively. This is done in the sieving stage of the nfs.

Suppose that the product on the left hand side ranges over B_1 elements, and the right hand side over B_2 elements. If we have more than B_1+B_2 relations, then we can, as in mpqs, use linear algebra to generate solutions to $x^2 \equiv y^2 \bmod n$, and thus factor n.

We have seen that for mpqs the number of small relations obtained is a linear function of the effort spent on finding relations. With the nfs the situation is different. There the yield becomes quite noticeably lower and lower, with the possibility of the unpleasant discovery, after spending years of CPU time, that the algorithm is not going to make it because the supply of solutions to (7) as generated by the nfs dries up. The theory simply tells us to start all over again with a bigger value for B. From a practical point of view this is less desirable: for numbers in our current range of interest B would have to be chosen so large that storing the exponent matrix, even in sparse form, becomes problematic, let alone finding a dependency among its rows.

So, to make the nfs practical, it is important to keep B as small as we can, while avoiding the problem of running out of solutions to (7). This can be achieved as follows. While sieving we not only collect relations as in (7), but we collect the following types of relations as well:

- As (7), but allow one large prime element at the left hand side, the *partial-fulls* or *pf*'s;
- As (7), but allow one large prime at the right hand side, the *full-partials* or *fp*'s;
- As (7), but allow both a large prime element at the left hand side and a large prime at the right hand side, the *partial-partials* or *pp*'s.

Relations as in (7) will be called *full-fulls* or *ff*'s.

The sieving stage can easily be changed so that it not only collects the ff's, but the pf's, fp's and pp's as well. For pf's and fp's this is trivial, as it was for partial relations in mpqs. For pp's this follows from the fact that the large prime element and the large prime involved come from different numbers (namely from $a+b\alpha$ and from $a+bm$, respectively). So, the problem of slower performance that mars finding pp's in mpqs when applied to comparatively small numbers does not occur here.

Clearly, the sieving stage should find many more pf's and fp's than ff's, and even more pp's. The pf's can be combined among themselves, just as the partials in mpqs, with the difference that we divide pf's with the same large prime element instead of multiplying them to avoid the problem of computing a generator for the large prime ideal. Similarly, fp's with the same large prime can be combined, either by multiplication or by division, to produce a useful relation. And the pp's, finally, can be used in almost the same way as the pp's in mpqs. The difference is that the pp's now give rise to a bipartite graph (with vertices identified with prime elements in $\mathbb{Z}[\alpha]$ connected to vertices identified with primes in $\mathbb{Z}$), plus one extra vertex (identified with 1) to put the pf's and the fp's in the same graph.

To give some examples, for a certain 122 digit number we needed a total of 49,000 relations. After two weeks sieving (on many machines simultaneously) we had gathered 10,688 ff's, 116,410 pf's, 103,692 fp's and 1,138,617 pp's. By that time it had become clear that our choice of B was too low to factor the number using only ff's, because the supply of ff's was drying up rapidly. The same was true for the pf's and fp's. Although the 116,410 pf's gave already 5,341 combinations, and the 103,692 fp's gave 5,058 relations, it was clear that they were coming in too slowly to make our choice of B feasible for this number, at least without using pp's. Using the cycle finder we found more than 28,000 independent cycles involving pp's, which was enough to factor the number. It took five days (on a single machine) to find a dependency in the resulting matrix. We are not sure what value for B we should have chosen to obtain this factorization without

using pp's, but it is unlikely that we could have factored the number within a reasonable amount of time in that case.

For a 138 digit number, it took seven weeks to gather 17,625 ff's and a total of 1,741,365 pf's, fp's, and pp's, which gave 62,842 combinations. It took two weeks to process the resulting 80,000×80,000 matrix. Without pp's we would never have succeeded: B would have to be taken so large that the sieving would take almost forever, and we would not even be able to store the sparse representation of the resulting matrix.

Notice that relations that follow from combinations lead to denser rows in the matrix of exponents than the ff's. So, although the combinations are just as useful for factoring as the ff's, they lead to a denser matrix, and therefore to a slower second stage. The same holds for mpqs. However, this is a small price to pay if the only alternative leads to unsurmountable problems.

As remarked above, we gained our first experience with pp's because we had to while experimenting with the nfs. This naturally led us to the application of the same idea in mpqs.

We have seen that this relatively simple technique of finding cycles among partial and partial-partial relations is very useful for mpqs, and of great importance to make nfs practical. If the generalized nfs ever becomes practical, there can be little doubt that an important role will be played by the partial-partial relations. We therefore feel that it is an important technique that should be brought to the attention of everyone interested in factoring.

References

1 G. Brassard, *Modern Cryptology*, Lecture Notes in Computer Science, vol. 325, 1988, Springer-Verlag.

2 J.Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, S.S. Wagstaff, Jr., *Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers, second edition*, Contemporary Mathematics, vol. 22, Amer. Math. Soc., Providence, Rhode Island 1988.

3 W. Bosma, M.-P. van der Hulst, A.K. Lenstra, "An improved version of the Jacobi sum primality test," in preparation.

4 J. Buhler, H.W. Lenstra, Jr., C. Pomerance, in preparation.

5 H. Cohen, A.K. Lenstra, "Implementation of a new primality test," *Math. Comp.*, v. 48, 1987, pp. 103-121.

6 A.K. Lenstra, H.W. Lenstra, Jr., "Algorithms in number theory," in: J. van Leeuwen, A. Meyer, M. Nivat, M. Paterson, D. Perrin (eds), *Handbook of theoretical computer science*, to appear.

7 A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, J.M. Pollard, "The number field sieve," to appear.

8 A.K. Lenstra, M.S. Manasse, "Factoring by electronic mail," Proceedings Eurocrypt '89.

9 H.W. Lenstra, Jr., personal communication.

10 F. Morain, "Primality testing: News from the front," Proceedings Eurocrypt '89.

11 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*, Mathematical Centre Tracts 154, 155, Mathematisch Centrum, Amsterdam, 1982.

12 R.D. Silverman, "The multiple polynomial quadratic sieve," *Math. Comp.*, v. 48, 1987, pp. 329-339.