Status Report on Factoring
(At the Sandia National Laboratories)*

James A. Davis, Diane B. Holdridge and Gustavus J. Simmons

Sandia National Laboratories
Albuquerque, New Mexico 87185

Introduction

It is well known that the cryptosecurity of the RSA (Rivest-Shamir-Adleman) two key cryptoalgorithm [1] is no better than the composite modulus is difficult to factor. Except for one special case, the converse statement is still an open and extremely important question. It is not so well known, perhaps, that there are several other crypto-like schemes whose performance is also bounded by the difficulty of factoring large numbers: the digital signature schemes of Ong-Schnorr [2], of Ong-Schnorr-Shamir [3] and of Schnorr [4], the oblivious transfer channel of Rabin [5] and the subliminal channel of Simmons [6] to name only a few. The point is that the difficulty of factoring large integers has become a vital parameter in estimating the security achievable in many secure data schemes -- and conversely factoring techniques are potentially a tool for the cryptanalyst if the cryptographer misjudges the difficulty of factoring a composite number on which he bases a system.

The Sandia National Laboratories have already fielded several secure data systems that are dependent on the difficulty of factoring for their security [7,8,9] and at least as many other applications are approaching

* This work performed at Sandia National Laboratories supported by the U. S. Department of Energy under contract No. DE-AC04-76DP00789.

T. Beth, N. Cot, and I. Ingemarsson (Eds.): Advances in Cryptology - EUROCRYPT '84, LNCS 209, pp. 183-215, 1985.

realization. As a result, a concerted research effort was initiated in 1982 in the Mathematics Department at Sandia to define as sharply as possible the bounds on the computational feasibility of factoring large numbers, using the most powerful computers available -- as efficiently as possible -- with the factoring algorithms being carefully matched to the architecture of the machine on which the algorithm was to be run [10]. Our primary objective in this paper will be to present an overview of the advances in factoring resulting from this research. Later, we shall discuss in detail the mathematical and coding advances themselves. Suffice it to say that a roughly three-order of magnitude improvement in factoring -- as measured by the time required to factor a particular size number -- has been achieved by the Sandia researchers over what was possible (and well benchmarked) a few years ago. This is a combined effect due in part to a new generation of computers with much increased computing power and especially due to the unique architecture of the Cray family of machines, in part due to substantial advances in factoring algorithms and finally -- and equally significant -- in part attributable to the efficiency with which the algorithms have been coded for the specific computers. Our secondary objective will be to separate out the contributions of these three factors (to factoring progress) in order to both understand how the improvements of the past three years were achieved as well as to project what the state of the art in factoring is likely to be 5 to 10 years from now.

An Overview

The easiest question to ask concerning integer factoring and the hardest to answer, is; "How large a number is it computationally feasible to factor using a general purpose factoring routine?" Figure 1 gives one

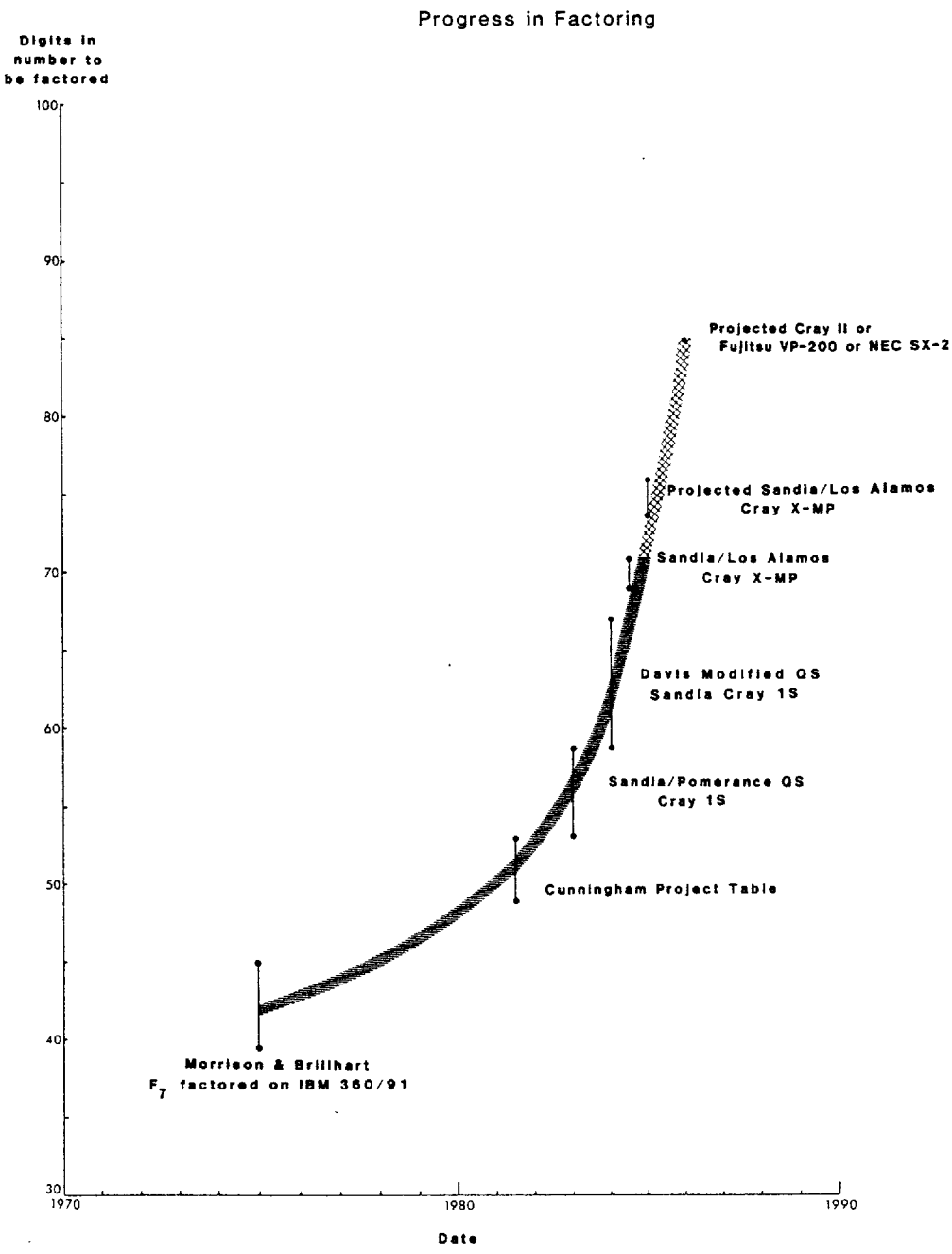
Progress in Factoring
Digits in number to be factored
100
90
80
70
60
50
40
30
1970
1980
1990
Date
Projected Cray II or Fujitsu VP-200 or NEC SX-2
Projected Sandia/Los Alamos Cray X-MP
Sandia/Los Alamos Cray X-MP
Davis Modified QS Sandia Cray 1S
Sandia/Pomerance QS Cray 1S
Cunningham Project Table
Morrison & Brillhart
F_7 factored on IBM 360/91

Figure 1.

answer showing the record size numbers that were factored as an approximate function of the year in which the factorization occurred -- over a period of roughly a decade. The data in Figure 1 were selected as being the most indicative of the state of the art in factoring at the time, either because the factorization was generally acknowledged as a major achievement or advance or, as in the case of the Cunningham Table Project [11], because of the thoroughness with which the benchmark was defined. The reader should be aware, however, that these data points are virtually impossible to cross-compare. Within the Sandia data alone the times required to factor the reported numbers range from 7.2 minutes to 32.3 hours, involve three generations of continuously changing computer codes and were run on either a Cray 1S or a Cray X-MP computer. The other data have even more variability. The algorithms were run on widely different machines -- some in a dedicated computing environment and others in a time sharing mode. In some instances, the total time required was reported, in others only that time required for the part of the algorithm that was of primary concern to the research. At this late date, there is no way that these results can be "normalized" to make them directly comparable. Instead, we exhibit Figure 1 as the best indicator we have of the progress in factoring during the past decade -- with a strong caution to the reader to not try to read more into (or from) the figure than this.

The first data point is the landmark factorization of F_7, a 39-digit number, by Morrison and Brillhart [12] using their continued fraction algorithm. This algorithm was to become the progenitor of a series of steadily improving continued fraction algorithms that dominated the factoring scene until Pomerance's quadratic sieve [13] was first implemented at Sandia in 1982 [10]. Using an IBM 360/91 over a period of several weeks,

Morrison and Brillhart found the necessary number of completely factored quadratic residues in a total CPU time of only 90 minutes. They don't report the time required to carry out the Gaussian elimination (in 2700 variables -- primes) but it must have been large. Based on recent discussions with John Brillhart, it appears that their technique and machine could have factored numbers in the mid-forty digit range in times comparable to the average of the times required for the two dozen or so factorizations that make up the Sandia data points. The error bar on the Morrison and Brillhart data point therefore reflects a rough attempt to show the true capability of this factorization technique.

Unquestionably, the most extensive -- and up to date -- compilation of integer factorizations ever made is the Cunningham Project Table [11] published by the AMS in 1983. As the authors say in the introduction, "The present tables are now at the limit of what can be done by factoring through 50 digits ...." The mid-1981 data point representing this benchmark in Figure 1 indicates the spread above and below this 50-digit figure accounted for by the variation in difficulty of specific numbers. Roughly speaking, the Cunningham Project Table established a well defined standard of the computational feasibility of factoring any 50-digit number in at most one day's computing time. This was the state of the art in the fall of 1982 when the quadratic sieve in the form originally proposed by Carl Pomerance [13] was implemented by Davis and Holdridge at Sandia on a Cray 1S. The Sandia effort was prompted by the recognition by Simmons, Tony Warnock of Cray Research and Marvin Wunderlich that the Cray's ability to efficiently pipeline vector operations on vectors containing thousands of elements could be matched to the sieving operation that was the heart of the quadratic sieve factoring algorithm. The immediate results were start-

ling. A pair of 51 and 52-digit numbers — taken from the composite cofactor list in the Cunningham Project Table that gave them a recognized certificate of difficulty — were factored in under two hours. This represented a speed improvement of better than an order of magnitude on the first attempt over what had been possible only a year earlier when the Cunningham Project Table was sent to press. This algorithm, using a very memory efficient Gaussian elimination routine for binary matrices devised by Parkinson and Wunderlich [16], was found to have a feasible range of 50-58 digits, i.e., it could factor up to 58-digit numbers in approximately a day's CPU time.

The next big advance occurred in 1983 when Davis discovered the special q variation to the basic quadratic sieve [10]. This innovation is so vital to the Sandia advances, and to the most recent factoring results shown in Figure 1, that it will be discussed in detail later. We also give some precise cross-comparisons of the time required to factor numbers using the quadratic sieve, both with and without the special q variation, later in this section, but roughly speaking this improvement bought another order of magnitude improvement in the speed of factoring.

The last two data points in Figure 1 are another factor of six or seven removed from the points in the error bar for the special q algorithm. This is due to the optimization of the coding of the special q algorithm for the Cray -- attributable in large part to an improved search algorithm developed by Tony Warnock. In addition, Holdridge found that by "unrolling" the nested loops in the code the running times could be substantially improved. In other words, the six-fold improvement between the special q case and the 69 and 71-digit examples is primarily due to the substantially improved efficiency with which the computers were coded and used.

The research on factoring at the Sandia National Laboratories has been proof tested at each stage of algorithm development on numbers that were left unfactored in [11] and which were cited as being of either extraordinary interest or difficulty to factor or both. For example, in [11], there is a table of the "Ten 'Most Wanted' Factorizations" that included as the first two entries the composite cofactors of the only two surviving unfactored composite numbers from Mersenne's 1640 list; $2^{211}-1$ and $2^{251}-1$. Since this list was essentially an open challenge to the factoring community, we have responded by factoring all ten of them (nine of them for the first time). In the "shorthand" notation of [11], the numbers and the vital statistics of their factorization are shown in Table I. 2,211- C60 denotes the 60-digit composite cofactor of $2^{211}-1$, etc. The cofactors themselves are

Table I.

10 Most Wanted Factorizations

CPT No.	Digits	Cunningham Designation	Sieve Time (hrs)	Process Time (hrs)	Program Configuration
1	60	2,211-	22	.25	(a)
2	69	2,251-	31.9	.4	Sq
3*	54	2,212+	.1	.9	Sq, SGE
4	55	10,64+	4.3	.1	BQS
5	61	10,67-	1.0	.22	Sq
6	71	10,71-	8.75	.75	Sq, SGE
7	58	3,124+	1.6	.2	Sq
8	53	3,128+	5.9	.15	BQS
9	67	11,64+	15	.34	Sq
10	55	5,79-	.66	.33	Sq

* This number originally factored by Sam Wagstaff. The times shown for 3 are for the Sandia factorization. The other nine were all first factored at Sandia.

(a) 18.5 hours sieving using basic algorithm obtained 1/2 needed relations 3.5 hours using special q's conpleted the sieving.

BQS = Basic Quadratic Sieve (+)

Sq = Special q (□)

SGE - Segmented Gaussian Elimination (△)

tabulated in [11]. Figure 2 plots the total computing time required to factor these numbers. As already mentioned, there have been three distinct generations of quadratic sieving algorithms, although refinements and improvements have occurred steadily in each generation of software. The + symbol denotes factorizations made with the original quadratic sieve, □ the special q algorithm and △, the segmented (partioned matrix) Gaussian elimination codes that make it possible to handle much larger prime bases than would otherwise be possible. The 54-digit outlier (2,212+) is the result of factoring a small number using the partitioned matrix code, so that almost all of the time shown was overhead spent in moving blocks of the matrix into and out of memory. It is included for completeness, but

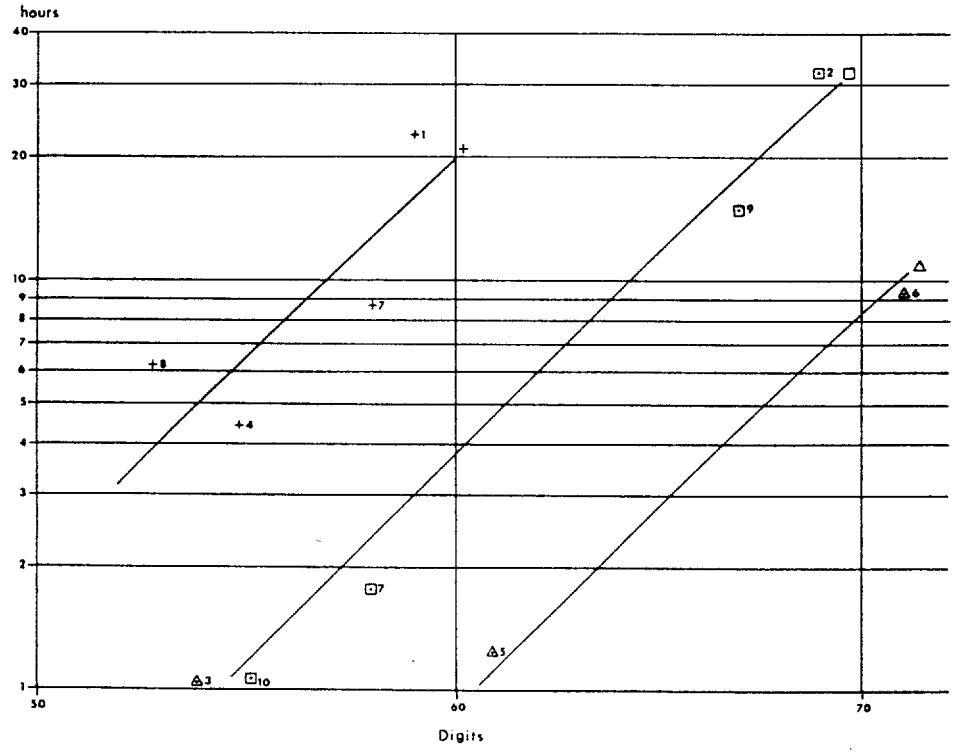

Figure 2.

would have taken roughly the same time to factor using only the special q algorithm. Incidentally, the approximating curves are simply fits of $(L(n))^c$ where $L(n) = e^{\sqrt{\log n \times \log\log n}}$ is the bound that most of the general purpose factoring algorithms seem to obey [13].

Especially noteworthy is the 58-digit number (#7, 3,124+ from [11]) that was factored twice; first using the basic quadratic sieve in a time of 8.78 hours and then again using the special q algorithm in 1.76 hours with a five-fold improvement in speed. This provides a crisp cross-comparison of the algorithms since both factorizations were done on the Cray 1S with codes developed by Holdridge within a very short time span. Hence, the improvement in this case is directly attributable to the mathematics (special q algorithm). An even more spectacular, but also more difficult to interpret cross-comparison is possible. In [13] Pomerance discusses the benchmark 49-digit number $\frac{3^{121}-1}{(3^{11}-1)11617}$ factored by Sam Wagstaff in 70 hours of computing and projects that it might be possible by various refinements (such as the early abort technique) to reduce the running time to as little as 20 hours. The latest generation of Sandia algorithms factored this number in 4 minutes and 34 seconds: a ratio of 920 to 1 in computing time! Admittedly, this timing comparison is hard to interpret since different machines and different factoring algorithms were used, but the comparison supports our earlier statement that a roughly three-order of magnitude overall improvement in the speed of factorization has been achieved. Other comparisons yield similar results.

Twenty-five large numbers (>40 digits) have been factored at Sandia -- plus many other smaller numbers for which the overhead obscures the time actually spent in factoring. Figure 3 shows a least squares fit of $(L(n))^c$ to the data on numbers of at least forty digits for the three generations of algorithms; marked +, $\square$ and $\triangle$ as before. Note, however, that in an

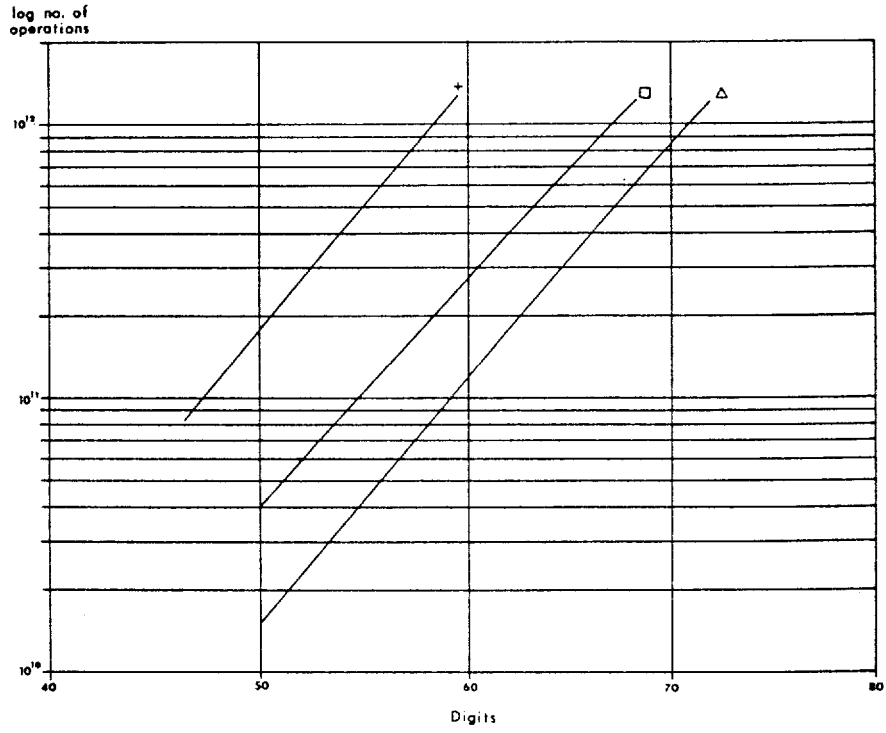

Figure 3.

effort to make the data machine independent, we have plotted the number of elementary machine operations (shifts, adds, XOR, etc.) rather than the total time required to factor a number. The △ curve is translated upward in Figure 3 compared to the same curve in Figure 2 since the Cray X-MP has a basic clock frequency of 105 MHz compared to the 80 MHz clock frequency for the Cray 1S, so that the elapsed time (Figure 2) for a given number of elementary operations on the X-MP is roughly 3/4 of what it would be on the 1S. The most obvious conclusion to be drawn from Figure 3 is that the Sandia work has -- for a given number of machine operations -- roughly increased the size of the number that can be factored by thirteen digits. This may not sound like much of an improvement, but over the range from 40 to 75 digits -- essentially independent of the algorithm used -- for each

three-digit increase in the size of the number to be factored, the time required roughly doubles. This translates into slightly more than a 20-fold improvement in factoring resulting from the Sandia work, independent of the machine. This latter statement assumes that the machine is vectorized so that the quadratic sieve can be accommodated efficiently and also that the memory is organized in such a way that data can be "streamed" through an arithmetic unit and back into memory, etc., as is needed for an efficient implementation of a quadratic sieve. The Crays have this type of architecture, but so does the NEC SX-2, the Fujitsu VP-200 and the Hitachi S-810 [14,15].

A sort of "sound barrier" in computing is 10^{12} operations. At present this is a generally accepted dividing line between what is computationally feasible and infeasible. Figure 4, taken from the same data shown in Fig-

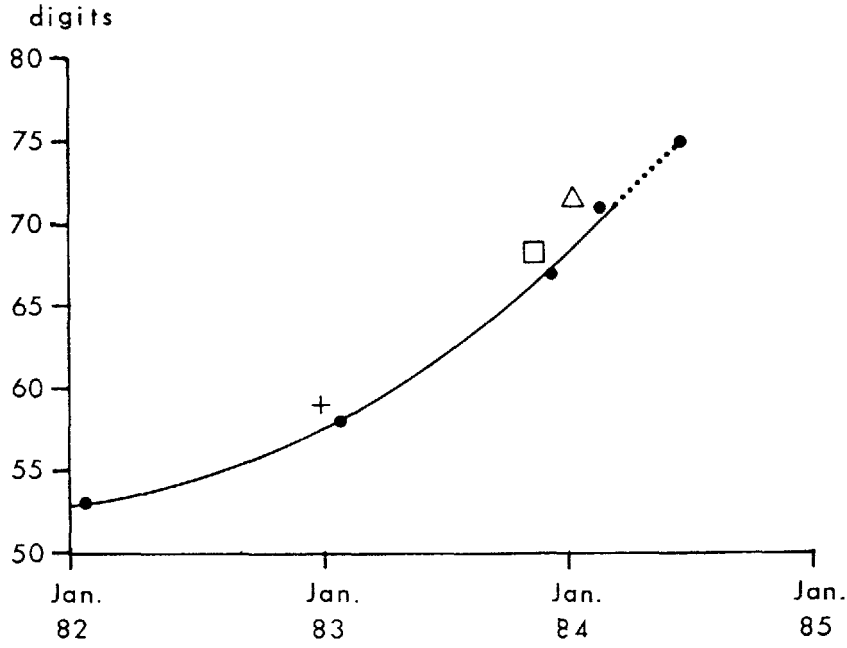

Figure 4. Size of composite "hard" number factorable by 10^{12} operations.

ures 1 and 3, shows how large a composite "hard" number could be factored using 10^{12} operations with the various generations of Sandia quadratic sieving algorithms and codes. Again, the roughly thirteen-digit overall improvement mentioned earlier can be seen to hold at 10^{12} operations decreasing to roughly ten digits improvement at 10^{10} operations -- the difference being due to the relative effect of the fixed overhead in the computation.

The third factor, in addition to the algorithmic improvements and the advances in the speed and power of the machines on which the algorithms are run, that has made a major contribution to speeding up the factorization of large numbers is the architecture of the Cray family of computers (or of the Cray-like vectorized machines such as the NEC SX-2, the Fujitsu VP-200 and the Hitachi S-810). We presuppose here that the reader is either already acquainted with the essential steps in factoring using a quadratic sieve, or else that he will return to this portion of the paper after having read the detailed discussion of the algorithm steps. Roughly speaking there are three major time-consuming steps. One involves the subtraction of the logarithm of a prime number, p_i, from on the order of $(1/p_i) \times 10^{10}$ locations for the largest numbers factored. Another requires forming the ring sum (exclusive OR, or $\oplus$) of a pair of binary vectors 7-15,000 bits long several million times. The third task, which has often been described as searching for a needle in a 10^9 haystack, is a search over $\approx 10^{10}$ locations looking for linear dependencies, where we expect on average 20 "hits" in the 10^{10} items searched. To appreciate the impact of the computer architecture on the speed of execution -- consider the first operation described above in which the same quantity, $-\log p_i$, is to be added to a string of memory locations that can be indexed in such a way that the locations to which $-\log p_i$ is to be added differ by a constant

p_I^j. The total string length is $\approx 10^{10}$. In a machine of more or less conventional architecture in which data is fetched from memory, operated on in the arithmetic unit (AU) and the result then returned to memory, this sort of operation is slow. Programmed optimally on a CDC 7600 only a 1 megabit per second effective throughput is possible. The Cray however has the ability to "stream" information from memory through the AU and back into memory for a fixed operation without pausing for fetch, store or interpret states. As a result, we can carry out this operation,

```
      DO 10, I = J, N, p_I^j        N ≈ 10^10
10    A(I) = A(I) - X(p_I)   ,
```

where $X(p_I)$ is the logarithm of p_I, at 1/2 the clock rate of 80 MHz on the Cray 1S and at the full clock rate of 105 MHz on the Cray X-MP. In other words, the architecture alone has accounted for a speed up of nearly forty times (Cray X-MP with 105 MHz clock versus CDC 7600 with a 37 MHz clock rate) for this type of operation! In order to exploit the capability to stream information from the memory through the AU and back into memory, the algorithm must have many long strings on which a fixed operation needs to be performed. The recognition that quadratic sieving could be organized in such a way that this feature of the Crays could be exploited is what prompted the Sandia research in the first place.

The exclusive or operation

$$A(I) = A(I) \oplus B(I)$$

goes at the same rate as the subtraction, i.e., the Cray 1S streams at 1/2 × 80 MHz while the Cray X-MP can stream data at 1 × 105 MHz. The search operation in either of the Crays has an overhead that only allows a throughput of 2/3 of the clock rate, i.e., 2/3 × 80 MHz for the Cray 1S

or 2/3 × 105 MHz for the Cray X-MP.

In addition to using the ability of the Crays to stream data, Holdridge did a timing analysis and found that if the major sieving loop was "unrolled" that the same computation could be carried out even faster.

As a result of the timing analysis of the sieving code it was also determined that a great deal of time was spent in searching. Once the sieving is done those vector entries that have reached a specified limit must be found and saved. The search, written in Fortran with an "if" statement was not vectorized by the Cray compiler. The search is now done by a Cray Assembly Language (CAL) subroutine which does use the vectorization capability and is much faster.

The bottom line, when all of these refinements are included and when one weighs the efficiencies for the various operations with the relative times spent in carrying out the associated calculation, is that the Cray 1S, running the quadratic sieve, has a throughput (bits of meaningful information processed per second) of 1/4 × 80 MHz while the Cray X-MP achieves 3/4 × 105 MHz. Both of these figures are quite impressive since they indicate that the coding is exceptionally taut -- so much so that Cray scientists have said that these codes come the closest to running the Crays "flat out" of any codes they know of. The point is that since no code can have a throughput greater than the clock rate, and since the throughput with these codes (especially on the X-MP) is so close to the clock rate, there is only a marginal improvement possible from further refinements of the coding -- for the present factoring algorithms. Almost an order of magnitude of the total advance in factoring achieved at Sandia is attributable to the efficiency with which the Crays are being used, i.e., to the tautness of the codes.

We can extrapolate the future of factoring a short distance into the

future with relatively high confidence. First, the Cray X-MP is a dual processor machine in which the present Sandia code has only used one of the processors. Preliminary work on splitting up the main parts of the quadratic sieving calculation so that two processors can be efficiently employed -- a nontrivial task incidentally -- suggests that it may be possible to gain a factor of ≈ 1.7 in computing effectiveness by using the X-MP to its fullest. Using the rule of thumb that doubling the computing time roughly equates to increasing the size of the number that can be factored with a fixed amount of work by three digits -- taking advantage of the dual processor capability of the Cray X-MP should make it possible to factor numbers of 73-74 digits in the same time required to factor the 71-digit number using a single processor. Another way of stating this result is; with the present code and using the Cray X-MP, 75-digit numbers should be factorable in roughly a day's computing time.

Looking at the next generation of vectorized machines -- especially the Cray II and also the Fujitsu VP-200 or the NEC SX-2 [14,15], all will have a 256 million word high-speed memory compared to the four million word memory on the Cray X-MP used in the research reported here. The Cray II has a projected arithmetic capability of 2000 megaflops (millions of floating point operations per second) while the Japanese machines have 533 and 1300 megaflops respectively compared to $\approx$ 100 megaflops for the Cray X-MP. Perhaps more significantly for the quadratic sieve algorithm, all have an improved vectorization capability; 80 K for the SX-2 and 64 K for the VP-200 compared to the 4 K capability of the Cray 1S or 2×4K of the Cray X-MP. All of these factors when combined suggest that the Cray II and probably the Fujitsu VP-200 or the NEC SX-2 will be roughly eight to nine times more effective in factoring using the quadratic sieve than is the Cray X-MP. This translates into an increase in the size of the numbers that can be factored of $\approx$ 10 digits. We therefore feel quite confident in pro-

jecting that 85-digit numbers will be factorable in a day's time using the machines that will be available in the next year or so as indicated in Figure 1.

Beyond that point, we leave it to the reader to draw his own conclusions. It is unlikely, however, that either of the curves in Figures 1 or 4 showing recent progress in factoring will suddenly go "flat", but whether the exponential rate of change will continue is impossible to predict. What does appear plausible to predict, though, is that it will be feasible to factor 100 digits by the end of the decade, i.e., by 1990.

Fanciful Factoring

Most general purpose factoring algorithms (continued fraction, Schroeppel's sieve and the various quadratic sieves) depend for their success on the following simple observation. In the ring of residues modulo a composite number n, any quadratic residue, y, i.e., a residue that is the square of some other element in the ring, has at least four "square roots" -- and perhaps many more depending on the choice of y and on the prime decomposition of n. If there existed an oracle that when presented with a quadratic residue, y, would pronounce a square root of y, then n could be factored with probability that goes to 1 exponentially fast. For example, if $n = pq$, p and q distinct primes, and $y = x^2 \pmod n$ where x has the unique representation $x = ap + bq \pmod n$, where $0 \leq a < q$ and $0 \leq b < p$, then y has the four square roots $(\pm a)p + (\pm b)q$ where we interpret $-a = q-a$ and $-b = p-b$. To factor n using the services of the oracle, choose $x = ap + bq$ (at random) and compute the quadratic residue $y \equiv x^2 \equiv a^2p^2 + b^2q^2 \pmod n$. We, of course, do not know a, p, b or q since we don't yet know the factorization of n, but we do know x and y. The oracle when presented with y, would with probability 1/2 pronounce either $y^{1/2} = ap + bq$ or $y^{1/2} = (-a)p + (-b)q$ in which case we learn nothing

about the factorization of n. On the other hand, with probability 1/2 the oracle would pronounce either $y^{1/2} = ap + (-b)q$ or $y^{1/2} = (-a)p + bq$. In which case the two greatest common divisors;

$$(x+y^{1/2}, n) \quad \text{and} \quad (x-y^{1/2}, n)$$

would be either p and q or else q and p, respectively, depending on which root the oracle chose.

All of the general purpose factoring algorithms mentioned cause the computer to function (ultimately) in the same way as our fancied oracle. The main difference is that instead of getting back a square root as the response to a submitted quadratic residue, the algorithm yields a sequence of intermediate answers, that ultimately amount to one of the oracle's responses. Just as in the case of the oracle, a quadratic residue, Q_i, is presented to the algorithm -- but the response is not (except in the rarest of cases) a square root of Q_i, but rather the prime decomposition of Q_i, in which some of the prime factors may occur to an odd power. Hence it is computationally infeasible to infer a square root of Q_i from the response, since this would be equivalent to being able to factor n. If after sufficiently many responses, however, a subset of the Q's can be found for which each of the primes that has occured as a factor in some one of the Q's has occurred an even number of times in all, then we are able to effectively recreate one of the oracle's responses. Since the product of the Q's is a quadratic residue of a root that we know and the square root of the product of the primes is trivially the product of each of the primes raised to half of its even exponent, it is also feasible to calculate a square root. Just as in the case of the oracle, when $n = pq$ there is only a 50-50 chance that this will lead to a factorization of n with comparable probabilities for other composite n, but this is the essential notion underlying the various factoring schemes.

Quadratic Sieving: Plain

Given an odd number, n, to be factored, the basic quadratic sieving scheme [13] calculates a sequence of (relatively) small quadratic residues

$$Q(x) = (x+m)^2 - n \qquad (1)$$

where $m = [\sqrt{n}]$. If $|x| \leq B$ and $B << \sqrt{n}$, then Q(x) will be "close" to $\sqrt{n}$. It is important to keep Q(x) small since the algorithm attempts to factor Q(x) over a prescribed -- but restricted -- set of primes. This set of possible factors of Q(x) consists of precisely those primes for which n is a quadratic residue, i.e., 2 and the odd primes, p, for which the Legendre symbol (n/p) = 1. Fortunately, the Legendre symbol is easy to calculate in a manner similar to the Euclidean algorithm for finding the greatest common divisior, so that it is computationally easy to find the t-1 smallest odd primes, p_i, for which $(n/p_i) = 1$. The set of t+1 elements $(-1, 2, p_i)$ we shall refer to as the factor base. In order for the algorithm to succeed, we must find sufficiently many quadratic residues, Q(x), that factor completely into factors in the factor base so that it is possible to find some subset of the Q(x) among which the prime factors have all occurred an even number of times. The justification for referring to the procedure as a sieve is now easy to see. If $p^\alpha | Q(x)$ for some x, then $p^\alpha | (Q(x \pm hp^\alpha)$, $h = 0,1,2,\ldots$, hence the division of the resulting sequence of quadratic residues can be performed by a sieve-like operation at argument values indexed in an arithmetic progression with spacing of p^α. One of the primary reasons for the speed of the quadratic sieving algorithm is that instead of having to carry out multiple precision trial divisions as is required in some of the other general purpose factoring algorithms, we can use single precision subtraction of approximate logarithms on the $Q(x \pm hp^\alpha)$, i.e., at only those positions where it is

already known that p^{α} is a divisor.

Since we must ultimately be able to combine a subset of the factored residues by multiplication to form a perfect square, i.e., to simulate a response by the oracle, we need to find a linear combination of the exponents for the primes appearing in the various factorizations such that the sum is even in each entry (for each prime). In order to have a reasonable chance of finding such a dependency we should have approximately as many completely factored residues as we have primes in the factor base. One might conclude from this, that t should be small. However, if we take t to be too small, then a given residue is not likely to factor. On the other hand, if we take t to be too large, we spend more time sieving and will have to find many more factorizations in order to be able to find a linearly dependent subset. It is clear, though, that qualitatively speaking as the magnitude of n increases, the number of entries, t, in the factor base should also increase. If one had no storage constraints, it would be possible to optimize the size of the factor base in order to minimize running time. In fact Wunderlich has analyzed, partly theoretically and partly empirically, the optimum size of the factor base as a function of the size of n, but the conclusion is that this optimum is so large that using a t of this size would result in an impractically large matrix even for the Cray X-MP; hence, we simply use as large a factor base as we can accommodate.

A detailed discussion of the coded implementation of quadratic sieves is inappropriate to the objectives of this paper, however, it is necessary to understand the essential steps involved in using sieves for factoring in order to appreciate why and how the Cray machines can be so well matched to the algorithm. For the Q(x) defined in (1), we wish to find solutions, x, to the related congruences

$$Q(x) \equiv 0 \pmod{p_i} \qquad (2)$$

for p_i an element in the factor base. As already remarked (2) has solutions precisely when $(n/p_i) = 1$, $p_i | n$ or $p_i = 2$. If $(n/p_i) = 1$ then (2) has two solutions, which are usually represented as A_{i1} and $B_{i1} = -(A_{i1} + 2m) \pmod{p_i}$. The sieving process depends on the fact that if we had a list of consecutive values of $Q(x)$ indexed by x, that for all indices $x = A_{i1} \pm hp_i$ and $x = B_{i1} \pm hp_i$ the associated $Q(x)$ would be divisable by p_i. The sieving procedure consists of dividing out (effectively) p_i from only these $Q(x)$ while leaving all other $Q(x)$ unaffected. This requires two sievings of the array per solution to (2) -- but as we shall see, the whole procedure can be implemented very efficiently.

As a matter of fact, we actually wish to solve a slightly more general version of (2)

$$Q(x) \equiv 0 \pmod{p_i^j} \qquad (3)$$

since the smaller primes may occur to some power > 1 in the factorization of $Q(x)$ over the factor base. We therefore choose a bound L and sieve for all $p_i^j \leq L$ where p_i is in the factor base. We have generally taken L to be the length of one sieving block (8×10^5 on the Cray 1S). This gives at least one successful division per prime power per sieving interval.

For each odd prime p_i in the factor base and each exponent j such that $p_i^j \leq L$ compute and save the two integers A_{ij} and B_{ij}, that are obvious generalizations of the A_{i1} and B_{i1} defined in connection with (2). A_{ij} is the least nonnegative residue $(\text{mod } p_i^j)$ that satisfies (3) and $B_{ij} = -(A_{ij} + 2m) \pmod{p_i^j}$ is its paired solution. These starting addresses for sieving are stored along with the associated weight $\log p_i$. In the same way that $Q(x)$ was sieved for p_i, we sieve at $x = A_{ij} \pm hp_i^j$ and $x = B_{ij} \pm hp_i^j$ by subtracting the weight $\log p_i$, $i \geq 1$. If ℓ is

the highest power of p_i that divides $Q(x)$ for some particular argument x and $p_i^{\ell} \leq L$, then $\log p_i$ will be subtracted from $Q(x)$ precisely ℓ times as it should be by this procedure.

The prime 2, of course, has every odd integer as a quadratic residue but $x^2 \equiv n \bmod 4$ has a solution if and only if $n \equiv 1 \bmod 4$. Also for $k \geq 3$, $x^2 \equiv n \bmod 2^k$ has solutions if and only if $n \equiv 1 \bmod 8$. Thus, the indices for sieving with powers of 2 must be chosen in a somewhat different fashion depending on the residue class of n mod 8. Following a suggestion of Pomerance, these sieving parameters are assigned as follows:

In all cases

$$A_{11} \equiv (1-m) \pmod 2$$

$$A_{12},\ B_{11},\ B_{12} \quad \text{undefined}$$

The other values of A_{1j}, B_{1j} must be treated as three distinct cases:

1) $n \equiv 1 \bmod 8$

For $j = 3,4,\ldots,\ell$, $\frac{B}{2} < 2^{\ell} \leq B$, A_{1j} is chosen such that

$$(A_{1j}-m)^2 \equiv n \pmod{2^{j+1}} \quad \text{and} \quad 0 \leq A_{1j} < 2^j$$

$$B_{1j} \equiv -(A_{1j} + 2m) \pmod{2^j} \ .$$

A_{11} is assigned weight 3 log 2. All other defined A_{1j}, B_{1j} have weight log 2.

2) $n \equiv 3 \bmod 4$

A_{11} is assigned weight log 2. Other A_{1j}, B_{1j} are undefined.

3) $n \equiv 5 \bmod 8$

A_{11} is assigned weight 2 log 2. Other A_{1j}, B_{1j} are undefined.

In sieving, start from the origin (x = 0) and sieve in both positive and negative directions until approximately t of the Q(x) have factored completely over the factor base. Because of some overhead in the initializing of arrays and the pipelining capability of the Cray's, we sieve on intervals which are as large as possible, say of length k, [0,k), [-k,0), [k,2k),... : k = 765,000 on the Cray 1S and 1.5×10^6 on the Cray X-MP.

In order to be able to carry out the factorization by subtraction, we need to first fill the arrays with approximate (single precision) values of $\log|Q(x)|$, $x \in [\ell k,(\ell+1)k)$. After the first positive and negative blocks, these logarithms are taken as constant in a given sieving interval. When an array has been initialized in this way, we sieve on p_I^j by subtracting the assigned weight (usually $\log p_I$ as discussed above) from each $\log Q(x)$ in the arithmetic progression of indices

$$x = A_{ij} \pm hp_I^j \qquad \text{and} \qquad x = B_{ij} \pm hp_I^j$$

and in the sieving interval $[k\ell, k(\ell+1))$.

When the sieving procedure is completed for a given block, the contents remaining in each location are compared with $\log p_t$: where p_t is the largest prime in the factor base. Values that are smaller then $\log p_t$ indicate the residues corresponding to these locations have been factored entirely into the primes in the factor base and these addresses are stored. Occasional false alarms due to approximations are eliminated later. Notice that very little multiple precision arithmetic is needed. The sieving procedure is repeated block-by-block until the desired number of Q's have been found that factored completely over the factor base. Once this has been done, it should be possible to find a subset of the Q's among which each of the primes has occurred an even number of times in total. The

problem of finding such a subset can best be treated as a problem of finding a linear dependency among vectors over GF(2). For each address x_i at which $Q(x_i)$ factored completely over the factor base as indicated by the entry being less than $\log p_t$, $Q(x_i)$ is now actually factored by dividing out the factors in the prime base to get

$$Q(x_i) = \prod_{j=0}^{t} p_j^{\alpha_{ij}}$$

with which we associate a binary, t+1 element, vector $V_i = (v_{ij})$ where $v_{ij} = 1$ if α_{ij} is odd and 0 otherwise. This results in a roughly $t \times t$ binary array, in which all of the essential information concerning the factorization of each $Q(x_i)$ is stored in t+1 bits, or $2^{-6}(t+1)$ words in the Cray. Now we need to find a subset, S, of the V_i such that $\bigoplus_S V_i = \phi$ where $\oplus$ denotes exclusive OR and ϕ denotes the zero vector. This is a straightforward problem in Gaussian elimination over GF(2). We use an improved version of a code developed by Parkinson and Wunderlich for this part of the calculation [16]. Once such a subset S is found, and w and z are calculated from

$$w \equiv \prod_S (x+m) \pmod{n}$$

where

$$Q(x) = (x+m)^2 - n \quad \text{and} \quad Q(x) \in S \ .$$

and

$$z \equiv \prod_{j=0}^{t} \prod_S p_j^{\frac{1}{2}\Sigma\alpha_{ij}} \pmod{n}$$

if $w \not\equiv \pm z$, n can be factored by calculating the greatest common divisors

$$(w+z, n) \quad \text{and} \quad (w-z, n) \ ,$$

both of which will be proper divisors of n. It may be the case that neither is a prime and that the procedure will have to be iterated to eventually obtain the prime decomposition of n, however the factor base remains the same and the factorizations of the Q(x) have already been done, so that only the end calculation would need to be repeated with some other linear dependent subset.

For Example

Since this has been -- unavoidably -- a rather lengthy discussion, we illustrate the basic quadratic sieve factoring algorithm using a small example.

Let $n = 37 \cdot 137 = 5069$, so that $m = \lfloor \sqrt{5069} \rfloor = 71$.

This example was "cooked" so as to have many small primes in the factor base, i.e., 5069 is a quadratic residue of 5,7,11 and 13, hence the factor base for t = 5 consists of -1,2,5,7,11,13.

First $5069 \equiv 5 \pmod 8$, so that $2^2 | Q(0 \pm 2h)$ and no other values of Q(x) are divisable by a power of 2 from the earlier discussion of A_{1j} and B_{1j}. Similarly, it is easy to show that for $p_2 = 5$, $A_{21} = 1$ and hence that $B_{21} = -(1 + 2m) \equiv -3 \equiv 2 \pmod 5$. Thus $5 | Q(1 \pm 5h)$ and $5 | Q(2 \pm 5h)$, etc. Similar results hold for p = 7, 11 and 13. Table II shows the

Table II

x	1	2	3	4	5	6	7	8	9
x+m	72	73	74	75	76	77	78	79	80
Q(x)	115	260	407	556	707	860	1015	1172	1331
Factors from base	5	$2^2 \cdot 5 \cdot 13$	11	2^2	7	$2^2 \cdot 5$	$5 \cdot 7$	2^2	11^3
Residual	23		37	139	101	43	29	293	

x	0	-1	-2	-3	-4	-5	-6	-7	-8
x+m	71	70	69	68	67	66	65	64	63
Q(x)	-28	-169	-308	-445	-580	-713	-844	-973	-1100
Factors from base	$-1 \cdot 2^2 \cdot 7$	$-1 \cdot 13^2$	$-1 \cdot 2^2 \cdot 7 \cdot 11$	$-1 \cdot 5$	$-1 \cdot 2^2 \cdot 5$	-1	$-1 \cdot 2^2$	$-1 \cdot 7$	$-1 \cdot 2^2 \cdot 5^2 \cdot 11$
Residual				89	29	713 $(23 \cdot 31)$	211	139	

shows the quadratic residues $Q(x)$ for $-8 \leq x \leq 9$, six of which factor completely over the factor base. The periodic appearance of the factors on which sieving is based is easy to see. The corresponding binary matrix **V** is of the form.

	-1	2	5	7	11	13
v_9					1	
v_2			1			1
v_0	1			1		
v_{-1}	1					
v_{-2}	1			1	1	
v_{-8}	1				1	

Three subsets of the v_i sum (exclusive OR) to ϕ:

$$v_0 \oplus v_{-1} \oplus v_{-2} \oplus v_{-8} = \phi \qquad \text{(a)}$$

$$v_9 \oplus v_{-1} \oplus v_{-8} = \phi \qquad \text{(b)}$$

$$v_9 \oplus v_0 \oplus v_{-2} = \phi \qquad \text{(c)}$$

The relationship in (a) corresponds to having presented the quadratic residue 3625 to the oracle:

$$\begin{aligned} 3625 &\equiv Q(0)Q(-1)Q(-2)Q(-8) \pmod{5069} \\ &\equiv (71)^2(70)^2(69)^2(63)^2 \pmod{5069} . \end{aligned}$$

In this case the algorithm (oracle) returns the result that

$$z \equiv 2^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \equiv 4557 \pmod{5069}$$

while we can calculate

$$w \equiv 71 \cdot 70 \cdot 69 \cdot 63 \equiv 512 \pmod{5069}$$

which tells us nothing whatsoever about the factorization of n, since $4557 \equiv -512 \pmod{5069}$.

If however we use either of the other two linear dependencies, we have: corresponding to b;

$$80 \cdot 70 \cdot 63 \not\equiv \pm 2 \cdot 5 \cdot 11^2 \cdot 13 \pmod{5069}$$

$$3039 \not\equiv \pm 523 \pmod{5069}$$

and hence

$$(3039 + 523, 5069) = 137$$

$$(3039 - 523, 5069) = 37 \ .$$

While corresponding to c;

$$80 \cdot 71 \cdot 69 \not\equiv \pm 2^2 \cdot 7 \cdot 11^2 \pmod{5069}$$

$$1067 \not\equiv \pm 3388 \pmod{5069}$$

and hence

$$(3388 + 1607, 5069) = 37$$

$$(3388 - 1607, 5069) = 137$$

either of which leads to the factorization of n.

It is this "plain" quadratic sieve factoring algorithm that was first implemented at Sandia, with the results documented in earlier portions of this paper. In attempting to use this technique to factor numbers larger than 57 to 58 digits, it was found that as the sieving interval became large enough to find $\approx$ t residues that factored completely that the magnitudes of the quadratic residues to be factored themselves became prohibitively large. Eventually the frequency with which a quadratic residue could be completely factored over the prime base became so small that the sieving times were intolerable. For the largest numbers factored, we were examining many tens of millions of residues to find even one complete factorization.

Quadratic Sieving: Fancy

If it is the case that after all of the prime factors from the prime base have been factored out of a quadratic residue,

$$Q(x) = q \prod_{i=0}^{t} p_i^{\alpha_i} \quad ,$$

the residual factor, q, is bounded by $p_t < q < p_t^2$, then q is necessarily a prime. Use of these "large" primes in the factorization by simply adding them to the prime base has been suggested and implemented [13], since if two quadratic residues, $Q(x_1)$ and $Q(x_2)$ can be found such that

$$Q(x_1) \equiv q \prod_{i=0}^{t} p_i^{\alpha_{1i}} \pmod{n} \quad ,$$

and

$$Q(x_2) \equiv q \prod_{i=0}^{t} p_i^{\alpha_{2i}} \pmod{n} \quad ,$$

then

$$Q(x_1)\cdot(x_2)q^{-2} \equiv \prod_{i=0}^{t} p_i^{\alpha_{1i}+\alpha_{2i}} \pmod{n} \quad ,$$

i.e., a quadratic residue that can be factored over the prime base can be constructed.

Although this approach, known as the large prime variation, does improve the performance of the algorithm over the "plain" quadratic sieve, the improvement isn't great enough to asymptotically make any difference. The reason that one only gets a marginal improvement from augmenting the prime base with a large prime is that for all intents and purposes we are randomly generating the quadratic residues -- at least so for as their divisability by a particular prime is concerned. Therefore the probability

that we will find another quadratic residue, Q(x'), — by sequentially indexing on x -- such that

$$q \mid Q(x')$$

is $\approx 1/q$ per trial, which is a very small quantity. If instead of simply searching for a Q(x') such that $q \mid Q(x')$, we could systematically generate a new sequence of Q's, such that $q \mid Q(x)$, and in which $Q(x)/q$ is small, then we could recover the same comparatively high probability that the resulting quotients would completely factor over the factor base that we had for Q(x) when x, and the Q(x), were small. This is the essential idea behind Davis' special q variation or "quadratic sieving; fancy."

Assume that we have found in the regular quadratic sieving an x for which

$$Q(x) = q \prod_{i=0}^{t} p_i^{\alpha_i}$$

where $p_t < q < p_t^2$.

A candidate for such an x is found when the quantity remaining in one of the indexed entries after the sieveing is completed lies between $\log p_t$ and $2 \log p_t$. If by chance some of these candidate Q(x)'s actually factor over the factor base because of large prime powers that were not considered in the original sieving, they are identified later and included among the complete factorizations. If q is actually prime, which is almost always the case, then note that:

$$Q(x \pm hq) = (x \pm hq + m)^2 - n = Q(x) \pm 2hq(x + m) + h^2q^2 \quad ,$$

where each term on the right is divisible by q, and the magnitude $Q(x + hq)/q$ is essentially that of Q(h), i.e., $\frac{Q(x \pm hq)}{q} \approx 2hm$ for $x << \sqrt{n}$. We can then form subsequences of residues starting at x and at $-(x + 2m) \pmod q$

whose magnitudes are comparable to those of Q(x) at the start of the original sieve. The sieving on the subsequences is done exactly as in the plain quadratic sieve. One problem that may be encountered when using special q's is that the arguments may become larger than single precision words in the computer. We overcame this by using the pairs (x,h) to represent x + hq and thus temporarily avoid multiprecision operations.

These special q's are relatively easy to find compared to finding complete factorizations. Thus in order to keep reduced residues "small", for each special q we sieve the subsequence for only a short interval: typically for a few blocks. When dealing with a single special q any complete factorization of a quadratic residue contains the factor q, which can be eliminated before going to the Gaussian reduction by combining pairs of factorizations to get quadratic residues in which q occurs an even number of times.

The sieving property is not dependent on the primality of the divisor, q, so why require the special q's to be prime? This is to prevent "collisions" between special q subsequences; that is, to prevent the same factorization being generated by two subsequences. If $q_1|Q(x)$ and $q_2|Q(x)$ and $p_t^2 > q_1 > q_2 > p_t$, then we would have $q_1 \cdot q_2|Q(x)$. But then Q(x) could not have passed the factorization criterion in the first place.

The special q modification introduced a few complications to the computation such as multiprecision arguments, and required writing a new computer code, but the increased capability was dramatic. The bottom line is that the special q variation enabled factorization of 63-64 digit integers in times comparable to those required by the original sieve to factor 55-56 digits. Furthermore, the relatively constant success rate for complete factorizations within the subsequences enables an accurate

early estimation to be made of how much computing time will be required for a given factorization.

Examining the residuals in Table II we see that 23, 29 and 31 are all candidate special q's. We let q = 23 be the special q in the example used earlier. Table III shows the resulting quadratic residues S_x based

Table III

Sieve on special q = 23

$$Q'(x) = Q(x_0 + xq) = (x_0 + xq + m)^2 - n$$

	S_x $x_0 = 1$					T_x $x_0 \equiv -(1-2m) \equiv -5 \pmod{23}$				
x	-2	-1	0	1	2	-2	-1	0	1	2
$x_0 + xq + m$	26	49	72	95	118	20	43	66	89	112
$Q'(x)/q$	-191	-116	5	172	385	-203	-140	-31	124	325
Factors from base	-1	$-1\cdot 2^2$	5	2^2	$5\cdot 7\cdot 11$	$-1\cdot 7$	$-1\cdot 2^2\cdot 5\cdot 7$	-1	2^2	$5^2\cdot 13$
Residual	191	29		43		29		31	31	

on $x_0 = 1$ and T_x based on the paired $x_0 \equiv -(1-2m) \equiv -5 \pmod{23}$ for $|x| < 2$. Four residues factor completely over the prime base -- extended by 23.

	-1	2	5	7	11	13	23
S_2			1	1	1		1
S_0			1				1
T_{-1}	1		1	1			1
T_2						1	1

Eliminating 23 by multiplying each row by the first we get;

	-1	2	5	7	11	13
S_2S_0				1	1	
S_2T_{-1}	1				1	
S_2T_2			1	1	1	1

Referring to Table II, we see that

$$V_0 \oplus V_{-8} \oplus S_2S_0 = \phi \qquad \text{(a)}$$

$$V_{-1} \oplus V_{-2} \oplus S_2S_0 = \phi \qquad \text{(b)}$$

$$V_{-1} \oplus V_0 \oplus V_9 \oplus S_2S_0 = \phi \quad . \qquad \text{(c)}$$

From (a) we find

$$w = 71 \cdot 63 \cdot 118 \cdot 72 \equiv 315 \pmod{5069}$$

and

$$z = 2^2 \cdot 5^2 \cdot 7 \cdot 11 \equiv 4754 \pmod{5069}$$

where

$$w \equiv -z \pmod{5069} \quad .$$

Similarly from (b) we find

$$w = 70 \cdot 69 \cdot 118 \cdot 72 \equiv 2125 \pmod{5069}$$

and

$$z = 2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 23 \equiv 2125 \pmod{5069}$$

neither of which tells us anything about the factorization of n. On the other hand, from (c)

$$w = 70 \cdot 71 \cdot 80 \cdot 118 \cdot 72 \equiv 2655 \pmod{5069}$$

and

$$z = 2 \cdot 5 \cdot 7 \cdot 11^2 \cdot 13 \cdot 23 \equiv 3099 \pmod{5069}$$

where $w \not\equiv \pm z$. Hence

$$(3099 + 2655, 5069) = 137$$

and

$$(3099 - 2655, 5069) = 37 \quad .$$

One cannot expect such a small example to illustrate the advantages of using special q's -- although the range of the parameters is slightly smaller in the example with the special q than without.

References

1. R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," Commun. ACM 21, 2 (Feb. 1978), 120-126.
2. H. Ong and C. P. Schnorr, "Signatures through Approximate Representations by Quadratic Forms," Proceedings of Crypto 83, Santa Barbara, CA, August 21-24, 1983, to be published by Plenum Press.
3. H. Ong, C. P. Schnorr and A. Shamir, "An Efficient Signature Scheme Based on Quadratic Equations," to appear Proceedings of 16th Symposium on Theory of Computing, Washington D.C., April 1984.
4. C. P. Schnorr, "A Cubic OSS-Signature Scheme," private communication, May 1984.
5. S. Even, O. Goldreich and A. Lempel, "A Randomized Protocol for Signing Contracts," in Advances in Cryptology, Proceedings of Crypto 82, Ed. by David Chaum, Ronald L. Rivest and Alan T. Sherman, Plenum Press, New York (1983).
6. G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel," Proceedings of Crypto 83, Santa Barbara, CA, August 21-24, 1983, to be published by Plenum Press.

7. P. D. Merillat, "Secure Stand-Alone Positive Personnel Identity Verification System (SSA-PPIV)," Sandia National Laboratories Tech. Rpt. SAND79-0070 (March 1979).

8. G. J. Simmons, "A System for Verifying User Identity and Authorization at the Point-of-Sale or Access," Cryptologia, Vol. 8, No. 1 (January, 1984), pp. 1-21.

9. G. J. Simmons, "Verification of Treaty Compliance -- Revisited," Proceedings of the 1982 Symposium on Security and Privacy, Oakland, CA (April 25-27, 1983), pp. 61-66.

10. J. A. Davis and D. B. Holdridge, "Factorization Using the Quadratic Sieve Algorithm," Sandia National Laboratories Tech. Rpt. SAND83-1346 (Dec. 1983).

11. J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman and S. S. Wagstaff, Jr., Factorizations of $b^n \pm 1$ b = 2, 3, 5, 6, 7, 10, 11, 12 up to High Powers," AMS Contemporary Mathematics, Vol. 22 (1983).

12. M. A. Morrison, J. Brillhart, "A Method of Factoring and the Factorization of F_7," Math. Comp. 29 (1975), 183-205.

13. C. Pomerance, "Analysis and Comparison of Some Integer Factoring Algorithms," in Number Theory and Computers, Ed. by H. W. Lenstra, Jr., and R. Tijdeman, Math. Centrum Tracts, No. 154, Part I, Amsterdam (1982), pp. 89-139.

14. R. H. Mendez, "The Japanese Supercomputer Challenge," SIAM News, Vol. 17, No. 1 (January 1984), pp. 1 and 5.

15. R. H. Mendez, "Benchmarks on Japanese and American Supercomputers -- Preliminary Results," IEEE Trans. Comp, Vol. C-33, No. 4 (April 1984). pp. 374-375.

16. D. Parkinson, M. C. Wunderlich, "A Memory Efficient Algorithm for Gaussian Elimination over GF(2) on Parallel Computers," private communication (Feb. 1983).