COMPUTING LOGARITHMS IN GF (2^n)*

I.F. Blake[1], R.C. Mullin[2], S.A. Vanstone[2]

[1]Department of Electrical Engineering
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

[2]Department of Combinatorics and Optimization
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

1. The Problem.

Consider the finite field having q elements and denote it by $GF(q)$. Let α be a generator for the nonzero elements of $GF(q)$. Hence, for any element $b \neq 0$ there exists an integer x, $0 \leq x \leq q-2$, such that $b=\alpha^x$. We call x the discrete logarithm of b to the base α and we denote it by $x = log_\alpha b$ and more simply by $log\ b$ when the base is fixed for the discussion. The discrete logarithm problem is stated as follows:

Find a computationally feasible algorithm to compute $log_\alpha b$ for any $b \in GF(q)$, $b \neq 0$.

Several cryptographic schemes have been proposed which base their security on the intractability of the discrete logarithm problem for large q.

In 1976 Diffie and Hellman [7] proposed the following public key passing scheme. Let A and B be two parties who wish to share a common key K. The finite field $GF(q)$ and a generator α are stored in a public file. Party A generates a random integer a and computes α^a. Party B generates a random integer b and computes α^b. A sends to B the field element α^a and B sends to A the field element α^b. A computes $(\alpha^b)^a=\alpha^{ab}$ and B computes $(\alpha^a)^b=\alpha^{ab}$. A and B now share the common key $K=\alpha^{ab}$.

Recently, ElGamel [8] has proposed a public key cryptosystem and an authentication scheme which is based on discrete exponentiation. We describe only the public key system here.

Consider $GF(q)$ generated by α. The message space M will consist of all nonzero field elements. Party A generates a random integer a and stores α^a in a public file. Suppose party B wishes to send a message m to A. B generates a random integer k and computes $(\alpha^a)^k$ (since α^a is made public this is possible), α^k and $m\alpha^{ak}$. B sends A the pair $(\alpha^k, \alpha^{ak}m)$. Since A knows a he can compute $(\alpha^k)^a=\alpha^{ak}$, and, hence, obtain m from $\alpha^{ak}m$. This scheme and the Diffie-Hellman method appear to have the same degree of security. It remains an open problem as to whether or not the security of these systems is entirely dependent on comput-

*Research supported in part by the Department of Communications under contract # 20ST 36001-4-0853.

G.R. Blakley and D. Chaum (Eds.): Advances in Cryptology - CRYPTO '84, LNCS 196, pp. 73-82, 1985.

ing discrete logarithms.

A number of special purpose algorithms for computing discrete logs have appeared in the literature (see, for example, [2], [9], [11], [14]). In this article we address only the most general methods currently available.

In the next three sections we describe three subexponential algorithms for computing logs. The algorithms of sections 3 and 4 are variants of the one presented in section 2. These algorithms are referred to as the index-calculus algorithms in an excellent and in-depth article on the subject by Odlyzko [13].

For the purposes of this paper we restrict our discussion to $GF(2^n)$. (The algorithms of sections 2 and 3 apply more generally.) We think of the elements in $GF(2^n)$ as polynomials of degree at most $n-1$ over $GF(2)$ and multiplication is performed modulo some fixed irreducible polynomial of degree n over $GF(2)$. The examples cited in this paper refer to $GF(2^{127})$ as it has been of some interest recently. (See, for example, [15], [17]).

2. Adleman Algorithm

The basic ideas involved in the following subexponential algorithm for computing discrete logarithms are due to Western and Miller [16]. Adleman [1] independently discovered the algorithm and partially analysed its computational complexity.

The algorithm consists of two parts. The first part requires the construction of a large database of logarithms. This database only needs to be constructed once for $GF(2^n)$. Part 2 of the algorithm computes individual logarithms.

Part 1 (Database).

Find the logarithms of all irreducible polynomials of degree at most b where b is a fixed positive integer determined by $GF(2^n)$.

Part 2.

To find the log of an element $g(x) \in GF(2^n)$, $g(x) \neq 0$, generate a random integer a and compute $h(x) = g(x)\alpha^a(x)$ where $\alpha(x)$ generates $GF(2^n)$. Now, factor

$$h(x) = \prod_{i=1}^{t} p_i^{e_i}(x).$$

If each irreducible factor $p_i(x)$ has $deg\ p_i(x) \leq b$ then

$$log\ g(x) = \sum_{i=1}^{t} e_i\ log\ p_i(x) - a$$

which can easily be evaluated by looking up $log\ p_i(x)$, $1 \leq i \leq t$, in the database. If not all $p_i(x)$ have $deg\ p_i(x) \leq b$ then generate another random integer and repeat.

We define $p(n,b)$ to be the probability that a randomly chosen polynomial of degree exactly n has all of its irreducible factors with degrees of at most b. If $N(n,b)$ is the number of polynomials of degrees exactly n such that each has all of its irreducible factors with degree at most b then

$$p(n,b) = \frac{N(n,b)}{2^n}.$$

and the expected runtime of the second part of the algorithm should be approximately $p(n,b)^{-1}$. Odlyzko [1] shows that

$$p(n,b)^{-1} \cong \left(\frac{n}{b}\right)^{(1+O(1))\frac{n}{b}}$$

and that the asymptotic running time of the entire algorithm is $exp\ (c_1(n\ log\ n)^{\frac{1}{2}})$.

Let S be the set of all irreducible polynomials of degree at most b. In order to find the logarithms of all elements in S we set up a system of $|S|$ linear equations in $|S|$ unknowns where the unknowns are the logarithms. We can find this system of equations by applying part 2 of the algorithm to each element of S. The resulting system must be solved over the integers modulo 2^n-1. The number of iterations of part 2 to produce the necessary equations is approximately $|S|p(n,b)^{-1}$. For $GF(2^{127})$ this quantity is minimized by $b=23$. Since there are 766,150 irreducible polynomials of degree at most 23 then we require this many equations. A random polynomial of degree at most 126 will factor into the database with $b=23$ with probability .000138. This means that to produce the desired set of equations will require about $5{,}549\times10^6$ iterations of part 2 of the algorithm.

The next section gives a variant of the Adleman algorithm which improves the situation for $GF(2^{127})$.

3. The Waterloo Algorithm

This algorithm (see [3]) differs from the former algorithm in two ways. In part 2 where the polynomial $h(x)=g(x)\alpha^a(x)$ is factored this algorithm applies the extended Euclidean algorithm to the polynomials $h(x)$ and $f(x)$ ($f(x)$ is the irreducible polynomial of degree n which defines $GF(2^n)$.) so that we can write

$$h(x) = \frac{s(x)}{t(x)}$$

where $(s(x),t(x))=1$ and $deg\ s(x), deg\ t(x) \le \frac{n}{2}$. One observation we can make at this point is that every polynomial $h(x)$ of degree at most n (n odd) can be written uniquely as a quotient of polynomials where each has degree at most $\frac{n}{2}$ and which are relatively prime. If both $s(x)$ and $t(x)$ factor into S then $log\ g(x)$ is readily computed by table lookup.

The advantage to this algorithm over the previous one is that it is more probable for two polynomials of degrees at most $\frac{n}{2}$ to factor into the database than it is for one polynomial of degree n. Let $N(b,i,j)$ be the number of pairs of relatively prime polynomials $(A(x),B(x))$ such that $deg\ A(x)=i, deg\ B(x)=j$ and both are smooth with respect to b. (By smooth we mean that the polynomial factors into irreducibles all of whose degrees are at most b.) For each irreducible polynomial $b(x)$ with degree $k \le b$ define the enumerator of $b(x)$ by

$$(1+y^k+y^{2k}+\cdots+z^k+z^{2k}+\cdots).$$

Letting I_k be the number of irreducible polynomials of degree k we obtain the generating function for the $N(b,i,j)$ as

$$F(y,z) = \prod_{k=1}^{b}\left[\sum_{i=0}^{n} y^{ik} + \sum_{i=1}^{n} z^{ik}\right]^{I(k)}$$

$$= \prod_{k}[(1-y^kz^k)/(1-y^k)(1-z^k)]^{I(k)}$$

$$= E(y)E(z)/E(yz)$$

where $E(y)$ is the generating function for one smooth polynomial. The probability that an ordered pair of relatively prime polynomials $(A(x),B(x))$ each of degree at most $\frac{n}{2}$ are both smooth with respect to b is

$$p^*(n,b) = \left\{\sum_{0\le i,j\le\frac{n}{2}} N(b,i,j)\right\}/2^n.$$

In the case of $GF(2^{127})$ Coppersmith [6] has evaluated this expression for $b=17$ and found $p^*(127,17) \cong \frac{1}{7000}$. In order to simplify calculations we approximate $p^*(n,b)$ by $[p(n,b)]^2$. For $GF(2^{127})$, $[p(127,17]^2 \cong \frac{1}{5000}$.

In the following table we list these probabilities for b ranging between 1 and 30 and also we list the probabilities associated with the Adleman algorithm. The table also includes the expected number of iterations to produce enough equations to construct the database for each value of b in the given range.

We see from the table that the number of iterations for the database is minimized by $b=20$. For our implementation we selected $b=17$ in order to keep the system of equations manageable.

The second difference in the Waterloo algorithm is in producing the equations for the database. We did not rely entirely on Part 2 of the algorithm. A number of equations were readily obtained by several techniques which make use of the fact that squaring is a linear operator in the field. Our principle technique here is referred to as the generation of systematic equations.

We briefly describe this technique with regard to $GF(2^{127})$ generated by $f(x)=x^{127}+x+1$. Since $f(x)$ divides $x^{128}+x^2+x$, it is easily shown that x^{2^i}, $0\le i\le 126$, can be written as $e = \sum_{j=0}^{6} \gamma_j x^{2^j}$ where $\gamma_j\in\{0,1\}$ and so the log of any element of the form e can be readily found. Similarly the log of any element of the form $\gamma_{-1}+\sum_{j=0}^{6}\gamma_j x^{2^j}$ where $\gamma_j\in\{0,1\}$, $-1\le j\le 6$ can be found. This gives 31 equations where the maximum degree is 16. Related to this idea is the observation that if $u(x)$ is any irreducible polynomial of degree d then the degrees of all irreducible polynomials of $w(u(x))$ are divisible by d. Using this method we obtained 142 linearly independent equations involving the 226 logarithms of irreducible polynomials of degree ≤ 10.

Probability and expected number of runs for the new and Adleman algorithm, $n=127$.

Degree	Total no. of irred. polys	New algorithm Probability	New algorithm Expected no. of runs	Adleman algorithm Probability	Adleman algorithm Expected no. of runs
1	1	1.27141469E−32	7.8652544E+31	0.47772090E−34	0.20932720E+35
2	2	1.61023467E−30	1.24205499E+30	0.10391370E−32	0.19246730E+34
3	4	1.42003032E−27	2.81684126E+27	0.10686600E−30	0.37430020E+32
4	7	1.15313844E−24	6.07038995E+24	0.16022050E−28	0.43689770E+30
5	13	3.46321796E−21	3.75373429E+21	0.12806100E−25	0.10151400E+28
6	22	2.43083741E−18	9.05037906E+18	0.60489980E−23	0.36369650E+25
7	40	1.75009226E−15	2.28559379E+16	0.58661830E−20	0.68187380E+22
8	70	2.99279106E−13	2.33895379E+14	0.20402690E−17	0.34309190E+20
9	126	2.39477444E−11	5.26145586E+12	0.40463370E−15	0.31139250E+18
10	225	7.73424039E−10	2.90914154E+11	0.31781350E−13	0.70796220E+16
11	411	1.42517804E−08	2.88385022E+10	0.13612270E−11	0.30193290E+15
12	746	1.48157461E−07	5.03518348E+09	0.29555160E−10	0.25240930E+14
13	1376	1.0660927E−06	1.29069451E+09	0.41129290E−09	0.33455440E+13
14	2537	5.46899677E−06	463887639	0.37461550E−08	0.67722710E+12
15	4719	2.19028276E−05	215451634	0.24991280E−07	0.18882570E+12
16	8799	7.12191038E−05	123548311	0.12723630E−06	0.69154690E+11
17	16509	1.96662481E−04	83945854.4	0.52428530E−06	0.31488570E+11
18	31041	4.71015488E−04	65902291.5	0.17992970E−05	0.17251720E+11
19	58635	1.00764675E−03	58190035.4	0.53284450E−05	0.11004140E+11
20	111012	1.96260912E−03	56563479.1	0.13895770E−04	0.79888990E+10
21	210870	3.54177251E−03	59537985.4	0.32587350E−04	0.64709140E+10
22	401427	5.96621694E−03	67283339.5	0.69740460E−04	0.57560100E+10
23	766149	9.45338803E−03	81044911.9	0.13806940E−03	0.55490060E+10
24	1465019	.0142135998	103071637	2.25535770E−03	0.57373450E+10
25	2807195	.0204564492	137227872	0.44523690E−03	0.63048900E+10
26	5387990	.0283794933	189855046	0.73751900E−03	0.73054240E+10
27	10358998	.0381757153	271350462	0.11680410E−02	0.88686300E+10
28	19945393	.0500248252	398709899	0.17773340E−02	0.11222780E+11
29	38458183	.0640949758	600018684	0.26095230E−02	0.14737250E+11
30	74248450	.0805164802	922152208	0.33185230E−01	—

Another example of making use of the linearity of squaring is a method referred to as weight manipulation [4] (for more details see [5]). Define $p(x)=x^m+g(x)$ where $g(x)$ has degree $k<<m$. Define d to be the largest integer such that $2^d m \le n$. Let $s=n-2^d m$ and consider

$$\ell(x)=x^s p(x)^{2^d} (mod\ f(x)).$$

It follows that

$$deg\ \ell(x)=max\ \{k2^d, deg\ (f(x)+x^n)\}.$$

If $p(x)$ and $\ell(x)$ are smooth with respect to b then we obtain an equation. For example, in $GF(2^{127})$ generated by $f(x)=x^{127}+x+1$ take $p(x)=1+x^7$ and $d=7$ then $s=15$. This gives

$$\ell(x)=x^{15}(1+x^7)^{16}=1+x+x^{15}.$$

Clearly, both $\ell(x)$ and $p(x)$ are smooth with respect to $b=17$. A similar result can be developed for

$(p(x))^{2^{d+1}}$.

Coppersmith [6] has extended the idea behind these techniques and has obtained striking improvements in the index calculus methods. We should mention that asymptotically the Waterloo algorithm is of the same order ($exp\ (c_2(n\ log\ n)^{\frac{1}{2}})$) as the Adleman algorithm but for fields $GF(2^n)$ of practical interest it gives much better running times. The Coppersmith algorithm is described in the next section.

The database for $GF(2^{127})$ was constructed at Denelcor using their HEP computer [12]. A HEP consists of from one to sixteen process execution modules and each is a pipelined processor with a depth of 8. Since the HEP can run a given program in parallel with itself, the index-calculus algorithms are ideally suited to this architecture. Several copies of the algorithm were run in parallel to produce linear equations which when added to those found systematically and by weight manipulation would yield 16,510 linearly independent equations in the 16,510 unknown logs. Sixteen copies of the algorithm in parallel appeared to be optimal. The database took about 7 hours to build and computing individual logs using Part 2 takes under 1 second on the HEP. Having the database our implementation on a VAX11/780 takes about five minutes per logarithm.

As mentioned in the previous paragraph, the index calculus algorithms are ideally suited to parallel processing. If we run n copies of the program in parallel then the logarithm is obtained from the copy which finishes first. We compute the expected number of iterations to find the logarithm of an element given that n copies of the algorithm are running simultaneously. Let p be the probability that an iteration will succeed in computing the logarithm and let $q=1-p$. Suppose i of the processes complete on the k^{th} iteration and the remaining $n-i$ do not. The probability of this event is

$$(pq^{k-1})^i\left[1-\sum_{r=1}^{k}pq^{r-1}\right]^{n-i}$$

and the probability that at least one of the processes finishes on the k^{th} iteration is

$$\sum_{i=1}^{n}\binom{n}{i}[pq^{k-1}]^i[1-\sum_{r=1}^{k}pq^{r-1}]^{n-i}. \qquad (*)$$

Since $p\sum_{r=1}^{k}q^{r-1}=1-q^k$, $(*)$ equals $q^{(k-1)n}[1-q^n]$. To compute the expected number of iterations we evaluate

$$\sum_{k=1}^{\infty}kq^{(k-1)n}[1-q^n]=(1-q^n)^{-1}.$$

Since $q=1-p$ then it follows that $(1-q^n)^{-1}\cong(np)^{-1}$ as one might expect. This expected number of iterations did occur when we made experimental runs on the HEP.

4. The Coppersmith Algorithm

The algorithm described in this section makes extensive use of the linearity of squaring in $GF(2^n)$. The method applies also to p^{th} powers in $GF(p^n)$ for $n>1$.

Coppersmith's modification [6] of this basic index-calculus algorithm of section 2 improves substantially the performance of both parts 1 and 2. Let's first consider part 1.

The algorithms of the previous sections and the algorithm given here rely on the fact that polynomials of degree k tend to factor into polynomials whose degrees are "small". In order to generate enough equations to construct the database for $GF(2^n)$ select pairs of polynomials $(a(x),b(x))$ such that $dega(x)\leq d$ and $degb(x)\leq d$ where $d<b$ and the gcd of $a(x)$ and $b(x)$ is 1. It is easily seen that there are precisely 2^{2d+1} such pairs. Let k be a power of 2 near $\sqrt{n/b}$ and choose h to be the least integer greater than $\frac{n}{k}$. Suppose also that the generating polynomial for the field has the form $f(x)=x^n+g(x)$ where $g(x)$ has low degree. Let $h(x)=x^{hk}=g(x)x^{hk-n}$ and define

$$c(x)=x^hA(x)+B(x)$$

and

$$d(x)=[c(x)]^k.$$

If both $c(x)$ and $d(x)$ are smooth with respect to b, then we obtain an equation for the database. In the case $n=127$ we take $b=17$, $k=4$, $h=32$ and $d=10$. The polynomials $c(x)$ and $d(x)$ have degrees ≤ 42. Coppersmith [6] shows that the 2 million possible pairs $(a(x),b(x))$ yield about 47,000 equations in the 16,510 unknowns. The above procedure is a variant on the weight manipulation method of the previous section. A more direct application of weight manipulation can be described. We illustrate the technique in the case $GF(2^{127})$ generated by $f(x)=x^{127}+x+1$. Select pairs of polynomials $(a(x),b(x))$ which are relatively prime and such that deg $a(x)\leq 7$ and deg $b(x)\leq 8$. Form the polynomials $c(x)=a(x)x^{31}+b(x)$ and $d(x)=x^3[c(x)]^4$ where deg $c(x)\leq 38$ and deg $d(x)\leq 35$. If $c(x)$ and $d(x)$ are smooth with respect to $b=17$ we get an equation.

We now describe part 2 of the algorithm. Let $g(x)$ be a field element whose log is to be found. Suppose $t=$deg $g(x)$. Let k be a power of 2 close to $\sqrt{n/t}$ and h be the least integer greater than n/k. For example, if $n=127$, $b=17$ and $t=33$ then choose $k=2$ so that $h=64$. Finally select d close to $(t+\sqrt{n/b})\log n)/2$. In our example we take $d=23$. Choose a relatively prime pair of polynomials $a(x)$ and $b(x)$ each of degree $\leq d$ such that $g(x)$ divides $c(x)=x^ha(x)+b(x)$. The choices for $a(x)$ and $b(x)$ can easily be determined by solving a linear system of equations over $GF(2)$. Let $d(x)=[c(x)]^k$. If both $c(x)$ and $d(x)$ are smooth with respect to the bound $b^{(1)}=h\sqrt{b/n}$ then we have at most $2t$ irreducible factors of degree at most $b^{(1)}$. For each factor with degree $>b$ we iterate this procedure and reduce the bound $b^{(1)}$ with each iteration. That is, at iteration i, $b^{(i)}=h(b/n)^{-2^i}$. If not both of $c(x)$, $d(x)$ are smooth then choose a new pair $a(x)$, $b(x)$ and start again.

The asymptotic running time of this algorithm is computed by Coppersmith [6] to be $exp\ (cn^{\frac{1}{3}}(log\ n)^{\frac{2}{3}})$ which improves the results of the previous two sections.

We have implemented part 2 of Coppersmith's algorithm. The following table displays test results of 8 samples each consisting of 100 randomly generated polynomials of degree at most 126. The degree bound b is always 17 but for each sample we varied the Coppersmith degree bound b' and the value of d. We have a column labelled "Total Number of Basic Iterations". This refers to the number of iterations required to obtain smoothness with respect to b'. The column labelled "Total Number of Coppersmith Iteration" is the total number of iterations to obtain smoothness with respect to degree bound 17 for those polynomials with degree between 17 and b'. We also found that in this case a single step reduction of the Coppersmith reduction was optimal.

b'	d'	Total number of basic iterations	Total number of Coppersmith iterations	Average time in sec per log
20	12	99,496	6,731	57
20	13	72,519	7,262	44
21	12	98,245	9,087	49
21	13	39,327	9,995	19
22	13	38,390	11,859	28
22	14	24,741	16,299	28
23	14	20,296	20,726	27
23	15	12,857	27,045	47

From the table it appears that $b'=23$ and $d=14$ is optimal. With these values we ran a sample of 500 randomly generated polynomials with the following result. The column headings are the same as in the previous table.

23	14	120,661	101,563	26.6

For $GF(2^{127})$ we refer to an equation of the form

$$[C(x)]^4 = [A(x)x^{32}+B(x)]^4$$

as a Coppersmith equation. As a direct generalization of the weight manipulation technique discussed earlier we consider equations of the form

$$x^l[C(x)]^4 = x^{127-4^l}[A(x)x^l+B(x)]^4$$

and refer to these as *underflow* equations. With the degree of $A(x)$ and $B(x)$ at most 9 in the Coppersmith equations we generated 19461 equations for the database in 8.1 hours on the VAX. With the degrees of $A(x)$ and $B(x)$ at most 8 we obtained 7777 equations in 2.23 hours. The following table lists results when underflow equations were used. The columns labelled *deg A* and *deg B* refer to the highest degree used for A and B respectively. For polynomial A we require that it has a constant term so that no Coppersmith equations are generated this way and the equations obtained from distinct rows are all different.

deg A	*deg B*	*i*	Value of Equations	Number of (Hours)
8	9	31	7891	2.89
9	8	30	7193	2.88
10	7	29	5975	2.80

If we use the Coppersmith equations with the degrees of $A(x)$ and $B(x)$ at most 8 and we use the underflow equations given in the first two rows of the table we get 22,861 equations in 8 hours or one equation every $.3499\times10^{-3}$ hours. This is approximately a 16% improvement over collecting the 19461 equations using the Coppersmith equations with *deg* $A(x)$, and *deg* $B(x)\leq 9$. This data was obtained by running our program on a VAX11/780 at the University of Waterloo. The program is written in Fortran 77 with the gcd routines in assembler.

5. Conclusion

We have described the basic index-calculus algorithm and two variants of it. The basic algorithm and the Waterloo algorithm both carry over to $GF(p)$, p a prime. Using the Euclidean algorithm for integers it is easy to show that given an integer a, $1\leq a\leq p-1$, that there exist integers s and t such that $as\equiv t(mod\ p)$ and $1\leq|s|$, $t\leq p-1$.

As for $GF(2^n)$ it is clear that for $n=127$ this field is insecure and should be avoided in cryptographic schemes. In [13] Odlyzko analyses the performance of the Coppersmith algorithm for various values of n running on various types of equipment. The conclusion seems to be that an ambitious effort might be able to produce the necessary database for $n\leq 1280$.

References

[1] L.M. Adleman, A subexponential algorithm for the discrete logarithm problem with applications to cryptography, *Proc. 20th IEEE Found. Comp. Sci. Symp.* (1979), 55-60.

[2] B. Arazi, Sequences constructed by operations modulo 2^n-1 or mod 2^n and their application in evaluating the complexity of a log operation over $GF(2^n)$, preprint.

[3] I.F. Blake, R. Fuji-Hara, R.C. Mullin and S.A. Vanstone, Computing logarithms in finite fields of characteristic two, *SIAM J. Alg. Disc. Methods*, Vol. 5 #2 (1984), 276-285.

[4] I.F. Blake, R. Fuji-Hara, R.C. Mullin and S.A. Vanstone, Finite field-techniques for shift registers with applications to ranging problems and cryptography, Final Report Project #106-16-02, Department of Communications (1983).

[5] I.F. Blake, R. Fuji-Hara, R.C. Mullin and S.A. Vanstone, An attack on the discrete logarithm problem in $GF(2^{127})$, Progress Report, Project #106-16-02, Department of Communications (1982).

[6] D. Coppersmith, Fast evaluation of logarithms in fields of characteristic two, *IEEE Trans. Inform. Theory*, (July 1984), 587-594.

[7] W. Diffie and M.E. Hellman, New directions in cryptography, *IEEE Trans. Inform. Theory*, IT-22 (1976), 644-654.

[8] T. ElGamel, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Inform. Theory*, to appear.

[9] T. Herlestam and R. Johanneson, On computing logarithms over $GF(2^p)$, *BIT* 21 (1981), 326-334.

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

[11] D.L. Long and A. Wigderson, How discreet is the discrete log? *Proc. 15th ACM Symp. Theory of Computing* (1983), 413-420.

[12] R.C. Mullin, E. Nemeth and N. Weidenhofer, Will public key crypto systems live up to their expectations? HEP implementation of the discrete log codebreaker, preprint.

[13] A. Odlyzko, Discrete logarithms in finite fields and their cryptographic significance, *Eurocrypt-84* (to appear).

[14] S.C. Pohlig and M. Hellman, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Trans. Inform. Theory* IT-24 (1978), 106-110.

[15] B.P. Schanning, Data encryption with public key distribution, *EASCON Conf. Rec.*, Washington D.C., October 1979, 653-660.

[16] A.E. Western and J.C.P. Miller, Tables of indices and primitive roots, *Royal Society Mathematical Tables*, Cambridge University 9 (1968).

[17] K. Yiu and K. Peterson, A single-chip VLSI implementation of the discrete exponential public key distribution system, *Proc. GLOBECOM-82, IEEE* (1982), 173-179.