Computation of Discrete Logarithms in Prime Fields

B. A. LaMacchia *
A. M. Odlyzko

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

The presumed difficulty of computing discrete logarithms in finite fields is the basis of several popular public key cryptosystems. The secure identification option of the Sun Network File System, for example, uses discrete logarithms in a field $GF(p)$ with p a prime of 192 bits. This paper describes an implementation of a discrete logarithm algorithm which shows that primes of under 200 bits, such as that in the Sun system, are very insecure. Some enhancements to this system are suggested.

1. Introduction

If p is a prime and g and x integers, then computation of y such that

$$y \equiv g^x \bmod p, \quad 0 \leq y \leq p-1 \tag{1.1}$$

is referred to as *discrete exponentiation.* Using the successive squaring method, it is very fast (polynomial in the number of bits of $|p| + |g| + |x|$). On the other hand, the inverse problem, namely, given p, g, and y, to compute some x such that Equation 1.1 holds, which is referred to as the *discrete logarithm* problem, appears to be quite hard in general. Many of the most widely used public key cryptosystems are based on the assumption that discrete logarithms are indeed hard to compute, at least for carefully chosen primes.

The current state of knowledge about discrete logarithms is surveyed in [28, 34]. There are algorithms that compute discrete logarithms in $GF(p)$ in time roughly $q^{\frac{1}{2}}$,

*Present address: MIT, Cambridge, MA 02139

where q is the largest prime dividing $p-1$. Therefore, it is advisable to choose p such that $p-1$ is divisible by at least one large prime q, say $q \gtrsim 10^{30}$. Further, since solutions x to Equation 1.1 (for p, g, and y given) are determined only modulo the multiplicative order of g modulo p, it is advisable to choose g with its order divisible by a large prime. If these precautions are observed, then the best published algorithms [12] for computing discrete logarithms modulo a prime p have running time

$$\exp((1+o(1))(\log p)^{\frac{1}{2}}(\log\log p)^{\frac{1}{2}}) \quad \text{as } p \to \infty. \tag{1.2}$$

(Section 8 discusses a very recent discovery, the number field sieve, that is expected to be much faster asymptotically, but is not practical at this time.) The estimate given by Equation 1.2 is of roughly the same form as that of most of the fast practical algorithms for factoring composite integers of about the same size as p. An important feature of the estimate above is that it applies to a precomputation phase that has to be carried out once for each prime p. Once that phase is completed, individual discrete logarithms modulo that prime are much easier to compute.

Equation 1.2 is only an asymptotic estimate and does not say how large a prime p one could handle in practice in a reasonable amount of time. Currently large hard integers (hard here means that they are not of a special form and do not have small prime factors) with 100 to 110 decimal digits are factored in the equivalent of under a year on a 100 mips (million instructions per second) computer [26]. (The actual computation is spread over hundreds of workstations, and uses only their idle time.) The general feeling among experts in factoring integers and computing discrete logarithms was that the two problems were of roughly equal difficulty in practice as well as in theory. However, no computations of discrete logarithms in large prime fields have been reported before. Discrete logs have been computed in the field $GF(2^{127})$ by Coppersmith [10] and by Blake, Fuji-Hara, Mullin, and Vanstone [5]. This showed that the scheme implemented first in software by MITRE and then on a prototype chip by Hewlett-Packard was insecure. However, since in fields of characteristic two the discrete log problem is much easier than it is in prime fields (especially when one uses the Coppersmith algorithm [10, 28, 34]), it was not clear what impact this result had for the security of prime fields.

The complexity of computing discrete logs in prime fields is of substantial practical interest, since it provides an estimate of the size of the prime that has to be used. Sun Microcomputers, Inc., has implemented a secure identification feature as

part of their Network File System (NFS), and this system has been incorporated into UNIX®System V Release 4.0. The Sun scheme uses discrete exponentiation modulo a prime of 192 bits. This paper shows that it is quite easy to compute discrete logs modulo that prime, which makes it possible to break the NFS security in a very clean way. In a few days on a moderately fast machine it is possible to prepare a database of less than a megabyte that will then enable the cryptanalyst to obtain any particular user's secret key in a matter of at most a few minutes. While the preparation of the database does require extensive and sophisticated programming, once it is accomplished, the breaking of individual logs can be performed with hardly any programming effort at all by using a symbolic manipulation system such as Macsyma, Maple, or Mathematica. (If such a system is used, though, the time to break an individual key might be on the order of hours instead of minutes.) While the NFS security feature was already known to be weak, our results show that it can be broken in a very simple way.

The attack on the Sun cryptosystem provides the first experimental evidence concerning the difficulty of computing discrete logarithms in $GF(p)$. In order to better estimate the running time required to calculate discrete logs in larger prime field, most of the basic computations were also performed modulo a particular prime of 224 bits. These computations are mentioned briefly in Section 5.

The general conclusion to be drawn from the results reported here is that computing discrete logarithms modulo a prime is only a little harder than factoring integers of that same size. In particular, with about the same amount of effort that is used now to factor 110 decimal digit integers, one should be able to compute discrete logarithms modulo primes of about 100 digits.

Section 2 below presents the Sun NFS cryptosystem, and discusses some of its deficiencies. We mention in Section 3 some of the methods that could be used to strengthen it. The discrete log algorithm that was used is outlined in Section 4. Sections 5 through 7 describe various parts of the implementation. Section 8 sketches a new algorithm that was invented recently and is not currently practical, but might become so with additional improvements. Finally, Section 9 presents a summary of the results and some recommendations.

®UNIX is a registered trademark of AT&T.

2. The Sun NFS Cryptosystem

The Sun security option [39] in their NFS is built into the basic Sun Remote Procedure Call (RPC) and provides authentication of both users and machines using a combination of the Needham-Schroeder [33] protocol which uses DES, and a public key cryptosystem that is a modification of the Diffie-Hellman key exchange system [14]. It has long been known that the Sun system is not very secure. There are problems with the Needham-Schroeder protocol [13], which make it possible to defeat the timestamp system. Furthermore, the "yellow pages" that contain the public authentication information are not authenticated, so one can attack the security of the system by installing a bogus file. However, so far it appears that nobody has pointed out that the public key subsystem that is used by Sun is very weak. This fact makes it possible to impersonate any user with very little effort and leaving few traces.

In the Sun system, there is a prime p and an integer g that are the same for all users on all machines around the world that use this software. Each user or machine has a secret key m, and $g^m \bmod p$ is public. Authentication involves proving that one possesses the key m. For details see [39].

Both the paper [39] and the comments in the software refer to p as a 128-bit prime. Actually, though,

$$p = 521361942427152037168701411317018234177756360368035441677 9 \qquad (2.1)$$

and p has 192 bits. Both p and $\frac{p-1}{2}$ are primes. The integer $g = 3$, and the comments in the software refer to g as being a primitive root modulo p. However, g is a quadratic residue modulo p. (This has no effect on the security of the system.) Curiously enough, 2 is a primitive root modulo p. Probably the confusion is due to the fact that originally the system was built with a 128-bit prime for which 2 was not primitive. When the length of the prime was increased, the verification of primitivity was apparently forgotten.

To break the Sun system and impersonate a user with public key x, it is only necessary to find one of the two values of m, $0 < m < p - 2$, such that

$$g^m \equiv x \bmod p. \qquad (2.2)$$

This paper shows that this can be done very fast.

3. Alternative Methods

The basic idea of using a modification of the Diffie-Hellman scheme is not necessarily bad. What this paper shows is that the prime p has to be much larger for security. As

we mention in Section 9, primes of 512 or fewer bits should definitely be avoided (even though it is not now feasible to compute discrete logarithms modulo a prime of 512 bits). The basic difficulty with this approach is of course the added computational burden. The time to execute the Diffie-Hellman scheme in software is proportional to D^3, where D is the number of bits of p, so that going from 200 to 800 bits raises the complexity by a factor of 64.

There are several ways to deal with the problems of using a larger modulus. One is simply to use faster machines. Speeds of even small workstations are increasing very rapidly. Another, complementary approach, is to use faster algorithms. At the level of basic arithmetic, there are methods such as that of P. Montgomery [32] or methods using precomputed tables [2] that offer substantial speedups in modular multiplication over standard algorithms. One can also use addition chain methods [21] to carry out modular exponentiation with fewer modular multiplications than the usual method. (See [6] for some recent work on this.) The survey paper [8] discusses these and related methods.

Another approach is to use variants of the system that require fewer modular multiplications. One way to do this is to use secret keys m that are relatively short (say m of 150 bits). Another, somewhat similar, way to do this is to use the idea of Schnorr [38], and choose g to be not a primitive element, but a generator of a multiplicative subgroup modulo p that is of order 2^{150} or so.

An easy way to improve the security of the scheme is to have a different prime for each user. With the presently known algorithms, the precomputation phase of the discrete logarithm algorithm when applied to a 100 decimal digit prime might take a year on a 100-mips computer. If that prime is used by everyone on a network, individual secret keys could then be obtained in minutes. However, if every user had a different prime, obtaining any such key would require a year, which might be regarded as sufficiently secure.

Instead of using discrete exponentiation modulo a prime, one could possibly gain some speed by using addition on elliptic curves [22, 31]. Operations on elliptic curves are rather complicated, but no subexponential algorithms are known for the analog of the discrete logarithm problem on general elliptic curves, and so one could use much smaller primes. However, recently Menezes, Vanstone, and Okamoto [29] have shown that on some elliptic curves discrete logarithms can be computed in subexponential time, so one has to be on guard against further breakthroughs that might destroy the attractiveness of elliptic curve cryptosystems.

The Diffie-Hellman scheme is only one of many that can be used for authentication. In recent years, some very fast public key schemes have been developed. The Fiat-Shamir one [17] is perhaps the most widely known, but there are many other schemes [4, 7, 9, 19, 30, 38]. They are more than an order of magnitude faster than RSA and Diffie-Hellman methods (for the same size modulus), and so can be implemented in software without excessive computational requirements. However, while those systems are very suitable for smart card applications, for example, they are not always appropriate for use in computer networks, when one might have to cope with active attackers who might control the communication channel.

The fast signature schemes mentioned above belong to the class of identity-based systems, in which there is no need to maintain a database of authentication information for all users. In those systems there is a trusted key authentication center (KAC) which provides each user A with a secret key K_A that is derived from A's basic identification information (login name, or machine routing number in a network). Using the secure key K_A and the universally known public key of the KAC, user A can then produce a signature that can only come from A, and which will not enable anyone to generate signatures later. This simplifies the security problems, since it is not necessary to worry about monitoring a large secure authenticated file of information about users.

There are several identity-based cryptosystems that serve not only to authorize users, but also to generate session keys [3, 20, 23, 35, 36, 40]. They could be used very effectively in settings like that of the Sun NFS. Unfortunately, their running times are all comparable to those of the RSA and Diffie-Hellman systems. It is desirable to find a scheme of this kind that would be as fast as some of the signature schemes that are known, since that would alleviate the problem of having to use large moduli.

4. Discrete Logarithms in Prime Fields

This section describes the algorithm used to compute discrete logarithms modulo p, where p is the prime used in the Sun NFS cryptosystem. A more detailed description is contained in [12].

All of the fast algorithms known for computing discrete logarithms are forms of the index-calculus algorithm [34]. In the index-calculus algorithm, an initial processing stage computes the discrete logarithms of a set Q of elements in the field. Once this table of logarithms has been computed, any other logarithm may be found relatively quickly utilizing the information in the table.

The basic idea behind the precomputation is to obtain a number of equations in the logarithms of elements of Q, and solve the resulting system modulo the prime factors of $p-1$. There are many techniques for obtaining the equations. We will discuss three of them: the linear sieve, the residue list sieve, and Gaussian integers. They were all shown in [12] to have the asymptotic running time given by Equation 1.2.

The first equation generating method is the linear sieve. Let $H = \lfloor \sqrt{p} \rfloor + 1, J = H^2 - p$. Let Q be the *factor base*, consisting of the "small primes," -1, and the set of integers around H. We will search for pairs of integers (c_1, c_2) such that in

$$(H + c_1)(H + c_2) \equiv J + (c_1 + c_2)H + c_1 c_2 \bmod p$$

the right-hand side is *smooth* with respect to the factor base (i.e., all of the factors of $J + (c_1 + c_2)H + c_1 c_2$ are in the factor base). For any pair of integers (c_1, c_2) for which $J + (c_1 + c_2)H + c_1 c_2$ is smooth with respect to Q, we may write

$$(H + c_1)(H + c_2) \equiv q_{k_1}^{h_1} q_{k_2}^{h_2} \ldots q_{k_n}^{h_n} \mod p, \tag{4.1}$$

where each of the $q_{k_i} \in Q$ (the terms $(H + c_1)$ and $(H + c_2)$ are also assumed to be in Q). Now, taking logs on both sides, we obtain:

$$\log_g(H + c_1) + \log_g(H + c_2) \equiv \sum_i h_i \log_g q_{k_i} \mod (p-1), \tag{4.2}$$

which is an equation in the logs of elements of the factor base. Thus, by finding a system of such equations, we can solve the system modulo the factors of $p-1$ and obtain the logarithms of all the elements in the factor base Q.

We use a linear sieve to search for (c_1, c_2) pairs for which $(H + c_1)(H + c_2)$ is smooth with respect to the small primes in Q. Fix c_1, and let C be an array whose indexes are c_2 values we wish to consider. For each prime power q^h with $q \in Q$ and h sufficiently small, compute

$$d \equiv (J + c_1 H)(H + c_1)^{-1} \bmod q^h. \tag{4.3}$$

Notice that

$$(H + c_1)(H + d) \equiv 0 \bmod q^h \tag{4.4}$$

and that

$$(H + c_1)(H + c_2) \equiv 0 \bmod q^h \quad \forall c_2 \equiv d \bmod q^h. \tag{4.5}$$

Then we may step through the array C of possible c_2 values and add $\log q$ to all locations $C[c_2]$ for which c_2 satisfies Equation 4.5. The array stores the real logarithm

of the "smooth part" of the residue. If after all the q^h have been sieved the value stored at $C[c_2]$ is close to the real logarithm of the residue, then that c_2 value yields a Q-smooth residue. The (c_1, c_2) pair may then be used to derive an equation similar to Equation 4.1 above.

The linear sieve suffers from two main problem. First, the factor base Q is large, since it contains all the $H + c_j$ in addition to the small primes q_i, and thus we must collect many equations so that the resulting system can be solved. In fact, as pointed out in [24], we would really like to have many more equations than unknowns in order to simplify the task of solving the system. The second problem concerns the computation of d in Equation 4.3. In a test implementation of the linear sieve, we found that the time it took to invert $(H + c_1)$ and calculate d for each prime q^h was greater than the time it took to step through the entire array C! (Notice also that since d depends on c_1 and q^h, it is impossible to precompute d values.) These two problems make the linear sieve undesirable for large values of p. However, these deficiencies are not very serious, at least for $p \lesssim 10^{60}$, and the linear sieve would have been only slightly less efficient than the method that we did implement for attacking the Sun system. In general, the linear sieve has the advantage that it produces sparser equations than the scheme that we actually used.

The second method of equation generation mentioned in [12] is the residue list sieve. We did not even consider implementing it, since it has huge space requirements and is slow.

The third method of equation generation, and the one which was ultimately used, is the method of Gaussian integers [12], which was inspired by the work of ElGamal [16]. The idea here is to map the field $GF(p)$ to a subset of $\mathbf{Z}^2$. Let r be a small negative integer which is also a quadratic residue modulo p, let S be an integer such that $S^2 \equiv r \bmod p$, and let s represent the imaginary number $\sqrt{r}$. Choose two integers $T, V \lesssim \sqrt{p}$ such that $T^2 \equiv rV^2 \bmod p$. Let the factor base Q consist of small complex primes $x + ys$ in $Z[s]$, small real primes q (some of which will factor into two complex primes), and the integer V. Now, let $p' = T + Vs$, and choose $g = e + fs$ to be a complex prime which generates $(Z[s]/p')^*$. This g is the new base for logarithms.

In order to generate logarithm equations, we search for pairs of integers (c_1, c_2) such that their residue $c_1V - c_2T$ is smooth with respect to the real primes q in the factor base Q. Notice that we may write

$$c_1V - c_2T \quad = \quad V(c_1 + c_2s) - c_2(T + Vs)$$

$$\equiv \quad V(c_1 + c_2 s) \bmod p'.$$

If $c_1 V - c_2 T$ is smooth with respect to the real primes in Q, and $(c_1 + c_2 s)$ is smooth with respect to the complex primes in Q, we may write a related equation in logarithms to base g. When we map complex numbers $a + bs$ to $a + bS$ and the base $g = e + fs$ to $e + fS$, the logarithms are preserved.

We again use a sieve to find pairs of integers (c_1, c_2) whose residues $c_1 V - c_2 T$ are smooth over the small real primes in Q. For a fixed value of c_1, we allocate an array C corresponding to possible values of c_2. For each real prime power q^h, compute

$$d \equiv c_1 V T^{-1} \bmod q^h.$$

Every value $c_2 \equiv d \bmod q^h$ will yield a residue $c_1 V - c_2 T$ with a factor of q^h. If we increment the contents of $C[c_2]$ by the logarithm of q, we may inspect C after all primes have been sieved for c_2 values likely to yield Q-smooth residues $c_1 V - c_2 T$.

Notice that the Gaussian integer method avoids both of the major problems associated with the linear sieve. Our factor base is not unreasonably large, since it consists of only the small real and complex primes, plus the integer V. Also, while it is true that we need to calculate the value of $d \equiv c_1 V T^{-1} \bmod q^h$ for every value of c_1 and q^h, we can precompute the values of $V T^{-1} \bmod q^h$. Then on every sieve we only need multiply the stored value by c_1 modulo q^h. In addition, for our specific p, we were able to avoid performing any multiprecision arithmetic when calculating values of d by appropriately choosing Q and the range of possible c_2 values.

5. Sieving

Once our implementation of the Gaussian integer method was working, we began searching for (c_1, c_2) pairs of integers whose residues were smooth. All values of c_1 in the interval $[1, 95000]$ were considered. We chose to look at only positive values of c_1 for simplicity; negative values of c_1 are acceptable as far as the algorithm is concerned. Our sieve size (the set of considered c_2 values for each value of c_1) was 3×10^5. Again, only positive c_2 values were considered. These choices were definitely not optimal.

Since -1 is a quadratic nonresidue modulo p, while -2 is a residue, we chose $r = -2$. (This simplified the algorithm, since $Q(\sqrt{-2})$ has class number 1.) By factoring $x^2 + 2$ modulo p, we obtained a value of S such that $S^2 \equiv -2 \bmod p$. We computed the convergents to the continued fraction of S/p, and selected V and T as follows:

$$V \quad = \quad 484540679366944801179594827 23,$$

$$T = 22760185083691921160273336139.$$

These specific values of T and V were chosen from several candidate pairs (T_i, V_i), $T_i, V_i \lesssim \sqrt{p}$ because they satisfy the desirable property $T^2 + 2V^2 = p$.

For each fixed value of c_1, the sieve initialized an integer array C of size 3×10^5, representing the possible values of c_2. For each prime power q^h, value $d \equiv c_1 V T^{-1} \bmod q^h$ was computed. The contents of the array were incremented by $\lfloor 1000 \cdot \ln q \rfloor$ at every location $c_2 \equiv d \bmod q^h$. (The factor of 1000 and the floor function $\lfloor \cdot \rfloor$ are used so that we may maintain three decimal digits of accuracy yet perform integer, instead of floating point, calculations.) When the loop over prime powers q^h had finished, $C[c_2]$ contained the (approximate) real logarithm of the smooth part of the residue $c_1 V - c_2 T$.

After all of the small primes had been sieved for a fixed value of c_1, any c_2 for which $C[c_2]$ (i.e., the sum of the logarithms of its small prime factors) was close to the actual (real) logarithm of the residue $c_1 V - c_2 T$ was considered "interesting." If

$$\lfloor 1000 \cdot \ln(c_1 V - c_2 T) \rfloor - C[c_2] < 18420, \tag{5.1}$$

then the (c_1, c_2) pair was considered *partially-smooth* and assumed to have at most one "moderate-size" prime factor. (The magic number 18420 is $1000 \cdot \ln 10^8$. We considered any prime between 814,279 (the largest "small prime" in Q) and 10^8 to be of moderate size.) If

$$\lfloor 1000 \cdot \ln(c_1 V - c_2 T) \rfloor - C[c_2] < 10000, \tag{5.2}$$

the (c_1, c_2) pair was considered *fully-smooth* and assumed to have no prime factors outside of the factor base. (We need to use a large bound here as a result of an implementation decision concerning how real logarithms of $c_1 V - c_2 T$ residues are calculated.) Although these assumptions were heuristic, they narrowed down the list of candidate (c_1, c_2) pairs so that time was not wasted on factoring $c_1 V - c_2 T$ residues with multiple large prime factors.

Even though we had collected both partially- and fully-smooth pairs of (c_1, c_2) values, we ended up using only the fully-smooth pairs to generate equations. We found enough fully-smooth pairs in the search space to generate the needed equations; it was not necessary to resort to a "large prime variation" and allow equations with one or two large prime factors [27]. (The large prime variation would have increased the density of the set of equations which have to be solved, which was undesirable.)

On average, for each value of c_1 we found 17.27 values of c_2 in the interval $[1, 300000]$ such that (c_1, c_2) was fully-smooth. Overall, we found about 1.6×10^6 fully-smooth (c_1, c_2) pairs (out of 2.7×10^{10} checked values), or about one fully-smooth pair per 1.65×10^4 possible pairs.

For each fully-smooth pair (c_1, c_2), the corresponding equation:

$$c_1 V - c_2 T \equiv V(c_1 + c_2 s) \bmod p'$$

was factored (if possible) over the extended (complex) factor base. It was likely that $c_1 V - c_2 T$ was smooth with respect to the small real primes; it was not always the case, however, that the right-hand side of the equation could be factored over the small complex primes. Only one-third of the fully-smooth (c_1, c_2) pairs of integers actually yielded equations which could be completely factored over the complex factor base.

Once factored, the equations were sorted, and duplicate appearances of equations were removed. We found two causes of duplicate equations. The first cause was two distinct (c_1, c_2) pairs of integers which happened to have the same residue $c_1 V - c_2 T \bmod p$. The second (and much more frequent) cause was two pairs (c_1, c_2) and (c_1', c_2') with $c_1' = k \cdot c_1$, $c_2' = k \cdot c_2$, where k is a small integer. In this case,

$$c_1' V - c_2' T = V(c_1' + c_2' s) \implies k(c_1 V - c_2 T) = kV(c_1 + c_2 s)$$

and the factors of k cancel, leaving the same equation generated directly from the (c_1, c_2) pair. (Future implementations should probably only consider (c_1, c_2) pairs where c_1 and c_2 are relatively prime.) When all the duplicate equations were removed, we were left with a system of 288,017 distinct equations in 119,775 unknowns.

Before this system of equations was reduced using structured Gaussian elimination (see Section 6), it was first "squeezed" to eliminate dependent variables. Consider a real prime $q \in Q$ which may be factored into complex primes $(a + bs)$ and $(a - bs)$. All three of these variables could appear as unknowns in any of the 288,017 equations. Since it is sufficient to compute the discrete logarithm for any two of the three, we may reduce the density of the system by rewriting all appearances of one of the three variables in terms of the other two. For each triple of related primes $q_i, (a_i + b_i s), (a_i - b_i s)$, the prime which appeared the fewest number of times in the system of equations was rewritten in terms of the other two. In most cases we replaced $(a_i - b_i s)$ with $q_i(a_i + b_i s)^{-1}$; in the remaining instances $(a_i + b_i s)$ was replaced with $q_i(a_i - b_i s)^{-1}$. This process reduced the number of unknown variables appearing in the system to 96,321.

The sieving program was written in C and utilized the portable multiprecision arithmetic package distributed by A. Lenstra and M. Manasse with their multiple polynomial quadratic sieve program. The sieve was run on a Silicon Graphics 4D-220 computer. It has 4 R3000 MIPS Computers, Inc., 25 MHz processors, each rated at about 18 mips, or 18 times the speed of a DEC VAX 11/780 computer (and about 1.3 times the speed of a DECstation 3100). This machine has 128 Mbytes of main memory. Since our program occupied about 20 Mbytes of memory, all four processors could sieve simultaneously over different c_1 intervals.

It took our implementation approximately one hour of CPU time (running on a single processor) to successfully sieve 1000 possible c_1 values, for a total running time of about 100 hours. Factoring the equations derived from fully-smooth (c_1, c_2) pairs took another 104 minutes, on average, per 1000 possible c_1 values. Neither of these running times is particularly impressive; performance could probably be improved significantly without too much effort. However, it should be noted that even our implementation was able to collect the necessary equations using only a few days worth of CPU time.

Aside from improving the efficiency of the algorithm, one could achieve about an order of magnitude improvement by working with a smaller factor base and using the single and double large prime variations. The reason we did not do this is that we wished to extrapolate our result to estimate the security of much larger systems. The efficiency of sieving is very easy to estimate, as one can run experiments on small ranges and also use the data from the large factoring experiments. On the other hand, this kind of extrapolation is harder to do in the case of solving linear equations, and so we decided to generate a very large system that we could experiment with.

In order to gain more information concerning the performance of the Gaussian integer scheme, experiments were also performed on a 224-bit prime q using essentially the same algorithm. This prime was chosen to be similar to the one used in the Sun system in that $\frac{q-1}{2}$ is a prime and $q = T^2 + 2V^2$ for particular integers $T, V \lesssim \sqrt{q}$. The factor base was the same as that used in the case of the 192-bit prime. Sieving occurred over a substantially larger range; all values of c_1 in the interval $[-3 \times 10^5, 9 \times 10^5]$ were considered. Our sieve size (the range of possible c_2 values) was kept at 3×10^5. Sieving took under 1200 hours of CPU time, and produced about 1.1×10^6 candidate fully-smooth (c_1, c_2) pairs. After removing those pairs for which c_1 and c_2 were not relatively prime, factoring both sides of the corresponding $c_1 V - c_2 T \equiv V(c_1 + c_2 s)$ equations, and "squeezing" the resulting system, we were

left with a system of 164,841 equations in 94,398 unknowns.

6. Linear Algebra

The problem of solving a large system of linear equations over a finite field has always loomed as a potential bottleneck in factoring integers and computing discrete logarithms. In fact, the major goal of this project was to explore the extent to which large systems could be solved efficiently. That was also the motivation for working with a much larger factor base than was necessary to break the Sun cryptosystem, and for obtaining many more equations than were needed.

The various algorithms that are available for solving large sparse linear systems over finite fields are surveyed in [24]. Here we will only mention briefly how they performed on our problem. The system of 288,017 equations in 96,321 unknowns was reduced by the structured Gaussian elimination method to a smaller system of 7,262 equations in 6,006 unknowns. This operation took a couple of hours on the SGI computer. The resulting smaller system was then solved modulo 2 in 1.9 hours using a conjugate gradient program, and modulo $\frac{p-1}{2}$ in about 44 hours using the Lanczos algorithm. (There was no real need to solve the system modulo 2, since variables are 0 or 1 depending on whether the corresponding prime is a quadratic residue modulo p or not. This exercise was undertaken in order to estimate the efficiency of the linear algebra algorithms in different settings.) With more careful programming, that time could be decreased substantially.

One disadvantage of the Gaussian integer scheme that we implemented is that it produces relatively dense equations. Even without using the large prime variation, it gives equations with density comparable to that of the single large prime variation of the quadratic sieve. The linear sieve does not have this disadvantage.

The general conclusion that can be derived from the results of [24] is that linear algebra is likely to be a significant but not an insurmountable problem in computing discrete logarithms modulo large primes. The major difficulty is that, while significant computing resources may be applied to sieving, such power is not available for solving the linear algebra. Sieving may be performed efficiently in a distributed environment, and, as A. Lenstra and M. Manasse have demonstrated, huge amounts of computing power may be obtained from distributed systems of workstations. Solving large systems of linear equations, though, appears to require either a single fast processor or else a closely coupled set of processors. One cannot hope to devote nearly as much computing power to this portion of the algorithm. However, as is explained

in [24], one can shift the load onto the sieving part by methods such as obtaining many excess equations or not using techniques that produce dense equations. These methods do decrease the efficiency of the sieving algorithm, but they make the linear algebra easier. In any case, the systems that are likely to arise in the near future appear to be tractable using known techniques.

7. Computing Individual Discrete Logarithms

The challenge, provided by M. Shannon of Sun Microsystems, was to find m such that:

$$3^m \equiv z \bmod p \tag{7.1}$$

where

$$z = 308899365792522917304711040535452115103232581944049898356 5 \tag{7.2}$$

Since z is a square of a primitive root modulo p, there are two solutions for m in the range $0 \leq m \leq p-2$. These two solutions are

$$m = 871373321106180104114279941663066214856051438571369779697 \tag{7.3}$$

and $m + \frac{p-1}{2}$. Since the main computation obtained discrete logarithms of small primes to base $g = 1 + s$, the log of z to base g was computed, and the value of m in Equation 7.3 was found by simple base conversion.

To obtain the log of z to the base g, a modification of the strategy outlined in [12] was employed. To express z in terms of what [12] called "medium-sized primes," the numbers $z \cdot g^k$ (reduced modulo p) for $k = 0, 1, \ldots$ were each expressed as

$$z \cdot g^k \equiv \frac{x}{y} \bmod p \tag{7.4}$$

with x and y close to $\sqrt{p}$. (For each k, several (x, y) pairs were tried.) The values of x and y were then factored, and for those x which split into primes $< 10^{10}$, the corresponding values of y were factored, until a value of y was found such that all of its prime factors were $< 10^{10}$. This occurred for $k = 73$, and the factorizations were:

$$x = -19 \cdot 41 \cdot 127 \cdot 9257 \cdot 62473 \cdot 653693 \cdot r_1, \tag{7.5}$$

$$y = 2 \cdot 89 \cdot 785737 \cdot r_2 \cdot r_3 \cdot r_4, \tag{7.6}$$

where

$$r_1 = 20261929, \tag{7.7}$$

$$r_2 = 435600073, \tag{7.8}$$

$$r_3 = 124325687, \tag{7.9}$$

$$r_4 = 832003. \tag{7.10}$$

Except for the r_i, all the other primes appearing in the factorizations of x and y were already in the factor base, so it was only necessary to find the logs of the r_i.

If we let r denote one of the r_i, its logarithm was computed by searching for pairs of relatively small integers c and d such that

$$cV - dT \equiv 0 \bmod r. \tag{7.11}$$

Candidate values of (c, d) were obtained by finding two pairs (c_1, d_1) and (c_2, d_2) that were quite small and satisfied Equation 7.11, and then considering linear combinations

$$(c, d) = \alpha(c_1, d_1) + \beta(c_2, d_2)$$

for small integers α and β. The pairs (c_1, d_1) and (c_2, d_2) were obtained from the continued fraction of σ/r, where $\sigma \equiv T/V \bmod r$. For example, for $r = r_1$, $(-6254, 1691)$ and $(3683, 2244)$ were used. Each pair (c, d) was tested to see whether $c + ds$ split into quadratic integers of small norm. Since the norm of $c + ds$ is $c^2 + 2d^2$ and so is small, this happened a large fraction of the time. When it did happen, $(cV - dT)/r$ was tested to see whether it factored into small primes. When this succeeded, the search was over, and the log of r could be obtained from the database. For example, for $r = r_1$, $(c, d) = (-15355, 78668)$ was a good pair, and gave the relation

$$r_1 \equiv -\frac{V(1+S)(3-S)(15+23S)(545+21S)}{59 \cdot 2699 \cdot 64781 \cdot 186581 \cdot 253787 \cdot 256079} \bmod p. \tag{7.12}$$

Since the logs of all the factors on the right side of Equation 7.12 were in the factor base, this expression gave the log of r_1.

The entire computation took several hours on a DEC VAX 8550 minicomputer. However, with appropriately written programs it should be possible to carry out similar computations in several minutes. The main reason for the long time that was required was that testing whether an integer was smooth was done using a general purpose factoring package that was available on the VAX. This package does some trial division, and then applies the multiple-polynomial quadratic sieve. It is only moderately efficient, and is meant to obtain complete factorizations. In our approach to computing individual logs, it is only important to check for smoothness with respect

to some bound, and it is only important to find some smooth numbers. Thus an algorithm utilizing some combination of trial division, the Pollard ρ-method, the elliptic curve method, and an early abort strategy (well known in integer factorization) ought to be many times faster. Furthermore, it is possible to optimize various choices that were made rather arbitrarily in our implementation. Since even with all of the inefficiencies in our program, the running time was expected to be (and was) quite moderate, the optimization task was not undertaken.

8. The Number Field Sieve

The number field sieve is a new factoring algorithm with a much lower asymptotic running time estimate than previous algorithms. It was first proposed by Pollard [37] for factoring integers near perfect cubes, with the basic idea derived from the early work of ElGamal [16]. A similar idea was proposed independently by Elkies (electronic mail communication of Feb. 13, 1989) a short time later. The Pollard approach was then extended by A. Lenstra, H. Lenstra, and M. Manasse [25] to factor so-called Cunningham integers, that is integers n of the form

$$n = a^k \pm 1 \tag{8.1}$$

where a is small and k is large. If we let

$$L_n(v, r) = \exp((r + o(1))(\log n)^v (\log\log n)^{1-v}), \tag{8.2}$$

then the number field sieve factors Cunningham integers n in time

$$L_n(1/3, 2(2/3)^{2/3}) = L_n(1/3, 1.526\ldots). \tag{8.3}$$

(This estimate is based on several assumptions that seem reasonable but have not been proven.) This algorithm is fast not only asymptotically, but also in practice, although it is quite complicated to implement. A. Lenstra and M. Manasse have recently used it to factor $F_9 = 2^{512} + 1$ in about the same amount of time that the quadratic sieve requires to factor integers of about 110 decimal digits. (F_9 has 155 decimal digits, and one prime factor of 7 decimal digits was already known. However, this did not help, since the number field sieve for Cunningham integers cannot take advantage of such auxiliary information. Thus it is a debatable point whether A. Lenstra and M. Manasse factored a 155 or a 148 decimal digit integer.)

The number field sieve can also be extended to factor general integers. At first, the only technique that could be shown to work in general was due to Buhler, H.

Lenstra, and Pomerance, and it yields a running time estimate of

$$L_n(1/3, 3^{2/3}) = L_n(1/3, 2.080\ldots). \tag{8.4}$$

Recently H. Lenstra and Adleman [1] have proposed two independent techniques which lower the running time estimate to

$$L_n(1/3, 4 \cdot 3^{-2/3}) = L_n(1/3, 1.922\ldots). \tag{8.5}$$

Even more recently, Coppersmith [11] has proposed a modification that achieves a running time of

$$L_n(1/3, 2^{1/3} \cdot 3^{-1} \cdot (46 + 13\sqrt{13})^{1/3}) = L_n(1/3, 1.901\ldots). \tag{8.6}$$

Although asymptotically this is still far better than other algorithms, the point at which this method would be faster than algorithms such as the quadratic sieve appears to be in the vicinity of 200 decimal digits. On the other hand, the number field sieve is a very recent invention, and so it is likely that substantial improvements might occur which would make it practical.

The number field sieve can be extended to computing discrete logarithms, as is shown in [18].

9. Conclusions and recommendations

The basic conclusion to be drawn from this paper is that computing discrete logs in prime fields $GF(p)$ with the Gaussian integer method is not much harder than factoring composite integers n with $n \approx p$ with the multiple polynomial quadratic sieve. Right now roughly 110 decimal digit integers are being factored in several months of elapsed time using idle time on hundreds of workstations all over the world. Since workstations are becoming dramatically faster and more numerous, and much more widely connected to networks, it seems prudent to allow for at least a 100-fold increase in the available computing power over the next few years, which would allow factoring of integers in the 130-140 decimal digit range. Furthermore, since many discrete log cryptographic schemes have the feature that they use a fixed prime which cannot easily be changed, one has to allow for attacks that consume not just a couple of months, but even a couple of years of computing time. Therefore even 512-bit primes appear to offer only marginal security, even against the Gaussian integer scheme.

10. Acknowledgements

The authors wish to thank N. Fernandez, K. McCurley, and M. Merritt for providing helpful information and stimulating this work.

References

[1] L. M. Adleman, Factoring numbers using singular integers, to be published.

[2] P. Barrett, Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor, *Advances in Cryptology: Proceedings of Crypto '86*, A. M. Odlyzko, ed., *Lecture Notes in Computer Science* **263**, Springer-Verlag, NY (1987), 311-323.

[3] F. Bauspiess and H.-J. Knobloch, How to keep authenticity alive in a computer network, *Advances in Cryptology: Proceedings of Eurocrypt '89*, J.-J. Quisquater, ed., to appear.

[4] T. Beth, Efficient zero-knowledge identification scheme for smart cards, *Advances in Cryptology: Proceedings of Eurocrypt '88*, C. G. Günther, ed., *Lecture Notes in Computer Science* **330**, Springer-Verlag, NY (1988), 77-84.

[5] I. F. Blake, R. Fuji-Hara, R. C. Mullin, and S. A. Vanstone, Computing logarithms in fields of characteristic two, *SIAM J. Alg. Disc. Methods* **5** (1984), 276-285.

[6] J. Bos and M. Coster, Addition chain heuristics, *Advances in Cryptology; Proceedings of Crypto '89*, G. Brassard, ed., *Lecture Notes in Computer Science* **435**, Springer-Verlag, NY (1990), 400-407.

[7] J. Brandt, I. Damgard, P. Landrock, and T. Pedersen, Zero-knowledge authentication scheme with secrete key exchange, *Advances in Cryptology: Proceedings of Crypto '88*, S. Goldwasser, ed., *Lecture Notes in Computer Science* **403**, Springer-Verlag (1989), 583-588.

[8] E. F. Brickell, A survey of hardware implementations of RSA (Abstract), *Advances in Cryptology; Proceedings of Crypto '89*, G. Brassard, ed., *Lecture Notes in Computer Science* **435**, Springer-Verlag, NY (1990), 368-370.

[9] E. F. Brickell and K. S. McCurley, An interactive identification scheme based on discrete logarithms and factoring, *Advances in Cryptology: Proceedings of Eurocrypt '90*, I. Damgard, ed., to be published.

[10] D. Coppersmith, Fast evaluation of discrete logarithms in fields of characteristic two, *IEEE Transactions on Information Theory* **30** (1984), 587-594.

[11] D. Coppersmith, Modifications to the number field sieve, to be published.

[12] D. Coppersmith, A. Odlyzko, and R. Schroeppel, Discrete logarithms in $GF(p)$, *Algorithmica* **1** (1986), 1-15.

[13] D. Denning, and G. Sacco, Timestamps in Key Distribution Protocols, *Communications of the ACM* **24** (1981) 533-536.

[14] W. Diffie and M. Hellman, New Directions in Cryptography, *IEEE Transactions on Information Theory* **22** (1976), 472-492.

[15] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory* **31** (1985), 469-472.

[16] T. ElGamal, A subexponential-time algorithm for computing discrete logarithms over $GF(p^2)$, *IEEE Transactions on Information Theory* **31** (1985), 473-481.

[17] A. Fiat and A. Shamir, How to prove yourself: practical solution to identification and signature problems, *Advances in Cryptology: Proceedings of Crypto '86*, A. M. Odlyzko, ed., *Lecture Notes in Computer Science* **263**, Springer-Verlag, NY (1987), 186-199.

[18] D. M. Gordon, Discrete logarithms in $GF(p)$ using the number field sieve, to be published.

[19] L. C. Guillou and J.-J. Quisquater, A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory, *Advances in Cryptology: Proceedings of Eurocrypt '87*, D. Chaum, ed., *Lecture Notes in Computer Science* **304**, Springer-Verlag, NY (1988), 127-141.

[20] C. G. Günther, Diffie-Hellman and ElGamal protocols with one single authentication key, *Advances in Cryptology: Proceedings of Eurocrypt '89*, J.-J. Quisquater, ed., to appear.

[21] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed., Addison-Wesley 1981.

[22] N. Koblitz, Elliptic curve cryptosystems, *Math. Comp.* **48** (1987), 203-209.

[23] K. Koyama and K. Ohta, Identity-based conference key distribution systems, *Advances in Cryptology: Proceedings of Crypto '87*, C. Pomerance, ed., *Lecture Notes in Computer Science*, **293**, Springer-Verlag, NY (1988), 175-194.

[24] B. A. LaMacchia and A. M. Odlyzko, Solving large sparse linear systems over finite fields, *Advances in Cryptology: Proceedings of Crypto '90*, A. Menezes, S. Vanstone, eds., to be published.

[25] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, The number field sieve, *Proc.* 22^{nd} *ACM Symp. Theory of Computing* (1990), 564-572.

[26] A. K. Lenstra and M. S. Manasse, Factoring by electronic mail, *Advances in Cryptology: Proceedings of Eurocrypt '89*, J.-J. Quisquater, ed., to be published.

[27] A. K. Lenstra and M. S. Manasse, Factoring with two large primes, *Advances in Cryptology: Proceedings of Eurocrypt '90*, I. Damgard, ed., to be published.

[28] K. S. McCurley, The discrete logarithm problem, in *Cryptography and Computational Number Theory*, C. Pomerance, ed., *Proc. Symp. Appl. Math.*, Amer. Math. Soc., 1990, to appear.

[29] A. Menezes, S. Vanstone, T. Okamoto, Reducing elliptic curve logarithms to logarithms in a finite field, to be published.

[30] S. Micali and A. Shamir, An improvement of the Fiat-Shamir identification and signature scheme, *Advances in Cryptology: Proceedings of Crypto '88*, S. Goldwasser, ed., *Lecture Notes in Computer Science* **403**, Springer-Verlag, NY (1989), 244-247.

[31] V. Miller, Use of elliptic curves in cryptography, *Advances in Cryptology: Proceedings of Crypto '85*, H. C. Williams, ed., *Lecture Notes in Computer Science* **218**, Springer-Verlag, NY (1986), 417-426.

[32] P. L. Montgomery, Modular multiplication without trial division, *Math. Comp.* **44** (1985), 519-521.

[33] R. Needham and M. Schroeder, Using encryption for authentication in large networks of computers, *Comm. ACM* **21** (1978), 993-999.

[34] A. M. Odlyzko, Discrete logarithms in finite fields and their cryptographic significance, *Advances in Cryptology: Proceedings of Eurocrypt '84*, T. Beth, N. Cot, I. Ingemarsson, eds., *Lecture Notes in Computer Science* **209**, Springer-Verlag, NY (1985), 224-314.

[35] E. Okamoto, Key distribution systems based on identification information, *Advances in Cryptology: Proceedings of Crypto '87*, C. Pomerance, ed., *Lecture Notes in Computer Science* **293**, Springer-Verlag, NY (1988), 194-202.

[36] E. Okamoto and K. Tanaka, Key distribution system based on identification information, *IEEE J. Selected Areas Commun.* **SAC-7** (1989), 481-485.

[37] J. M. Pollard, Factoring with cubic integers (parts I and II), unpublished manuscripts, August 1988 and December 1988.

[38] C. P. Schnorr, Efficient identification and signatures for smart cards, *Advances in Cryptology: Proceedings of Crypto '89*, G. Brassard, ed., *Lecture Notes in Computer Science* **435**, Springer-Verlag, NY (1990), 239-251.

[39] B. Taylor and D. Goldberg, Secure networking in the Sun environment, *Proc. USENIX Assoc. Summer Conference*, Atlanta 1986, 28-37.

[40] Shigeo Tsujii and Toshiya Itoh, An ID-based cryptosystem based on the discrete logarithm problem, *IEEE Journal on Selected Areas in Communications* **8** (1989), 467-473.