Fast Evaluation of Logarithms in Fields of Characteristic Two

DON COPPERSMITH

Abstract—**A method for determining logarithms in GF(2^n) is presented. Its asymptotic running time is $O(\exp(cn^{1/3}\log^{2/3} n))$ for a small constant c, while, by comparison, Adleman's scheme runs in time $O(\exp(c'n^{1/2}\log^{1/2} n))$. The ideas give a dramatic improvement even for moderate-sized fields such as GF(2^{127}), and make (barely) possible computations in fields of size around 2^{400}. The method is not applicable to GF(q) for a large prime q.**

I. Introduction

We are interested in computations in a finite field, specifically $F = \mathrm{GF}(2^n)$. This field is constructed as follows. Select a particular irreducible polynomial $P(x)$, of degree n, over GF(2). An element C of F is a polynomial $C(x)$ over GF(2), considered mod $P(x)$. There are 2^n elements. Addition and multiplication are defined as addition and multiplication of polynomials over GF(2), mod $P(x)$.

It is customary to select $P(x)$ to be *primitive*. This means that, as m ranges through the integers $0, 1, \cdots, 2^n - 2$, the field elements $A(x) \equiv x^m \bmod P(x)$ take on the value of each nonzero field element in F exactly once. Conversely, to each nonzero field element $A(x)$ in F, we associate the integer m, and say that m is the *logarithm* of $A(x)$. Since $x^{2^n-1} \equiv 1 \bmod P(x)$, the logarithm is only defined mod $2^n - 1$. Thus we consider the logarithm m to lie in the ring $Z/(2^n - 1)$, i.e., the integers mod $2^n - 1$. We refer to the calculation of m as the *discrete logarithm problem*. If the field is small enough, one can tabulate all the field elements and their logarithms, and use this table for computation within the field, much as one uses a table of natural logarithms for calculations involving real numbers.

For large fields, such as GF(2^{127}), it is infeasible to tabulate the logarithms. Further, it is relatively difficult to extract logarithms in a large field, while it is relatively easy to exponentiate. This disparity led Diffie and Hellman [7] to propose a cryptographic scheme based on exponentiation in a finite field. Their proposal involved the finite field GF(q) for a large prime q, although their scheme has since been adapted to GF(2^n), for ease of implementation. The point of this paper is that this *adaptation* was ill-advised; we show that logarithms are relatively easy to compute in GF(2^n). The techniques of this paper are inapplicable to GF(q) for large primes q. Thus, as far as we know today, GF(q) is more secure than GF(2^n), i.e., for a given level of security, GF(2^n) requires a far larger word size than GF(q).

The Diffie–Hellman scheme establishes a random secret key, for subsequent use as the key to a conventional cryptographic system, via communication over a public network. It works as follows: users C and D wish to establish a secret key. User C selects a random integer c, computes x^c in the field, and sends x^c to user D. Meanwhile user D selects a random integer d, and sends x^d to user C. Now user C can compute $(x^d)^c = x^{cd}$, while user D can compute $(x^c)^d = x^{cd}$. Both users know the same random quantity x^{cd}, and can use it as a cryptographic key for subsequent communication in a conventional cryptographic system. A wiretapper can obtain the key if he can find logarithms in F. Namely, if he calculates $\log(x^c) = c$, he can easily compute $(x^d)^c = x^{cd}$. Similar schemes exist for exchange of arbitrary (nonrandom) information.

This cryptographic application has sparked renewed interest in the discrete logarithm problem. An early algorithm published by Pohlig and Hellman [15] and also attributed to Roland Silver, works in time about $\sqrt{p}$, where p is the largest prime dividing $q - 1$, and where q is the size of the field. (In the present case, $q = 2^n$ is often chosen such that $q - 1$ is a prime, so that the running time is about $2^{n/2}$.) In this algorithm, to obtain $\log A(x)$ one would tabulate i and $x^{i(q-1)/p}$ for all i between 1 and $k = [\sqrt{p}]$, evaluate $A(x)^{(q-1)/p} x^{jk(q-1)/p}$ for all j between 0 and $[n/k]$, and compare to the table to find an instance of equality: $(A(x)x^{jk-i})^{(q-1)/p} = 1$. This yields the desired result mod p: $\log A(x) = i - jk \bmod p$. The Chinese remainder theorem takes care of $q - 1$ being the product of distinct prime factors p_i, and repeated prime factors pose no extra difficulty.

Adleman [1] found the first subexponential algorithm for the problem. Its running time is $\exp(c\sqrt{\log q \log\log q})$ for a field of size q, where c is a small constant. As proposed, his algorithm only deals with the case where q is a prime. Hellman and Reyneri [8] adapted the algorithm to work in GF(2^n), where the running time becomes $\exp(c'\sqrt{n \log n})$. This algorithm will be described in Section II. Like the Morrison–Brillhart algorithm for factoring integers ([12] and [10]), its success depends on the probability of a random integer being "smooth" (expressible as the product of small primes), or on the probability of a random poly-

Manuscript received November 2, 1983; revised December 20, 1983.

The author is with IBM Research, P.O. Box 218, Yorktown Heights, NY 10598.

0018-9448/84/0700-0587$01.00

nomial over GF(2) being expressible as the product of small irreducible polynomials. Its limitation lies in the fact that, the larger the degree of a polynomial, the less likely it is to be smooth.

The present algorithm has an asymptotic running time of $O(\exp(cn^{1/3}\log^{2/3} n))$. It relies on the fact that squaring is a linear operation in fields of characteristic two; that is, $(A+B)^2 = A^2 + B^2$. With this fact, we are able to produce polynomials of moderate degree, which we then require to be smooth. Since our polynomials are smaller than those involved in Adleman's algorithm, they have a better probability of being smooth. This fact allows the decrease in running time. The idea for our algorithm stems from the concept of "systematic equations" due to Blake, Fuji-Hara, Mullin, and Vanstone [4].

Throughout this paper we will use for our example the field $GF(2^{127})$. The primitive polynomial involved is $P(x) = x^{127} + x + 1$. The Diffie–Hellman key exchange algorithm, as described above, has been implemented in this field. To build the database necessary to take logarithms in this field, Adleman's algorithm seems to take two weeks; a modification due to Blake, Fuji-Hara, Mullin, and Vanstone [4] takes about nine hours, and the present scheme takes eleven minutes. (These estimated timings for an IBM 3081K assume 250 microseconds for a "smoothness test.")

The organization of the rest of the paper is as follows. In Section II, we describe Adleman's algorithm. In Section III, we describe the improvement due to Blake *et al.* Section IV details Blake, Fuji-Hara, Mullin, and Vanstone's concept of "systematic equations." Section V introduces the major idea in the present algorithm, and shows its applicability to fields of moderate size, such as $GF(2^{127})$. Sections VI and VII detail two parts of the present algorithm. Section VIII gives the full algorithm, as used in very large fields, and begins examination of the asymptotic running time of the algorithm; this examination is continued in Section IX. Section X gives specific examples, and Section XI gives some programming considerations based on our experience with $GF(2^{127})$.

II. Adleman's Algorithm

Adleman's algorithm [1] for the discrete logarithm is based on the ideas used by Morrison and Brillhart [12] for factoring large integers. For integers A and B, the statement *A is smooth with respect to B* is defined to mean that all of the prime factors of A are less than B. Adleman's algorithm, and the Morrison–Brillhart algorithm, depend on the fact that smooth integers are relatively common. Although Adleman's algorithm was described for $GF(q)$ for q a prime, it applies equally well for $GF(2^n)$, as noted by Hellman and Reyneri [8]. We describe the algorithm for this case.

Select a bound $b \sim c\sqrt{n\log n}$, where c is a small constant, perhaps $c = 1$. Say that $A(x)$ *is smooth with respect to b* when $A(x)$ is the product of irreducible polynomials of degree at most b. Select a random integer m between 0 and $2^n - 2$. Set $A(x) \equiv x^m \bmod P(x)$. Then $A(x)$ is a random nonzero polynomial of degree at most $n-1$. Test $A(x)$ for smoothness with respect to b. If it is smooth, factor $A(x)$ explicitly as $A(x) \equiv x^m \equiv \prod_j q_j(x)^{e_j} \bmod P(x)$, where $q_j(x)$ are various irreducible polynomials of degree at most b. Interpret in logarithms: $m \equiv \sum_j e_j \log q_j$ in the ring $Z/(2^n - 1)$. If $A(x)$ is not smooth, discard and try again.

Accumulate many such equations (slightly more equations than irreducible polynomials of degree at most b). Indexing the equations by i, we have $m_i \equiv \sum_j e_{ij} \log q_j$. Solve this collection of linear equations, using sparse matrix techniques. This yields $\log q_j$ for all irreducible polynomials q_j of degree at most b.

When given a particular field element $B(x)$ whose logarithm we want, choose a random integer m', calculate $A(x) \equiv B(x)x^{m'} \bmod P(x)$, and keep trying until $A(x)$ is smooth. Then simply read off the logarithm from the relation $A(x) \equiv B(x)x^{m'} \equiv \prod_j q_j(x)^{e_j} \bmod P(x)$ and the previously computed logarithms of the $q_j(x)$.

One major component of the running time of this algorithm is the search for smooth functions $A(x)$ in the compilation of the initial data base. We need to gather about $2^{b+1}/b$ equations, since there are about that many irreducible polynomials of degree no more than b. A random polynomial $A(x)$ of degree n is smooth with respect to b with probability $(n/b)^{-(1+o(1))n/b}$, for n and b in the range of interest, $n^{1/3} \leqslant b \leqslant n^{2/3}$ [13]. Thus the total number of trials necessary is about

$$(2^{b+1}/b)(n/b)^{+(1+o(1))n/b} = \exp\left(\left[c\log 2 + \frac{1}{2c} + o(1)\right]\sqrt{n\log n}\right),$$

with the particular choice of $b = c\sqrt{n\log n}$. (Here "log" denotes the natural logarithm, to the base e.) One chooses c (and b) to minimize this work factor at $\exp((1 + o(1))\sqrt{2n(\log n)(\log 2)})$. (In fact, one might select b to be smaller, since otherwise, depending on the sparse matrix techniques used, one might find the subsequent sparse matrix calculations to be more costly than this first stage. This is brought out in [13] and will be mentioned in Section IX below.)

In the case $GF(2^{127})$, one might select $b = 23$. Then there are 766 150 irreducible polynomials of degree at most 23, and at least that many equations must be developed. A random polynomial of degree at most 126 will be smooth with respect to $b = 23$ with probability 0.000138. Thus, for each equation, one expects to try 7692 random values of $A(x)$ before finding an appropriate one. So the precomputation requires about 5 549 000 000 smoothness tests. (This analysis is from [4].) At 250 microseconds per test, this would require about 400 hours.

III. The Waterloo Improvement

The bottleneck in Adleman's algorithm lies in the probability of smoothness. He is dealing with random polynomials of degree n, and waiting for them to be smooth. If one

could instead deal with random polynomials of degree less than n, one would have a better chance of smoothness, and a correspondingly faster algorithm. This was the objective of the work of Blake, Fuji-Hara, Mullin, and Vanstone at University of Waterloo [4], as well as the present work.

Blake, Fuji-Hara, Mullin, and Vanstone adapted Adleman's algorithm as follows. Having generated a random polynomial $A(x) \equiv x^m \bmod P(x)$ (during the precomputation stage) or $A(x) \equiv B(x)x^{m'} \bmod P(x)$ (during the actual usage), they use the extended Euclidean algorithm to develop polynomials $C(x)$ and $D(x)$, each of degree at most $n/2$, satisfying

$$A(x)D(x) \equiv C(x) \bmod P(x).$$

They then wait for both $D(x)$ and $C(x)$ to be smooth. When this happens, they have an equation of the form

$$A(x)D(x) \equiv x^m \prod_j q_j(x)^{e_j} \equiv \prod_k q_k(x)^{f_k} \equiv C(x) \bmod P(x),$$

which they use just as Adleman uses his equations. The point is that it is more likely that two random polynomials of degree at most $n/2$ are both smooth, than that one random polynomial of degree at most $n-1$ is smooth.

The Waterloo group worked on the case of $\mathrm{GF}(2^{127})$. With a choice of $b = 17$, there are 16 510 irreducible polynomials of degree at most b. If one randomly chooses $C(x)$ and $D(x)$, of degree at most 63, with $C(x)$ and $D(x)$ relatively prime, the probability of both $C(x)$ and $D(x)$ being smooth with respect to $b = 17$ is about $1/7277$, so that the precomputation should involve about 120 000 000 smoothness tests. Again assuming 250 microseconds for a smoothness test, this gives an estimated running time of nine hours.

Asymptotically, the Waterloo scheme has a running time of $\exp((1 + o(1))\sqrt{2n(\log n)(\log 2)})$, the same as Adleman's. But the "$o(1)$" term is much better than Adleman's, allowing for the marked decrease in running time. In the present work, we will ask that two polynomials of degree $n^{2/3}\log^{1/3} n$ both be smooth. As $n^{2/3}\log^{1/3} n$ is much less than either n or $n/2$, we will have a much better probability of smoothness, and thus a much better running time, than either Adleman or Blake *et al.*

IV. Systematic Equations

Besides the practical improvements mentioned in the previous section, Blake, Fuji-Hara, Mullin, and Vanstone [4] introduced a new theoretical idea, that of "systematic equations," which opened up a new line of thought. The present work is a continuation of this line of thought.

Consider the case $\mathrm{GF}(2^{127})$. Let $P(x) = x^{127} + x + 1$, which is primitive. We have that

$$x^{128} \equiv x^2 + x \bmod P(x). \tag{4.1}$$

Notice two facts about this equation: on the left-hand side, 128 is a power of 2, and on the right-hand side, $x^2 + x$ is of very low degree.

Let $A(x)$ be an irreducible polynomial over $\mathrm{GF}(2)$ of degree $k \leqslant b$. Since 128 is a power of 2, raising to the 128th power is a linear operation in fields of characteristic two. Thus $A(x)^{128} = A(x^{128})$. By (4.1), $A(x^{128}) \equiv A(x^2 + x) \bmod P$. Finally, $A(x^2 + x)$, considered as a polynomial in x, has degree $2k$. It can be shown that either $A(x^2 + x)$ is itself irreducible or it factors into two irreducible polynomials B and C, each of degree k. In the latter case we would have

$$A(x)^{128} \equiv B(x)C(x) \bmod P(x).$$

This would give a relation among the logarithms of three irreducible polynomials, each of degree $k \leqslant b$. (The other case will also give a useful equation if $2k \leqslant b$.)

Letting $A(x)$ range through all the irreducible polynomials of degree at most b, this procedure gives about half the equations necessary to construct the database for $\mathrm{GF}(2^{127})$, at very little cost. The rest of the equations must still be obtained by the techniques outlined in Section III. ($\mathrm{GF}(2^{127})$ is an especially favorable case because of the low degree of the right-hand side of (4.1). In another field, if the right-hand side has degree d, we will get at most $1/d$ of the required equations by this technique. Further, a heuristic argument implies that this degree d will typically be about $\log_2 n$.)

The present work will generalize this concept, in effect finding a richer source of "systematic equations," so that *all* of the necessary equations can be obtained at low cost.

V. The Present Scheme: An Example

Here we demonstrate the basic idea behind our algorithm. As before, we consider the example $F = \mathrm{GF}(2^{127})$, with the primitive polynomial being $P(x) = x^{127} + x + 1$.

We consider the first stage of precomputation, that of building up the linear equations relating logarithms of the various irreducible polynomials of degree at most $b = 17$. Select $A(x)$ and $B(x)$, two polynomials over $\mathrm{GF}(2)$, each of degree at most 10, such that $\gcd(A(x), B(x)) = 1$. There are two million (2^{21}) such choices available. Set $C(x) = x^{32}A(x) + B(x)$, and set $D(x) \equiv C(x)^4 \bmod P(x)$. We compute:

$$\begin{aligned} D(x) &\equiv C(x)^4 \bmod P(x) \\ &\equiv (x^{32}A(x) + B(x))^4 \bmod P(x) \\ &\equiv x^{128}A(x)^4 + B(x)^4 \bmod P(x) \\ &\equiv (x^2 + x)A(x)^4 + B(x)^4 \bmod P(x), \end{aligned}$$

since raising to the fourth power is a linear operation in fields of characteristic two, and since $x^{128} = x^2 + x \bmod P(x)$. By our choice of $A(x)$ and $B(x)$, we find that both $C(x)$ and $D(x)$ have degree at most 42. Test $C(x)$ and $D(x)$ for smoothness with respect to the bound $b = 17$. If both are smooth, then factor each explicitly into a product of irreducible polynomials of degree at most 17. Then

$$D(x) \equiv C(x)^4 \bmod P(x)$$

becomes an equation relating the small irreducible polynomials multiplicatively (mod $P(x)$), and can be interpreted as a linear relation among the logarithms of these irreducible polynomials in the field GF(2^{127}).

Repeat this procedure for all choices of $A(x)$ and $B(x)$ such that $\gcd(A(x), B(x)) = 1$. This involves two million smoothness tests, or eleven minutes. The procedure yields about 47 000 equations relating the 16 510 unknown logarithms. Alternatively, selecting $b = 12$, the procedure yields 1061 equations in the 747 unknown logarithms.

VI. Building the Database

Using the previous example as a guide, we show how our algorithm can be applied to larger fields GF(2^n).

Choose a primitive polynomial $P(x)$ of degree n, such that $P(x) = x^n + Q(x)$, where the degree of $Q(x)$ is smaller than $n^{2/3}$. (This should be possible; heuristically, for a given n, we expect the best possible $Q(x)$ to have degree about $\log_2 n$.) We remark that if the field is presented to us with a different defining polynomial $P_1(x)$, we can just solve the problem in our preferred presentation (given by $P(x)$) and transfer the results to the given presentation (given by $P_1(x)$) [17].

The first step is to construct a database of many linear equations relating the logarithms of small irreducible polynomials. Select a bound $b \geq c_1 n^{1/3} \log^{2/3} n$ for a small constant c_1. Irreducible polynomials of degree at most b will be considered "small." Choose an integer d near b; we will see later how to select d. Let k be a power of 2 near $\sqrt{n/d}$. Let h be the least integer greater than n/k. In our example, we had $b = 17$, $d = 10$, $k = 4$, and $h = 32$.

Set $R(x) = x^{hk} \bmod P(x)$, so that $R(x) = Q(x)x^{hk-n}$. Let $r = \deg R(x)$. If $P(x)$ was well chosen, r may be less than k. Select a pair of polynomials $A(x)$ and $B(x)$, both of degree at most d, with $\gcd(A(x), B(x)) = 1$. (The gcd condition avoids redundant equations: if $A(x) = e(x)a(x)$ and $B(x) = e(x)b(x)$, then the equation obtained from the pair $(A(x), B(x))$ is the same as that obtained from the pair $(a(x), b(x))$.) Set $C(x) = x^h A(x) + B(x)$, and set $D(x) = C(x)^k \bmod P(x)$. We compute:

$$\begin{aligned} D(x) &= C(x)^k \bmod P(x) \\ &= \left(x^h A(x) + B(x)\right)^k \bmod P(x) \\ &= x^{hk} A(x)^k + B(x)^k \bmod P(x) \\ &= R(x) A(x)^k + B(x)^k \bmod P(x). \end{aligned}$$

The degree of C is at most $h + d$, and the degree of D is at most $r + kd$. Both these bounds are about $\sqrt{nd}$. If C and D were independent random polynomials of these degrees, then both would be smooth with probability about

$$\left(\frac{h+d}{b}\right)^{-(h+d)/b}\left(\frac{r+kd}{b}\right)^{-(r+kd)/b} \sim \left(\frac{\sqrt{nd}}{b}\right)^{-2\sqrt{nd}/b} = \exp\left(-\frac{\sqrt{nd}}{b}\log\frac{nd}{b^2}\right)$$

[13]. By choice of b and d, this is $\exp(-c_2 n^{1/3}\log^{2/3} n)$ for some constant c_2. (In fact C and D are neither random nor independent, so this estimate of the probability of smoothness must be viewed as only heuristic.)

If both C and D are smooth, proceed as in the example. Express C and D explicitly as the product of small irreducible polynomials, convert the equation $D(x) \equiv C(x)^k \bmod P(x)$ to a linear equation relating the logarithms of these irreducible polynomials in the field, and add this linear equation to our database. Thus

$$C(x) = \prod_j q_j(x)^{e_j},$$

$$D(x) = \prod_j q_j(x)^{f_j},$$

$$D(x) \equiv C(x)^k \bmod P(x),$$

$$\sum_j (ke_j - f_j)\log q_j \equiv 0.$$

We need to find slightly more equations than there are irreducible polynomials of degree at most b. Thus we need about $2^{b+1}/b$ equations. The number of tests required to develop each equation is about

$$\exp\left(+\frac{\sqrt{nd}}{b}\log\frac{nd}{b^2}\right) = O\left(\exp\left(c_2 n^{1/3}\log^{2/3} n\right)\right)$$

from the above discussion. Thus the total amount of work necessary to develop enough equations is about

$$\exp\left(b\log 2 + \frac{\sqrt{nd}}{b}\log\frac{nd}{b^2}\right) = O\left(\exp\left(c_3 n^{1/3}\log^{2/3} n\right)\right)$$

for a small constant c_3.

This forces a relation between d and b: the 2^{2d+1} pairs $(A(x), B(x))$ of relatively prime polynomials of degree at most d must be sufficient to provide for the $O(\exp(c_3 n^{1/3}\log^{2/3} n))$ tests.

The time required to invert the resulting sparse equations is of the same general form, roughly $\exp(c_4 b)$. Thus we have constructed a database consisting of the logarithms of all irreducible polynomials of degree at most b, using time and storage $O(\exp(O(n^{1/3}\log^{2/3} n)))$.

VII. Logarithms of Medium-Sized Polynomials

A second feature of our present algorithm is the ability to take logarithms of moderate-sized polynomials easily. That is, if we are given $G(x)$, where degree $G(x)$ is, say, $n^{2/3}$, we can take advantage of its small degree when taking its logarithm. This is in contrast to both Adleman's algorithm and the Waterloo improvement. Both algorithms begin by multiplying $G(x)$ by $x^m \bmod P(x)$, thus destroying its small degree, in order to randomize. Their only hope is that $G(x)$ might itself be smooth, and if it is not, anything they do to it (multiplying by x^m, extended Euclidean algorithm, etc.) destroys its small degree.

We use the techniques introduced in Sections V and VI. Suppose our database has been constructed with degree bound b. Suppose we are given $G(x)$ of degree $g \ll n$. Select d near $(g + \sqrt{n/b}\log_2(n/b))/2$. Let k be a power

of 2 near $\sqrt{n/d}$. Let h be the least integer greater than n/k. Choose $A(x)$ and $B(x)$ from among polynomials of degree at most d, subject to the usual restriction that $\gcd(A(x), B(x)) = 1$, and the new restriction that $G(x)$ divides $C(x) = x^h A(x) + B(x)$. (Given $G(x)$, it is a relatively easy matter to find the linear space of applicable pairs $(A(x), B(x))$.) Set $D(x) \equiv C(x)^k \bmod P(x)$. When both $C(x)/G(x)$ and $D(x)$ are smooth with respect to the bound $b' = [\sqrt{bg}]$, then we have expressed $\log G(x)$ in terms of the logarithms of irreducible polynomials of degree at most b'. Further, there are fewer than $2g < n$ such irreducible polynomials.

By the usual arguments, the number of trial pairs $(A(x), B(x))$ needed should be on the order of

$$\exp\left(2(h/b')\log(h/b')\right) \sim \exp\left(\sqrt{nd/bg}\log(nd/bg)\right) \sim \exp\left(\sqrt{n/b}\log(n/b)\right).$$

By our choice of d, we should have enough pairs $(A(x), B(x))$ available to find one pair that works. If not, choose a slightly larger value of d.

VIII. The Algorithm and Asymptotic Running Time

The algorithm is just a combination of the two steps just outlined. First we build a data base of logarithms of irreducible polynomials of degree up to bound $b = cn^{1/3}\log^{2/3} n$. The data base has size $2^{b+1}/b$ and takes time $\exp(c'n^{1/3}\log^{2/3} n)$ to construct.

Given a polynomial $G(x)$ whose logarithm we want, first we follow the scheme of Blake *et al.* outlined in Section III. For a random m, evaluate $G(x)x^m \bmod P(x)$. Use the extended Euclidean algorithm to set $G(x)x^m \equiv G_1(x)/G_2(x) \bmod P(x)$, where G_1 and G_2 have degree about $n/2$. Wait until both G_1 and G_2 are smooth with respect to the bound $b_1 = \sqrt{nb}$. Factor G_1 and G_2 into polynomials of degree at most b_1. So far we have spent time about $\exp(2(n/b_1)\log(n/b_1)) = \exp(\sqrt{n/b}\log(n/b))$, and we have fewer than n polynomials of degree between b and b_1, whose logarithms we must obtain. For each of these polynomials, apply the technique in Section VII. Each polynomial requires time $\exp(\sqrt{n/b}\log(n/b))$ to be expressed as the product and quotient of fewer than n irreducible polynomials of degree at most $b_2 = \sqrt{bb_1}$. To each of the resulting polynomials, apply the technique in Section VII again. After t stages, we have expressed $G(x)$ in terms of fewer than n^t polynomials, each of degree less than $b_t = b(n/b)^{2^{-t}}$. We have taken work less than

$$n^t \exp\left(\sqrt{n/b}\log(n/b)\right) = \exp\left(\sqrt{n/b}\log(n/b) + t\log n\right)$$

to do so. When $t = \log n$, all of our polynomials are in the original database, and we have found the logarithm of the given $G(x)$.

Thus we require time

$$\exp\left(\sqrt{n/b}\log n/b + \log^2 n\right) = \exp\left((1 + o(1))\sqrt{n/b}\log(n/b)\right)$$

to find each new logarithm. Selecting, for example, $b = n^{1/3}\log^{2/3} n$, this time is $O(\exp((\frac{2}{3} + o(1))n^{1/3}\log^{2/3} n))$. If b is larger, the larger data base allows us a smaller per-logarithm time.

IX. Dependence on Sparse Matrix Techniques

So far, our analysis of running time has concentrated on the first phase of precomputation, the acquisition of equations. But the second phase, the solution of these linear equations, may turn out to be the dominating factor.

Suppose that a sparse system of N linear equations in N unknowns can be solved in time $O(N^\omega)$, where by "sparse" we mean that each equation has about $\log N$ nonzero coefficients per equation. Using straight Gaussian elimination we would have $\omega = 3$; Strassen [16] would give $\omega = 2.807$; Coppersmith and Winograd [5] would give $\omega = 2.496$; all of these treat the matrix as dense. Taking advantage of the sparseness one might find $\omega \leqslant 2$. It turns out that this second stage will be a determining factor only if $\omega \geqslant 2$.

For a fixed value of ω, we relate d (the degree of $A(x)$ or $B(x)$) and b (the smoothness bound) by two equations. Balancing the work of the first stage (2^{2d+1} trials) against that of the second stage (solution of linear equations) we have

$$2^{2d+1} \sim \left(\frac{2^{b+1}}{b}\right)^\omega,$$

whence

$$2d \sim \omega b.$$

Requiring that the 2^{2d+1} trials give enough successful pairs $(A(x), B(x))$, we get

$$2^{2d+1} \sim \left(\frac{2^{b+1}}{b}\right)\exp\left(2\frac{\sqrt{nd}}{b}\log\frac{\sqrt{nd}}{b}\right),$$

or

$$2d\log 2 \sim (b\log 2) + \frac{2}{3}\frac{\sqrt{nd}}{b}\log n.$$

Solving, we find

$$b \sim \left(\frac{2\omega n\log^2 n}{9(\omega - 1)^2\log^2 2}\right)^{1/3}$$

and the work is

$$\exp\left[(1 + o(1))\left\{\frac{2\omega^4\log 2}{9(\omega - 1)^2}n\log^2 n\right\}^{1/3}\right].$$

Recall we selected k to be a power of 2 near $\sqrt{n/d}$. The preceding analysis assumed $\sqrt{n/d}$ was close to a power of 2, so that $\deg C$ and $\deg D$ were both around $\sqrt{nd}$. This is the best case. In the worst case, $\sqrt{2n/d}$ is close to a power of 2. A similar analysis there yields a work factor of

$$\exp\left[(1 + o(1))\left\{\frac{\omega^4\log 2}{4(\omega - 1)^2}n\log^2 n\right\}^{1/3}\right].$$

TABLE I
DEPENDENCE OF THE WORK FACTOR EXPONENT ON ω

Matrix Exponent ω	Best Case c	Case c	Comment
3	1.461	1.520	Gaussian Elimination
2.807	1.431	1.488	Strassen
2.496	1.387	1.443	Coppersmith–Winograd
2	1.351	1.405	Possible Sparse Techniques
< 2	1.351	1.405	No Longer a Bottleneck

Summarizing, we find that the overall work is $\exp((c + o(1))n^{1/3}\log^{2/3} n)$, where the dependence of c on the matrix exponent ω is given in Table I.

X. SPECIFIC CASES

These asymptotic running times are encouraging. But we feel compelled to analyze some specific cases, to see how the algorithm works in practice.

In the first stage, call an "operation" a smoothness test. Assume, for concreteness, that in the second stage, the solution of the N linear equations in N unknowns requires $N^3/3$ operations (Gaussian elimination, without taking advantage of sparseness, i.e., the most pessimistic assumption), where now an "operation" is a multiplication of two elements of the ring $Z/(2^n - 1)$(i.e., the multiplication of two integers mod $2^n - 1$).

We develop the estimates of Table II. The first column gives n. The second column gives the total number of "operations" involved in precomputation (either smoothness tests or multiplications in $Z \bmod 2^n - 1$). The third column gives the number of smoothness tests; the fourth, the number of multiplications. Columns 5 through 7 give the parameters b, d, and k.

To construct Table II we first selected b, and estimated the work involved in solving an $N \times N$ system of equations, where $N \sim 2^{b+1}/b$. We selected d so that the 2^{2d+1} smoothness tests would cost about as much as the solution of the linear equations. It turned out that $k = 4$ was the optimal setting of k throughout the range. Setting $\deg D = kd + k$, we found the probability that D was smooth. We could then hypothesize a probability for C being smooth, which would make the choice of d just sufficient to get enough equations. We worked backwards then to obtain $\deg C$, from which we could calculate n.

TABLE II
NUMBER OF OPERATIONS REQUIRED BY THE FIRST STAGE

n	Total	Stage 1	Stage 2	b	d	k
96	7.49E6	3.64E6	3.84E6	10	11	4
124	3.95E7	1.62E7	2.33E7	11	12	4
156	3.68E8	2.29E8	1.39E8	12	14	4
188	1.83E9	9.56E8	8.70E8	13	15	4
220	9.17E9	3.72E9	5.45E9	14	16	4
268	1.02E11	6.66E10	3.51E10	15	18	4
304	4.76E11	2.49E11	2.27E11	16	19	4
344	2.53E12	1.03E12	1.50E12	17	20	4
400	2.67E13	1.67E13	9.97E12	18	22	4
444	1.35E14	6.78E13	6.72E13	19	23	4
488	7.15E14	2.59E14	4.56E14	20	24	4
552	7.04E15	3.91E15	3.13E15	21	26	4
604	3.88E16	1.73E16	2.16E16	22	27	4
676	4.25E17	2.75E17	1.50E17	23	29	4

As we will see in the next section, GF(2^{127}) requires less than an hour to construct a data base for $b = 12$. Allowing 250 microseconds per "operation" (which was what we observed for the smoothness tests), this table would indicate three hours running time. One difference seems to be that we used $b = 12$ while the table indicates using $b = 11$; the smoothness probabilities must be fairly unstable in this region. The table's choice of $b = 11$ was influenced by the assumption of $N^3/3$ operations for the equation solving, while we had better running time than that.

Allowing one millisecond per "operation" for larger values of n, one can guess that GF(2^{241}) could be done in a year on an existing mainframe computer. Special-purpose chips might bring the "operation" time to the microsecond range, allowing calculations in fields of size 2^{400}. Better sparse matrix techniques would further increase the effective range of this algorithm, as would several chips running parallel.

XI. PROGRAMMING CONSIDERATIONS (WITH JAMES DAVENPORT, UNIVERSITY OF BATH, ENGLAND)

We have completely determined the logarithms of all polynomials of degree at most 12 in GF(2)$[x]/(x^{127} + x + 1)$, and looked at the possibility of extending this to higher degree, in particular 13. We note that, for the sieve size we chose ($d = 10$), 12 is the smallest number at which we can start this process—there are not enough equations to determine the logarithms of polynomials of degree at most 11 directly.

There are essentially three steps to building a data base of logarithms for all polynomials of degree at most b:

1) finding a sufficient number of polynomials $A(x), B(x)$ such that $C(x) = A(x)x^{32} + B(x)$ and $D(x) \equiv C(x)^4 \bmod P(x) = A(x)^4(x^2 + x) + B(x)^4$ are both smooth with respect to b;
2) factoring the polynomials $C(x)$ and $D(x)$ thus produced—each factorization giving a linear equation among the logarithms of the irreducible factors of $C(x)$ and $D(x)$;
3) solving these equations.

Step 1) was performed with a special machine-code program. To test whether $C(x)$ is smooth, we evaluate $C'(x)\prod_{k \leq b}(x^{2^k} - x) \bmod C(x)$. The factor $x^{2^k} - x$ contains as factors all irreducible polynomials of degree k, as well as all irreducible polynomials of degree j where j divides k. The factor $C'(x)$ (the formal derivative of $C(x)$) contains all repeated irreducible factors of $C(x)$. Thus the indicated product will be zero when $C(x)$ is smooth. By testing the product after each value of k, we can determine the size of the largest irreducible factor. In 250 microseconds (4000 instructions) per pair $(C(x), D(x))$, this program made six lists of pairs, according to the size of the largest irreducible factor appearing in either: up to 12, exactly 13, 14, 15, 16, or 17, and discarding pairs containing a larger irreducible.

The product will also be zero when $C(x)$ has a large repeated irreducible factor. In this rare event, it will falsely report "smoothness." In the actual calculations, of two million trials, the program reported that 1094 pairs $(C(x), D(x))$ were smooth with respect to the bound $b = 12$. Of these, 33 were false reports, the other 1061 giving useful equations.

Steps 2) and 3) were performed in the NEWSPAD algebra system [9] since their performance is (relatively) uncritical to the running of the algorithm, and since they can be solved by well-known algorithms.

The factorizations over GF(2) were performed by a three-stage process:

a) the problem is reduced to the factorization of square-free polynomials;
b) the problem is reduced to the factorization of polynomials all of whose irreducible factors are of the same degree;
c) such polynomials are factored.

We wrote (fortunately less than an hour's task) a special domain in NEWSPAD to handle GF(2), in which 0 and 1 were represented by the LISP objects *NIL* and *non-NIL*. The use of this, rather than a general domain which represented GF(p) as integers in the range $[0, p-1]$, halved the running time of the program. We used our general representation for sparse polynomials, as a list of exponent-coefficients pairs for all nonzero terms, even though there is only one nonzero coefficient, since we did not wish to embark on the task of writing a new polynomial representation. Clearly such a representation, e.g., by bit-vectors, could be written, and it would be much more efficient.

Step b) was performed by Knuth's [10, p. 389] algorithm *S*, which he attributes to being "fairly well-known." Step c) is often not needed, but, when it was, we used Berlekamp's "small field" algorithm [2]. The reader may wonder why we use Step b), since, as [3] points out, for small primes, his method is generally faster, even though the running times of the two methods have the same asymptotic growth, as functions of degree of the polynomials being factored. This is indeed true, but our polynomials are far from being "general." Indeed, all the "correct" ones have factors of degree at most 12. For our application, the method outlined is more than twice as fast as a direct application of Berlekamp's algorithm, and takes 8 minutes 21 seconds to factor the 2188 polynomials.

Cantor and Zassenhaus [6] propose another algorithm for the splitting of factors remaining after Step 2). In the case of GF(2), their algorithm requires one to work over a quadratic extension of GF(2). Since such programs were not available at the time these calculations were performed, we did not investigate this algorithm further, though it may in fact be more efficient.

The linear equations were solved by a relatively straightforward sparse matrix method written specially for the application (157 lines of code). There are eight irreducibles of degree at most 12 which do not appear in the equations resulting from the previous step, hence the rank of the matrix is at most $747 - 8 = 739$. In fact it is exactly 739, and the logarithms of the irreducibles that appear are discovered. The solution took fractionally under 20 minutes of CPU. This is the only step that required a significant amount of storage, using 1 880 768 bytes of working storage at its peak. The solution could probably be found more readily—since the system is over-determined, one could use fewer equations than the 1062 found from the previous step. We estimate that such improved algorithms could halve the amount of time taken.

We could, of course, find all the logarithms of polynomials of degree at most 13 the same way, but there is an easier way. The same sieve as described above found a further 1928 pairs $A(x), B(x)$ such that $C(x)$ and $D(x)$ were (apparently) smooth with respect to 13, but not 12. Factoring the resulting $C(x)$ and $D(x)$ showed that 18 of them were spurious, thus leaving 1910 valid equations for the logarithms of the 630 irreducible polynomials of degree 13. These equations determine the eight missing polynomials of degree 12, and 1205 of them only involve one polynomial of degree 13, thus enabling us to find the logarithms of 550 such polynomials (there are many duplications, of course) immediately. Simple back-substitution into the remaining equations enables us to find 77 of the remaining 80, and the last three never occur in these equations, and so have to be postponed until the next stage. The amount of working storage required for this stage is negligible, consisting of that for the factorization plus enough to hold the known logarithms, say 38 556 bytes. Furthermore, this step (and subsequent liftings to 14 ...) are highly parallelizable, since each factorization could be performed on a separate processor.

Based on these figures, we can extrapolate the total time required to build a database of logarithms of all irreducible polynomials of degree at most 17. The results are contained in Table III. Times in italics are estimates. All times are in CPU minutes on an IBM 3081 model K, and include garbage collection and operating system overhead.

TABLE III
TIME (IN MINUTES) FOR THE FIRST PHASE OF THE ALGORITHM FOR GF(2^{127})

Time	Operation	Factorizations
11	Sieve	
8	Factorization—$b = 12$	2188
20	Solve	
18	Factorization—$b = 13$	3856
40	Factorization—$b = 14$	8356
80	Factorization—$b = 15$	15946
130	Factorization—$b = 16$	26038
190	Factorization—$b = 17$	38966
497	Total	

ACKNOWLEDGMENT

Thanks are due to I. F. Blake, R. Fuji-Hara, R. C. Mullin, and S. A. Vanstone, who showed me their work and got me interested in the subject. Stimulating conversa-

tions with Victor Miller, Andrew Odlyzko, and James Davenport are gratefully acknowledged; Odlyzko, in particular, provided references, analysis, and ideas for future work. The author wishes to thank the Computer Algebra Group at IBM Research, particularly James Davenport and Patrizia Gianni, for adapting their symbolic algebra package to factor the resulting polynomials and to do the sparse matrix calculations involved in trying GF(2^{127}).

REFERENCES

[1] L. Adleman, "A subexponential algorithm for the discrete logarithm problem with applications to cryptography," in *Proc. IEEE 20th Annual Symposium on Foundations of Computer Science*, 1979, pp. 55–60.
[2] E. R. Berlekamp, "Factoring polynomials over finite fields," *Bell Syst. Tech. J.*, vol. 46, pp. 1853–1859, 1967.
[3] ——, "Factoring polynomials over large finite fields," *Math. Comp.*, vol. 24, pp. 713–735, 1970.
[4] I. F. Blake, R. Fuji-Hara, R. C. Mullin, and S. A. Vanstone, "Computing logarithms in finite fields of characteristic two," to appear in *SIAM J. Algebraic and Discrete Methods*.
[5] D. Coppersmith and S. Winograd, "On the asymptotic complexity of matrix multiplication," *SIAM J. Comput.*, vol. 11, no. 3, pp. 472–492, Aug. 1982.
[6] D. G. Cantor and H. Zassenhaus, "A new algorithm for factoring polynomials over finite fields," *Math. Comp.*, vol. 36, pp. 587–592, 1981.
[7] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 644–654, 1976.
[8] M. E. Hellman and J. M. Reyneri, "Fast computation of discrete logarithms in GF(q)," *Advances in Cryptography: Proceedings of CRYPTO '82*, D. Chaum, R. Rivest, and A. Sherman, Eds. New York: Plenum, 1983, pp. 3–13.
[9] R. D. Jenks and B. M. Trager, "A language for computational algebra," in *Proc. SYMSAC '81*, P. S. Wang, Ed. New York: ACM, 1981, pp. 6–13.
[10] D. E. Knuth, *The Art of Computer Programming*, Vol. 2. New York: Addison-Wesley, 1971, pp. 351–354.
[11] ——, *The Art of Computer Programming*, Vol. 3. New York: Addison-Wesley, 1973, p. 9.
[12] M. Morrison and J. Brillhart, "A method of factoring and the factorization of F_7," *Math. Comp.*, vol. 29, pp. 183–205, 1975.
[13] A. M. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance," to appear in *Proceedings of Eurocrypt '84*.
[14] W. W. Peterson, *Error-Correcting Codes*. Cambridge, MA: MIT 1961.
[15] S. C. Pohlig and M. Hellman, "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 106–110, 1978.
[16] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, pp. 354–356, 1969.
[17] N. Zierler, "A conversion algorithm for logarithms on GF(2^n)," *J. Pure and Appl. Algebra*, vol. 4, pp. 353–356, 1974.

Cryptanalytic Attacks on the Multiplicative Knapsack Cryptosystem and on Shamir's Fast Signature Scheme

ANDREW M. ODLYZKO, MEMBER, IEEE

Abstract—**The basic Merkle–Hellman additive trapdoor knapsack public-key cryptosystem was recently shown to be insecure, and attacks have also been developed on stronger variants of it, such as the Graham–Shamir system and the iterated knapsack cryptosystem. It is shown that some simple variants of another Merkle–Hellman system, the multiplicative knapsack cryptosystem, are insecure. It is also shown that the Shamir fast signature scheme can be broken quickly. Similar attacks can also be used to break the Schöbi–Massey authentication scheme. These attacks have not been rigorously proved to succeed, but heuristic arguments and empirical evidence indicate that they work on systems of practical size.**

Manuscript received April 22, 1983; revised December 11, 1983. This work was presented at Eurocrypt '83, Udine, Italy, March 21–25, 1983, and also at the International Symposium on Information Theory, St. Jovite, PQ, Canada, September 26–30, 1983.

The author is with AT & T Bell Laboratories, Room 2C-370, Murray Hill, NJ 07974.

I. INTRODUCTION

ONE of the best-known public-key cryptosystems, the basic Merkle–Hellman additive trapdoor knapsack system [18], was recently shown by Shamir to be easy to break [25]. Subsequently, Adleman [2] proposed attacks on the Graham–Shamir scheme and on the multiply iterated Merkle–Hellman system. Adleman's attack on the general multiply iterated knapsack systems does not seem to succeed [3], [7], but other attacks on the doubly iterated knapsack schemes have been proposed by Adleman and Lagarias (see [7], [14]). Furthermore, Brickell [6] and Lagarias and the author [15] have developed attacks on low-density knapsack cryptosystems. This paper develops attacks on several other public-key cryptosystems. We show

0018-9448/84/0700-0594$01.00