



University of  
Waterloo Research Institute

## Finite Field Techniques for Shift Registers and Encipherment

Final Report  
Project No. 106-16

DSS 235U.36001-1-1794

Prepared for  
The Department of Communications  
under DSS Contract No. OSU 81-00224

by

R. Fuji-Hara, S. Vanstone and I. Blake  
University of Waterloo

LKC  
P  
91  
.C655  
F83  
1983

IC

Scientific Authority  
B. Bryden  
Communications Research Centre



Government  
of Canada

Gouvernement  
du Canada

## MEMORANDUM

TO  
A

Distribution

FROM  
DE

J.L. Pearce

SUBJECT  
OBJET

re the attached Research Contract Report

## NOTE DE SERVICE

SECURITY - CLASSIFICATION - DE SÉCURITÉ

OUR FILE/NOTRE RÉFÉRENCE

CRC/DSS 8020-1

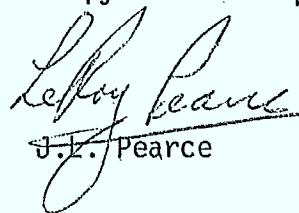
YOUR FILE/VOTRE RÉFÉRENCE

DATE

7 June 1983

Please find attached a copy of a report on the advances which have been made at the University of Waterloo in the area of Finite Field Mathematics. The front page is a short summary by B. Bryden, the contract Scientific Authority, of the application of the work. This work must be considered as state-of-the-art in this area of mathematics. The results are of significant value to the control of spread spectrum modem code generators, ranging and in the area of cryptoanalysis as it might be applied against some of the commercial grade privacy systems now becoming available. This research work has been jointly sponsored by DND and DOC.

If you do not wish to retain a copy of this report please return to the undersigned.



J.L. Pearce

JLP:jw

Dist'n

A.R. Abercrombie, TLO  
T. Davies, CGP  
A. Stewards, DREO  
E.B. Felstead, CRC  
M.A. MacArthur, CSE  
L. Baxter, CSE for CSE library  
Maj S. Zolmer, DTA(CE)  
Maj R. Bacon, DCEM  
R.J. Campbell, CRC  
circ to DGSTA and Directors  
circ to DSS professionals  
CRC library  
DOC library

P  
91

C655

F83

1983

S-Gen

## Executive Summary

by B. Bryden, CRC

CDMM

DOC-CR-SP-83-030

JULY 26 1983

LIBRARY - BIBLIOTHÈQUE

WITH Industry Canada  
Library - Queen

NOV 28 2013

Industrie Canada  
Bibliothèque - Queen

### The Problem

Given a shift register which is connected in a feedback fashion to generate a string of seemingly random numbers, it is very straightforward to calculate the state of the shift register (each stage being 0 or 1) given an initial state and a number of shifts or clock pulses applied to the shift register. It has, until now, been considered very difficult, given two states of the shift register, to determine the number of clock pulses that have been applied to the shift register in getting from one state to the other, especially for long shift registers.

### The application

One requirement for knowing the number of clock pulses between two states of a shift register can be shown using the following conceptual experiment. Suppose that a shift register sequence is being generated at a ground station and sent by radio to a deep space probe. The probe receives this signal and re-transmits it back to the ground station. If the ground station compares the received sequence to its locally generated sequence, and can deduce the number of clock pulses separating the two sequences, it can calculate the distance between the ground station and the deep space probe.

The application to cryptography can be shown by comparing the amount of work required to find the number of clock pulses between two shift register states (large) to the amount of work required to calculate the shift register state after a certain number of clock pulses (small). One gives the difficult job to the cryptanalyst and the easy job to the intended recipient.

### The solution

If the difficult problem, known as the discrete logarithm problem, can be made relatively easy, it is good news for people who design deep space probes and bad news for cryptographers. The key to the problem lies in examining the differences between the two shift register states, which we will call a polynomial. If the polynomial can be factored into the products of two smaller polynomials, each a discrete exponential, then the problem is made easier. An algorithm known as the Euclidean algorithm suggests that such a factorization probably exists. If we first make a table of all polynomials much smaller in size than the original polynomial, chances are that we will be able by trial and error to factor the unknown polynomial into smaller parts, all of which are in the table. Since we know the exponent of each entry in the table, we can sum the exponents of the factored parts to obtain the exponent of the original polynomial. This report shows a procedure for making the required table and factoring large polynomials of degree 127.

Declassified for Bryden 07/2000

Finite Field Techniques for Shift  
Registers and Encipherment

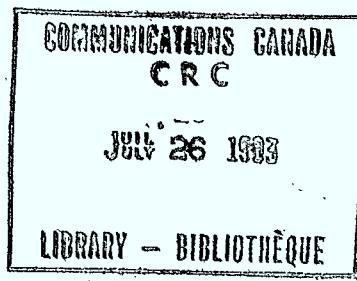
Final Report  
Project No. 106-16

Prepared for  
The Department of Communications  
under DSS Contract No. OSU 81-00224

by

R. Fuji-Hara, S. Vanstone and I. Blake  
University of Waterloo

Scientific Authority  
B. Bryden  
Communications Research Centre



## Table of Contents

1. Introduction	1
2. The Swedish Algorithm	4
3. The New Algorithm	6
4. Experimental Results	17
5. Comments on the Algorithm	22
References	23
Appendix	24

### 1. Introduction.

The problem of determining logarithms in a finite field has two important applications of interest here. For the first application, given two states of a linear feedback shift register, the problem of determining the number of clock cycles that has elapsed between them is equivalent to determining logarithms in the appropriate finite field. This equivalence has been demonstrated in a previous report [1]. The second application of interest is in public key cryptography. Specifically [2], for the finite field  $GF(q)$ , user  $i$  stores in a public file the key  $y_i = \alpha^{x_i}$  where  $\alpha$  is a fixed primitive element of  $GF(q)$ . If users  $i$  and  $j$  wish to communicate, they use the key  $K_{ij} = \alpha^{x_i x_j}$ . The security of the system depends on the difficulty of determining  $x_i$  from a representation of  $\alpha^{x_i}$  and the fact there is no apparent method for obtaining  $K_{ij}$  from a knowledge of  $y_i$  and  $y_j$ . A system utilizing this scheme has apparently been implemented at MIT [3].

Thus, it is of some interest to consider possible approaches to the determination of logarithms in a finite field. It has been observed ([4], [5]) that if  $q-1$  is highly composite, where  $q$  is the order of the field, then the complexity of finding logarithms is greatly reduced. It is also of interest to restrict attention to fields of characteristic 2. Thus from a security point of view, fields of the order  $2^p$ , where  $2^p-1$  is a Mersenne prime, are of interest. The MIT system mentioned earlier utilizes  $GF(2^{127})$  and  $2^{127}-1$  is a Mersenne prime.

The best known algorithm for finding logarithms in  $GF(q)$ , where  $q$  is a prime number, is of the order  $q^{1/2}$  [4], [6]. From the approach taken in this report one is tempted in one sense to conjecture a similar

result for  $GF(2^P)$  when  $2^P - 1$  is a Mersenne prime. This does not mean however that efficient algorithms for finding logarithms will not exist for specific values of  $p$ . These algorithms may exploit the explicit properties of the particular field. Examples of this are the Swedish algorithm [7] described briefly in the next section and the new algorithm presented in section 3.

As a final comment for this section it is noted that, if a good algorithm for computing logarithms with respect to one base is known, it is of interest to use this algorithm to compute logarithms with respect to another. Specifically, let  $\alpha$  be a primitive element for a field  $F_1$  of order  $2^P$  and let  $\beta$  be a primitive element for a field  $F_2$  of order  $2^P$ . Suppose that a good algorithm for computing logarithms to the base  $\alpha$  in  $F_1$  is known. Given an element  $x \in F_2$  such that we require its logarithm to the base  $\beta$  we could do the following: Let  $M_\beta(x)$  be the minimum polynomial of  $\beta$  and let  $\alpha^i$  be a root of  $M_\beta(x)$  in  $F_1$ . This polynomial can be used to define a mapping.

$$f: F_2 \rightarrow F_1 \text{ where } f(\beta^j) = (\alpha^i)^j, \quad 0 \leq j \leq 2^P - 2.$$

$f$  is a field isomorphism and can be represented by a linear transformation  $A$  between the bases  $B_1 = \{1, \beta, \beta^2, \dots, \beta^{P-1}\}$  and  $B_2 = \{1, \alpha, \alpha^2, \dots, \alpha^{P-1}\}$ . Now, for  $x \in F_2$ , write  $x$  with respect to the basis  $B_1$  and denote it  $x_\beta$ .

Thus,  $Ax_\beta \in F_1$ . Computing the logarithm of this element gives

$$Ax_\beta = \alpha^t \text{ from some integer } t.$$

or

$$x_\beta = A^{-1} \alpha^t.$$

Since  $A$  is an isomorphism so is  $A^{-1}$  and, hence,

$$x_\beta = (A^{-1} \alpha)^t$$

$$= \beta^{-it}.$$

Therefore, the log of  $x$  is readily obtained given a good algorithm for finding logs to the base  $\alpha$  in  $F_1$  exists.

## 2. The Swedish Algorithm

For later comparisons, a brief description of the algorithm of Herlestam and Johannesson [7] is given, hereafter referred to as the Swedish algorithm. It is convenient to represent elements of  $GF(2^p)$  as polynomials of degree at most  $p-1$ , the equivalence between binary  $p$ -tuples and polynomials being immediate. If the basis is  $\{1, \alpha, \alpha^2, \dots, \alpha^{p-1}\}$  and  $\alpha$  is a root of the polynomial  $g(x)$ , then two operations are defined:

$$S: f(x) \rightarrow f^2(x) = f(x^2) \bmod g(x)$$

$$T: f(x) \rightarrow xf(x) \bmod g(x).$$

The Hamming weight of  $f(x)$  is the number of its nonzero coefficients. To find the logarithm of  $f(x)$  the algorithm computes the  $p^2$  elements

$$T^{-2^r} S^s f(x) \quad 0 \leq r, s \leq p-1.$$

and chooses an element of this set of lowest Hamming weight. If such an element is

$$f'(x) = T^{-2^{r'}} S^{s'} f(x) \bmod g(x)$$

then

$$\log f'(x) = -2^{r'} + 2^{s'} \log f(x)$$

The operation is repeated until an element of Hamming weight either 1, 2 or 3 is obtained, depending on the amount of storage used. Variations of the algorithm are possible.

The algorithm is simple and elegant. Unfortunately it is difficult to analyze in terms of running time and it is not even clear that it will always terminate. For example for  $p=17$ , applying the previous transformations to the element

01 00111 01111 01010

produces an element of Hamming weight 4

1 000 1100 0000 00010

and further applications fail to reduce the Hamming weight (the basis used is  $\{1, \alpha, \alpha^2, \dots, \alpha^{16}\}$  and  $1 + \alpha^3 + \alpha^{17} = 0$ ). While it may be possible to overcome this difficulty, other difficulties in analyzing the algorithm remain.

It is of interest to continue the search for good algorithms for particular Mersenne primes. An algorithm for  $GF(2^{31})$  will be given in the next section which is guaranteed to converge and, in terms of programming on a general purpose computer, is considerably faster than the Swedish algorithm. The algorithm is efficient for computing logarithms in  $GF(2^p)$ ,  $p \leq 31$ .

### 3. The New Algorithm

The case of particular interest in this report is the finite field  $GF(2^{31})$  for which  $\alpha$  is a root of the primitive polynomial  $g(x) = 1 + x^3 + x^{31}$ . However, many of the properties discussed are true for the general case. In general  $\alpha$  will be a fixed root of primitive polynomial  $g(x)$  of degree  $n$  and the elements of  $GF(2^n)$  will variously be viewed as binary  $n$ -tuples (with respect to the basis  $\{1, \alpha, \alpha^2, \dots, \alpha^{n-1}\}$ ) and polynomials of degree at most  $(n-1)$ , the equivalence being clear.

Before discussing particulars of the algorithm a certain property of the Euclidean algorithm is mentioned which tends to support the view that, in general, the complexity of finding logarithms in a finite field  $GF(q)$  is of the order  $q^{1/2}$  when  $q-1$  is prime, as discussed earlier. As part of a proof of some properties of the Euclidean algorithm it is shown in [8] that, for two polynomials  $f(x)$  and  $g(x)$  and for two nonnegative integers  $\mu$  and  $v$ ,  $v \geq \deg(\gcd(f, g))$ ,  $\mu+v = \deg(g)-1$ , there exist polynomials  $t(x)$ ,  $r(x)$ ,  $\deg(t) \leq \mu$ ,  $\deg(r) \leq v$  such that

$$t(x)f(x) \equiv r(x) \pmod{g(x)}.$$

Thus if it is desired to find the logarithm of  $f(x)$  in  $GF(2^n)$ , choosing  $\mu = v = (n-1)/2$ ,  $\deg(g) = n$ , the implication is that the problem can be reduced to determining the logarithms of  $t(x)$  and  $r(x)$ , each of which by direct search would require on the order of  $2^{(n-1)/2}$  operations, and hence the result. It will not be convenient to use this approach for the new algorithm but it is an interesting point which may prove useful in future work.

The idea of the algorithm is to choose a subset  $H$  of polynomials, whose logarithms are predetermined, and an algorithm to decompose an

arbitrary polynomial  $f(x)$  in terms of products of powers of polynomials

$$f(x) = h_1(x)^{i_1} h_2(x)^{i_2} \dots h_k(x)^{i_k} \bmod g(x), \quad h_i(x) \in H.$$

The logarithm of  $f(x)$  is then easily calculated as

$$\log f(x) = \sum_{i=1}^k i_1 \log(h_i(x))$$

This is essentially the idea behind the Swedish algorithm as well but the techniques used here are unrelated to that algorithm. The problem is to determine a convenient set  $H$ , as small as possible, and an efficient decompositon algorithm with guaranteed convergence. Rather than state the algorithm immediately and prove the properties claimed for it, it will be simpler to trace the development of the algorithm.

In the first instance the set  $H$  is decomposed into two sets

$H_1, H_2$ :

$$H_1 = \{ f(x), \deg(f) \leq s < \left\lfloor \frac{n}{2} \right\rfloor \}$$

$$H_2 = \{ f(x), s < \deg f \leq \left\lfloor \frac{n}{2} \right\rfloor, f \text{ irreducible} \}.$$

The cardinality of  $H_1$  is  $2^s$  and the cardinality of  $H_2$

$$|H_2| = \sum_{i=s+1}^{\left\lfloor \frac{n}{2} \right\rfloor} \frac{1}{i} \sum_{d|i} \mu(d) 2^{i/d}$$

where  $\mu(\cdot)$  is the Mobius function. For the sequel  $s$  will be chosen as 8 for  $n=31$ . Using the property of the Euclidean algorithm mentioned, to find the logarithm of  $f(x) \in GF(2^n)$ , the problem is first reduced to finding the logarithm of two polynomials, each of degree less than  $n/2$ . Each of these polynomials is then factored into its factors using standard

techniques [9] until each factor is either irreducible or of degree less than 8. At this point the known logarithms of the polynomials of  $H$  can be used to determine the logarithm of  $f(x)$ . For  $n=15$ ,  $s=8$  the number of polynomials in  $H$  is approximately 6,000. While this number is manageable it is still large and much of the remainder of the report is concerned with modifying the set  $H_2$  and the decomposition algorithm. To do this certain notation and properties of polynomials are required.

If  $f(x) = f_0 + f_1x + \dots + f_{n-1}x^{n-1} \in GF(2^n)$  and  $f_0 = f_1 = \dots = f_{i-1} = 0$ ,  $f_i = 1$ , then  $f'(x) = x^{-1}f(x)$  is referred to as the standardized polynomial of  $f(x)$  and  $\deg(f'(x)) = \deg(f(x))-i$ . Suppose now that  $m$  consecutive coefficients of  $f(x)$ , a standardized polynomial of degree  $k$ , are zero i.e.  $f_i = f_{i+1} = \dots = f_{i+m-1} = 0$ . For such a polynomial the following two properties are true.

Property 1. If  $i + (n-k-1) \geq \deg(g(x)-x^n)$  then  $\deg(x^{n-1-m}f(x)) \leq n-1-m$

Property 2. If  $i + (n-k-1) \geq \deg(g(x)-x^n)$  and  $m \geq n-1-k$  then  $\deg(x^{n-1-m}f(x)) \leq k - \deg(f(x))$ .

The proofs are straight forward and we omit them.

A further useful property results in noting that if the weight (number of nonzero coefficients) of  $f(x)$  is even then  $f(1) = 0$  and  $(1+x) | f(x)$ ,  $f(x) = (1+x)h(x)$ ,  $\deg(h(x)) = \deg(f(x)) - 1$ .

The following two theorems will play a central role in the algorithm.

Theorem 1. Any standardized polynomial  $f(x)$  in  $GF(2^n)$  can be written as

$$f(x) = x^i f_1(x) f_2^{-1}(x) \pmod{g(x)}$$

where  $\deg(f_1(x)) = \deg(f(x))-1$ ,  $\deg(f_2(x)) \leq n-\deg(f(x))$  and  $i = n-1-\deg(f(x))$ .

Proof. Assume that  $f(x)$  is of degree  $k$ ,  $f(x) = \sum_{i=0}^k c_i x^i$  and let

$$h_1(x) = h_{n-k+1}^{(1)} x^{n-k+1} + h_{n-k+2}^{(1)} x^{n-k+2} + \dots + h_{n-1}^{(1)} x^{n-1}$$

$$h_2(x) = x^{-1} + h_0^{(2)} + h_1^{(2)} x + \dots + h_{n-k-1}^{(2)} x^{n-k-1}$$

Consider the product

$$f(x)h_2(x) = x^{-1}f(x) + h_0^{(2)}f(x) + h_1^{(2)}xf(x) + \dots + h_{n-k-1}^{(2)}x^{n-k-1}f(x)$$

and notice that by correctly choosing the coefficients of  $h_2(x)$ , the first  $(n-k-1)$  coefficients of  $f(x)h_2(x)$  can be made zero. This is possible since  $f(x)$  is a standardized polynomial. It follows immediately that if  $f_1(x) = x^{-(n-k)}h_1(x)$  and  $f_2(x) = xh_2(x)$  then  $\deg(f_1(x)) = k-1$ ,  $\deg(f_2(x)) \leq n-k$  and

$$f(x) = x^{n-k+1} f_1(x) f_2(x)^{-1} \pmod{g(x)}$$

as claimed. Notice that  $h_1(x)$  may not be a standardized polynomial in which case the equation can be modified in a straight forward manner.

The next theorem is a variation of the one above and although it is very similar in appearance it is not equivalent.

Theorem 2. Any standardized polynomial  $f(x)$  of degree  $k$  can be expressed as

$$f(x) \equiv x^i f_1(x) f_2(x)^{-1} \pmod{g(x)}$$

where  $\deg(f_1(x)) = \deg(f(x))$ ,  $\deg(f_2(x)) \leq n-k-1$  and  $i = n-k$ .

Proof. As in the previous theorem, it is possible to choose the coefficients of a polynomial  $h_2(x)$  of degree  $(n-k-1)$  such that the polynomial

$$(x^n + f(x)) + f(x)h_2(x) = h_1(x) \pmod{g(x)}$$

has the first  $(n-k)$  coefficients zero. Rearranging this equation as

$$f(x)(1+h_2(x)) = x^n + h_1(x) \pmod{g(x)}$$

Defining  $f_2(x)$  as  $1+h_2(x)$  it follows that  $\deg(f_2(x)) \leq n-k$ . Similarly if  $f_1(x) = x^{-(n-k)}(x^n + h_1(x))$ , then  $\deg(f_1(x)) = k$  although it may not be in standardized form. Rewriting the equation yields

$$f(x) \equiv x^{n-k} f_1(x) f_2(x)^{-1} \pmod{g(x)}$$

as required.

While the decompositions of theorems 1 and 2 sometimes give the same result, they are not equivalent.

Some operations used in the algorithm are now described. To find the logarithm of an arbitrary polynomial  $f(x) \in GF(2^n)$  the algorithm will construct a list of polynomials  $f_1(x), f_2(x), \dots, f_u(x)$ , and a list of integers  $e_1, e_2, \dots, e_u$  and two other integers A and B such that

$$f(x) = x^A (1+x)^B f_1(x)^{e_1} f_2(x)^{e_2} \dots f_u(x)^{e_u}$$

For reference, the list of polynomials is designated L and the list of integers E. Initially the list L contains the one polynomial  $f(x)$  whose logarithm is required:  $L=f_1, E=e_1, u=1$ . The following operations are used repeatedly in the algorithm on  $f_t$ , the polynomial on top of the list L:

$P_1(f_t)$ : using the three properties mentioned at the beginning of the section  $f_t(x)$  is expressed as  $x^a(1+x)^b h(x)$  where  $h(x)$  is in standardized form. The polynomial  $f_t$  is replaced by  $h$  and the integers A and B augmented,  $A \leftarrow A+a, B \leftarrow B+b$ .

$P_2(f_t)$ : using theorem 1  $f_t(x)$  is decomposed as  

$$f_t(x) \stackrel{=} {x^a h_1(x) h_2(x)^{-1}}$$

The polynomial  $f_t(x)$  is replaced by  $h_1(x)$ ,  $h_2(x)$  is added to the bottom of the list L and  $u \leftarrow u+1$ ,  $f_u \leftarrow h_2(x)$ ,  $e_u \leftarrow -1$ ,  $A \leftarrow A+a$

$P_3(f_t)$ : using theorem 2  $f_t(x)$  is decomposed as

$$f_t(x) = x^a h_1(x) h_2(x)^{-1}$$

The polynomial  $f_t(x)$  is replaced by  $h_1(x)$ ,  $h_2(x)$  is added to the bottom of list L and  $u \leftarrow u+1$ ,  $f_u \leftarrow h_2$ ,  $e_u \leftarrow -1$ ,  $A \leftarrow A+a$ .

$P_4(f_t)$ : (i) If  $f_t(x)$  is irreducible, pass to the next step of the algorithm.

(ii) If  $f_t(x)$  is reducible, factorize  $f_t(x)$  using standard techniques [9] into one of two forms:

a)  $f_t(x) = h_1(x)h_2(x)$ : replace  $f_t(x)$  by  $h_1(x)$  and add  $h_2(x)$  to L,  $u \leftarrow u+1$ ,  $f_u(x) \leftarrow h_2(x)$ ,  $e_u \leftarrow 1$ .

b)  $f_t(x) = h(x)^c$ : replace  $f_t(x)$  by  $h(x)$  and  $e_t$  by  $c \cdot e_t$ .

Three versions of the algorithm will be described, each succeeding one a refinement of the previous one. They will only be discussed for  $n=31$ .

Version I. This first version is essentially the one described earlier in the section. It decomposes  $f(x)$  as

$$f(x) = x^A (1+x)^B f_1(x)^{e_1} f_2(x)^{e_2} \dots f_u(x)^{e_u}$$

where either a)  $\deg(f_i(x)) \leq 8$  or b)  $9 \leq \deg(f_i(x)) \leq 15$  and  $f_i(x)$  is irreducible,  $1 \leq i \leq u$ . The algorithm is as shown in figure 1 with the exception that the block B1 is not used; the output (1) is connected directly to input (2).

Example 1.  $f(x) = 10001000101101010101111011111111$

The decompositon proceeds as:

$f_1(x) = 111\ 111\ 01$	$e_1 = 1$	$\deg(f_1(x)) = 7$
$f_2(x) = 111$	$e_2 = -1$	$\deg(f_2(x)) = 2$
$f_3(x) = 11001$	$e_3 = 1$	$\deg(f_3(x)) = 4$
$f_4(x) = 10010111101011$	$e_4 = 1$	$\deg(f_4(x)) = 13$
$A = -17,$	$B = 2$	

Thus

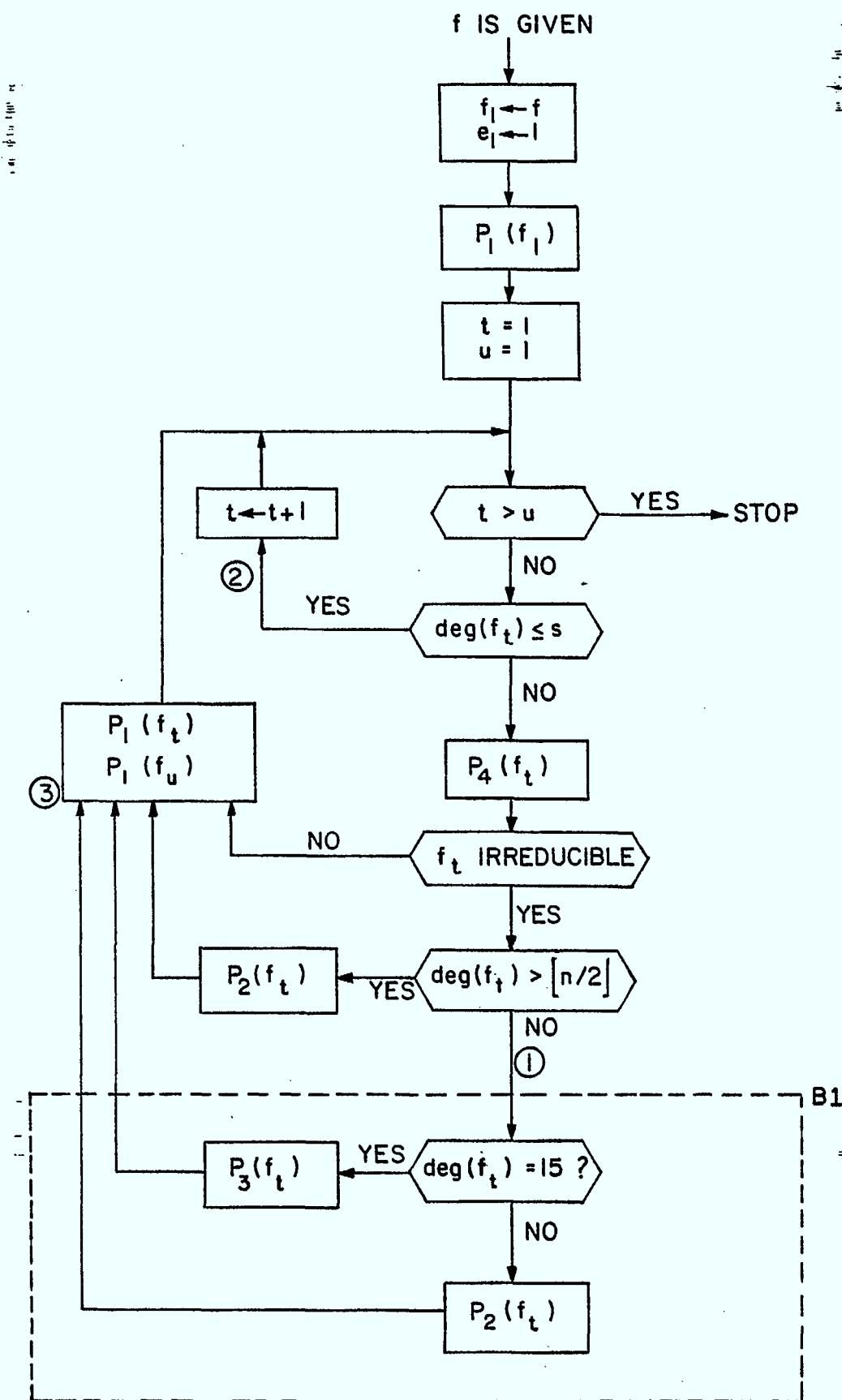
$$f(x) = x^{-17} (1+x)^2 f_1(x) f_2(x)^{-1} f_3(x) f_4(x)$$

Since  $f_1(x), f_2(x), f_3(x) \in H_1, f_4(x) \in H_2$  ( $H_1, H_2$  as defined earlier), the logarithm is easily computed.

As indicated before, the set  $H = H_1 \cup H_2$  contains approximately 6,000 polynomials whose logarithms are precomputed. This precomputation is a relatively simple matter but was not done because of the expense. The next two versions of the algorithm introduce techniques to reduce the size of  $H_2$ .

Version II. As a first step in this version, when an irreducible polynomial  $f_t(x), s < \deg(f_t(x)) \leq \left\lfloor \frac{n}{2} \right\rfloor$ , is encountered a further decomposition by the operations  $P_2(\cdot)$  and  $P_3(\cdot)$  is attempted. Of course a decomposition of an irreducible polynomial into a product of other polynomials is only possible modulo  $(g(x))$  and if the sum of the degrees of the constituent polynomials exceeds the degree of  $g(x)$ . It is not clear that the algorithm will converge in a finite number of steps to a product of polynomials, each of degree not greater than  $s$  although experimental evidence shows that it usually does. If it always did then the set  $H_2$  could be eliminated entirely, but this is not the case. The procedure is demonstrated in the following example.

Figure 1.



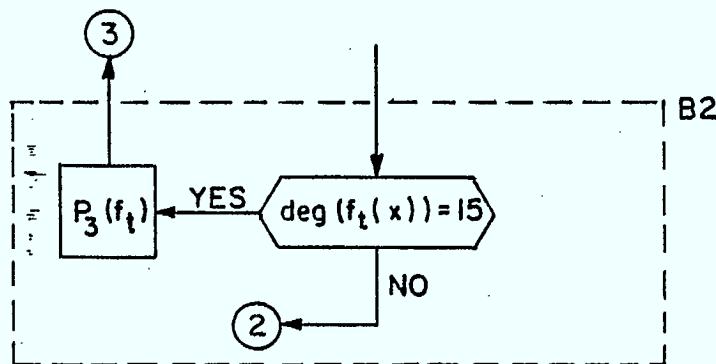
Example 2. The polynomial  $f(x)$  is as in the previous example

$f_1(x) = 11\ 1111\ 01$	$e_1 = 1$	$\deg(f_1(x)) = 7$
$f_2(x) = 111$	$e_2 = -1$	$\deg(f_2(x)) = 2$
$f_3(x) = 11001$	$e_3 = 1$	$\deg(f_3(x)) = 4$
$f_4(x) = 1101001001$	$e_4 = 1$	$\deg(f_4(x)) = 9$
$f_5(x) = 101\ 0111$	$e_5 = -1$	$\deg(f_5(x)) = 6$
$f_6(x) = 110000111$	$e_6 = -1$	$\deg(f_6(x)) = 8$
A = 5,	B = 1	

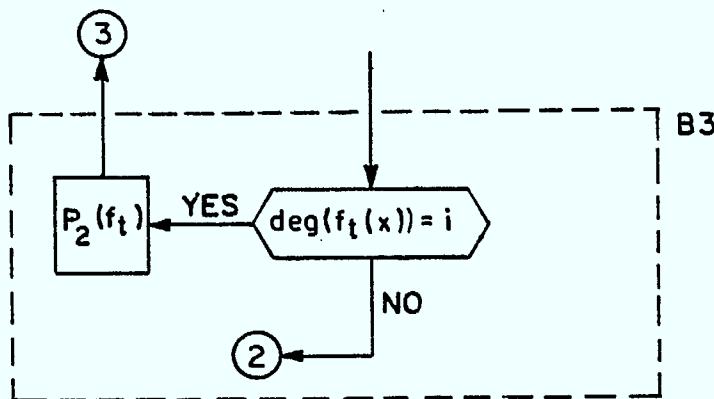
Comparing these results with those of example 1, it is seen that the polynomial of degree 13 has disappeared and polynomials of degrees 9, 6 and 8 respectively, have appeared. Although all the polynomials in this particular decomposition are in  $H_1$ , this will not always be the case.

The example does indicate however that, by operating on the irreducible polynomials it may be possible to eliminate the need to store many of them in the set  $H_2$ . This observation motivated the next step of the algorithm, which was carried out only for the case  $n=31$ ,  $s=8$  and  $g(x) = 1+x^3+x^{31}$ .

Let  $F_k$  denote the set of irreducible polynomials of degree  $k$ ,  $H_2 = F_{10} \cup F_{11} \cup \dots \cup F_{15}$ . The algorithm of figure 1 is run first for each polynomial in  $F_{15}$  with that part of the algorithm in box B1 replaced by that shown in box B2 below.



For each polynomial in  $F_{15}$  the algorithm is allowed to run until either a maximum of 20 polynomials accumulates in the list or the algorithm terminates with all polynomials in the list of degree not greater than 9. If, for a given irreducible polynomial, the limit of  $n=20$  is reached, that polynomial is stored in the set  $F_{15}^*$ . This experiment was repeated for the sets  $F_i^*$ ,  $i = 10, 11, \dots, 14$  with the box B2 above replaced by B3 below.  $F_i^*$ ,  $i = 10, 11, \dots, 14$  contains those polynomials of degree  $i$  which could not be decomposed into 20 or fewer polynomials of degree  $\leq 15$  and degree not equal to  $i$ . We note that this algorithm is not applied to  $F_9$  since a second pass is required and  $F_9$  will be computed at that time.



Notice that if, for a given irreducible polynomial  $f(x)$  of degree  $i$ , the algorithm terminates before  $n=20$  then that polynomial can be expressed as the product of fewer than 20 polynomials, each component being of degree 8 or less, or irreducible of degree between 10 and 15 but not of degree  $i$ . The sizes of the sets  $F_i^*$  which resulted are:

i	$ F_i $	$ F_i^* $	$ F_i^* / F_i $
15	2182	156	.0715
14	1161	241	.2076
13	630	135	.2143
12	335	94	.2806
11	186	40	.2151
10	<u>99</u>	<u>23</u>	.2323
	4593	689	

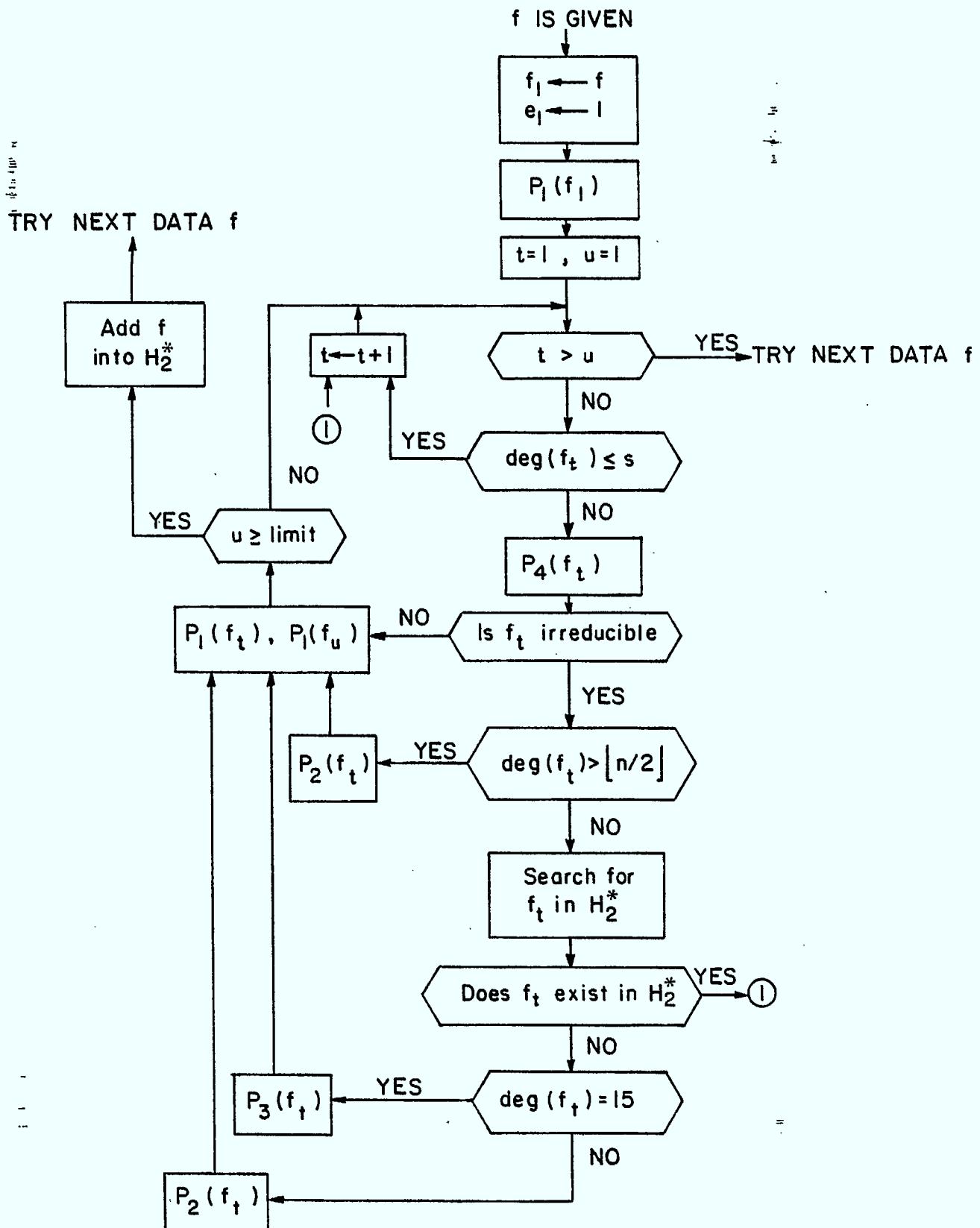
Unfortunately we are not yet finished. If a polynomial  $f(x) \in F_i \setminus F_i^*$  requires a polynomial in  $F_j \setminus F_j^*$ ,  $i \neq j$ , and this polynomial requires a polynomial in  $F_k \setminus F_k^*$  then infinite loops can arise in the decomposition process. The sets of polynomials  $F_i^*, i=10,11,\dots,15$  must therefore be enlarged to eliminate this possibility.

The algorithm by which this is accomplished is shown in figure 2. The set of input polynomials to this algorithm is  $F_9 \cup F_{10} \cup \dots \cup F_{14}$  and, initially, the set  $H_2^* = F_{10}^* \cup F_{11}^* \cup \dots \cup F_{15}^*$ . Let  $F_i^{**}$ ,  $i=9,10,\dots,15$  be the set of resulting polynomials in  $H_2^*$ , of degree  $i$  when the algorithm is finished. We note that during this pass we do not need to compute  $F_{15}$  as input polynomials and that  $F_{15}^{**} = F_{15}^*$ . Then the sizes of these sets was found to be:

i	$ F_i $	$ F_i^* $	$ F_i^{**} $
15	2182	156	156
14	1161	241	268
13	630	135	216
12	335	94	164
11	186	40	106
10	99	23	80
9	56		<u>49</u>

Figure 2.

15a.



The final set  $H_2^*$  thus contains 1039 polynomials. The set of all polynomials of degree less than or equal to 8 is of size 256 and all together 1295 polynomials and their logarithms must be stored,  $H = H_1 \cup H_2^*$ . The polynomials of  $H_2^*$ , but not their logarithms, are listed in the appendix.

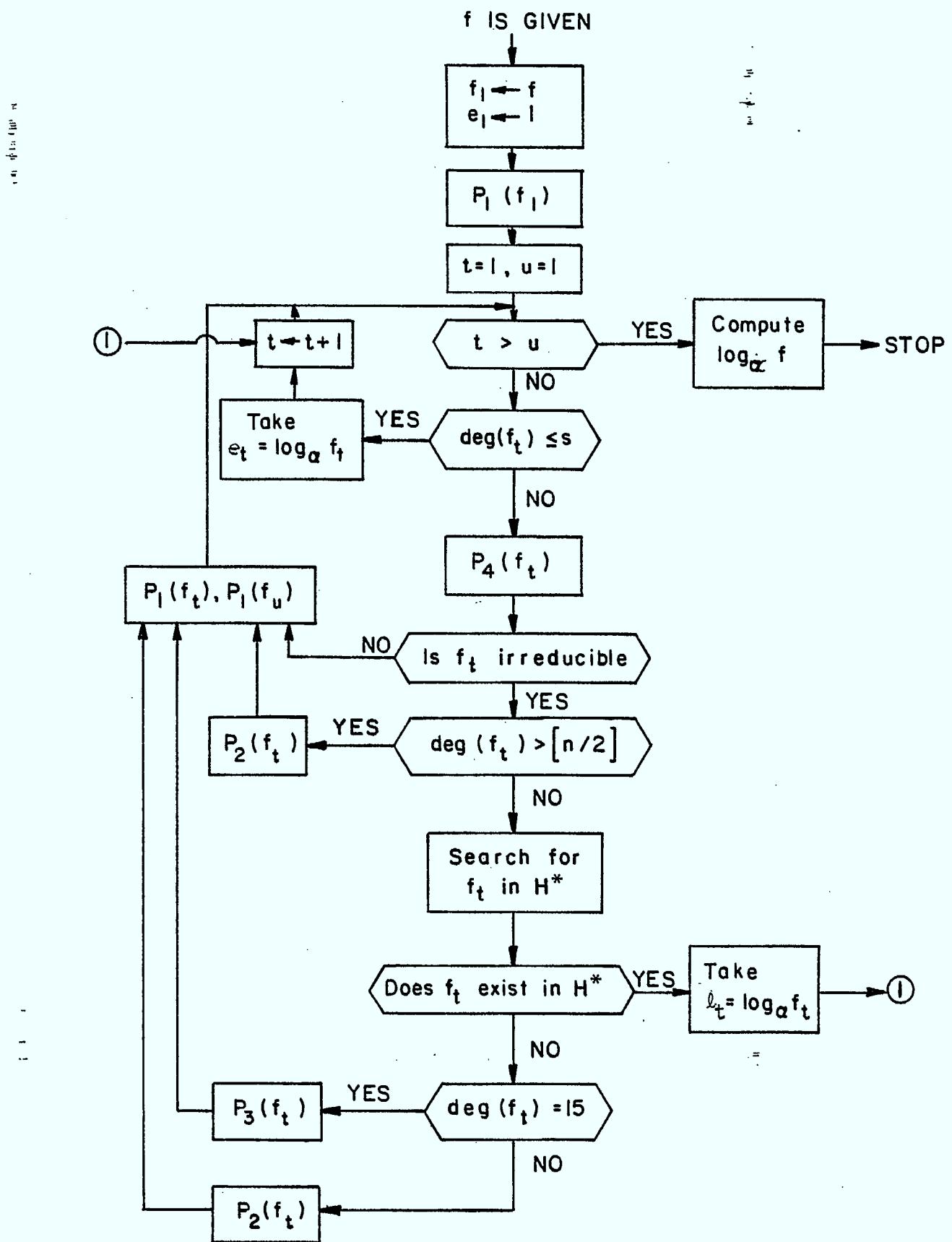
The final form of the algorithm is shown in figure 3. It was implicitly shown in the previous discussion that any polynomial  $f(x) \in GF(2^n)$  can be expressed as

$$f(x) = x^A (1+x)^B f_1(x)^{e_1} f_2(x)^{e_2} \dots f_u(x)^{e_u} \bmod(g(x))$$

for some finite  $u$ , where  $f_i(x) \in H$ ,  $1 \leq i \leq u$ . The determination of the logarithm is then straight forward.

Figure 3.

16a.



#### 4. Experimental Results.

The algorithm was programmed in FORTRAN on a Honeywell 60/66 for  $n=31$ ,  $g(x) = 1+x^3+x^{31}$  and  $s=8$ .

Experiment 1. The algorithm was initially used to decompose 1000 polynomials chosen at random. The average number of polynomials in a decomposition ( $u$ ) was 5.97 and the maximum number in any decomposition was 28. The frequency with which the polynomials of the various degrees appeared in the decomposition was as follows:

degree of polynomial	no. of times appearing in decomposition	probability
0....	42	0.007
1....	0	0.
2....	505	0.085
3....	625	0.105
4....	720	0.121
5....	734	0.123
6....	773	0.129
7....	777	0.130
8....	725	0.121
9....	354	0.059
10....	275	0.046
11....	188	0.031
12....	116	0.019
13....	83	0.014
14....	44	0.007
15....	9	0.002

The average processing time was 654.3 milliseconds and the maximum processing time was 2.4907 seconds. The distribution of the number of polynomials in a decomposition ( $u$ ) was as follows:

$u$	freq.	$u$	freq.	$u$	freq.	$u$	freq.
1	0	9	40	17	4	25	0
2	22	10	34	18	1	26	0
3	126	11	22	19	0	27	0
4	176	12	9	20	0	28	1
5	185	13	9	21	1	$\geq 29$	0
6	153	14	7	22	0		
7	121	15	6	23	0		
8	83	16	0	24	0		

Experiment 2. A further set of 5000 polynomials, chosen at random, was decomposed. The average number of polynomials in a decomposition was 5.99 and the maximum number of polynomials in a decomposition was 26. The frequency with which the polynomials of the various degrees appeared in the decomposition was as follows:

degree of polynomial	no. of times appearing in decomposition	probability
0....	174	0.006
1....	0	0.
2....	2539	0.085
3....	3285	0.110
4....	3567	0.119
5....	3587	0.120
6....	3819	0.128
7....	3784	0.126
8....	3520	0.118
9....	2014	0.067
10....	1524	0.051
11....	885	0.030
12....	613	0.020
13....	365	0.012
14....	191	0.006
15....	71	0.002

The average processing time was 651.6 milliseconds and the maximum processing time was 2.3783 seconds. The distribution of the number of polynomials in a decomposition was as follows:

u	freq.	u	freq.	u	freq.	u	freq.
1	1	8	369	15	25	22	1
2	140	9	252	16	9	23	2
3	672	10	168	17	10	24	0
4	785	11	121	18	2	25	0
5	886	12	69	19	6	26	1
6	828	13	44	20	3	$\geq 27$	0
7	552	14	24	21	3		

The results between the two experiments are remarkably consistent and indicate the effectiveness of the algorithm. A table of these polynomials  $f(x)$  whose decomposition required 15 or more polynomials is shown on the next page.

Table. A List of those 5000 Polynomials, Chosen at Random, whose Decomposition requires 15 or more Polynomials.

### 5. Comments on the Algorithm.

The algorithm presented in this report appears to be reasonably efficient and has guaranteed convergence in finite time to the required logarithm. Sharp estimates of the maximum time required could likely be obtained. It is difficult to compare the efficiency of this algorithm with that of the Swedish algorithm but based on the time estimates available it would appear this algorithm is faster, perhaps by as much as a factor of 20. The different characteristics of the machine and different languages used however, leave this figure with doubtful significance.

Perhaps more important than the algorithm itself, are the techniques introduced to reduce the size of the stored set of polynomials and their logarithms. This is essentially a technique whereby computation time is traded for storage, with guaranteed convergence. The fact that such a tradeoff is possible while preserving convergence is important. It would seem to place in doubt the wisdom of using the logarithm method for public key cryptography unless the size of the finite field used were several orders of magnitude larger than  $2^{127}$ . The fact that an adversary might have several days to work on a given key, place its security in jeopardy. In future work it is hoped to develop practical algorithms for  $\text{GF}(2^{127})$  to substantiate this statement.

### References

- [1]. I.F. Blake and J.W. Mark, Final Report, DSS Contract No. OSU 80-00117.
- [2]. W. Diffie and M.E. Hellman, New Directions in Cryptography, IEEE Trans. Information Theory, vol. IT-22, 644-654, 1976.
- [3]. S. Berkovits, J. Kowalchuk and B. Schanning, Implementing a Public Key Scheme, IEEE Communications Magazine, vol. 17, 2-3, 1979.
- [4]. S. Pohlig and M.E. Hellman, An Improved Algorithm for Computing Logarithms over GF( $p$ ) and its Cryptographic Significance, IEEE Trans. Information Theory, vol. IT-24, 106-110, 1978.
- [5]. R. Fuji-Hara, S. Vanstone and I. Blake, Progress Report, this contract, December, 1982.
- [6]. D. Knuth, The Art of Computer Programming, volume 3, Searching and Sorting, Addison-Wesley Publishing Company, Reading, Mass., 1973.
- [7]. T. Herlestam and R. Johannesson, On Computing Logarithms over  $GF(2^P)$ , IEEE International Symposium on Information Theory, Santa Monica, CA, February, 1981.
- [8]. R. McEliece, The Theory of Information and Coding, Encyclopedia of Mathematics, vol. 3, Addison Wesley Publishing Company, Reading, Mass. 1977.
- [9]. E. Berlekamp, Algebraic Coding Theory, McGraw Hill Book Co., New York, 1968.

Appendix. A List of the Polynomials in the Set  $H_2^{**}$ .

1 1 0 0 0 0 0 0 0 0 1	1 1 1 1 0 0 0 0 0 0 1	1 1 1 0 0 1 0 1 0 1 1
1 0 0 0 1 0 0 0 0 0 1	1 0 1 1 1 0 0 0 0 0 1	1 1 1 1 1 0 1 0 1 0 1 1
1 1 1 0 1 0 0 0 0 0 1	1 1 1 0 0 1 0 0 0 0 1	0 1 1 1 0 0 1 1 0 1 0 1 1
1 1 0 0 0 1 0 0 0 0 1	1 0 1 1 0 1 0 0 0 0 1	1 1 1 0 1 0 1 1 1 1 0 1 1
1 0 1 1 0 1 0 0 0 0 1	1 0 1 0 1 1 0 0 0 0 1	0 1 1 1 1 0 0 0 0 1 0 1 1
1 1 0 1 0 0 1 0 0 0 1	1 1 1 0 0 0 1 0 0 0 1	1 1 1 1 1 0 0 0 0 1 1 1 1
1 0 0 1 1 0 1 0 0 0 1	1 1 0 0 0 1 0 1 0 0 0 1	0 0 0 1 0 0 0 0 0 1 1 1 1
1 1 1 1 1 0 1 0 0 0 1	1 1 0 0 0 0 1 1 0 0 0 1	1 0 1 0 1 0 1 0 0 0 1 1 1 1
1 1 1 1 0 1 1 1 0 0 1	1 0 0 1 0 1 0 1 0 0 0 1	0 1 1 0 1 1 0 0 0 0 1 1 1 1
1 0 1 1 1 1 1 1 0 0 1	1 1 1 1 1 0 1 0 1 0 0 1	1 1 1 1 0 0 0 0 1 0 1 1 1 1
1 1 1 1 0 0 0 0 1 0 1	1 0 0 1 0 0 1 0 1 0 0 1	0 1 1 1 0 0 0 0 1 0 1 1 1 1
1 0 1 0 1 0 0 1 0 0 1	1 1 1 1 0 0 1 1 1 0 0 1	1 0 1 1 1 0 0 0 1 0 1 1 1 1
1 0 0 1 1 1 0 0 1 0 1	1 1 1 0 0 0 1 1 1 0 0 1	1 1 0 0 0 0 0 0 1 0 1 1 1 1
1 1 0 0 1 1 0 0 1 0 1	1 0 1 1 1 0 1 1 1 0 0 1	1 0 0 1 1 0 1 0 1 0 1 1 1 1
1 1 0 0 0 1 0 1 0 0 1	1 1 1 1 1 1 1 1 1 0 0 1	1 1 0 0 0 0 1 1 1 0 1 1 1 1
1 0 1 0 0 1 3 1 0 3 1	1 1 1 0 1 0 0 0 0 1 0 1	1 1 0 1 1 1 1 1 0 1 1 1 1 1
1 1 1 1 0 1 0 1 0 0 1	1 0 0 1 0 1 0 0 0 1 0 1	1 0 1 1 1 1 1 1 0 1 1 1 1 1
1 0 1 1 1 1 0 1 0 0 1	1 1 1 1 0 0 1 1 1 0 0 1	1 0 0 0 0 0 0 0 1 0 1 1 1 1 1
1 0 0 1 1 1 1 0 1 0 1	1 0 0 1 1 0 0 1 1 1 0 0 1	1 1 1 1 0 0 0 0 0 1 1 1 1 1 1
1 1 0 0 0 1 0 1 1 0 1	1 1 1 1 1 1 1 1 1 0 0 1	0 1 1 1 1 0 0 0 0 1 1 1 1 1 1
1 1 1 0 0 0 1 0 1 0 1	1 1 0 0 0 1 1 1 1 1 0 0 1	1 0 0 1 0 0 0 0 1 0 1 1 1 1 1
1 0 1 0 1 0 0 1 0 1 1	1 0 0 0 1 0 0 0 1 1 1 0 0 1	1 1 0 1 0 0 0 0 0 1 1 1 1 1 1
1 0 0 1 1 1 0 0 1 0 1	1 1 1 1 1 0 0 1 1 1 0 0 1	1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
1 1 1 0 0 0 1 0 1 1 0 1	1 1 0 0 0 1 0 0 1 1 1 0 0 1	0 1 1 1 1 0 0 0 0 0 1 1 1 1 1
1 0 1 0 1 1 1 0 1 0 1	1 0 0 0 1 1 1 0 0 1 1 1 0 0 1	1 0 0 1 0 0 0 0 0 0 0 0 0 0 1
1 0 0 1 0 1 1 1 1 0 1	1 1 1 0 0 0 1 1 1 1 0 0 1	1 1 1 0 1 0 0 0 0 0 0 0 0 0 1
1 0 0 1 1 1 1 1 1 0 1	1 0 0 0 1 1 1 1 0 0 1 1 0 0 1	1 1 1 0 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 3 0 1 1	1 0 0 0 0 0 0 1 0 1 1 0 0 1	1 0 1 1 1 1 1 1 0 1 1 1 1 1 1
1 0 1 0 0 1 0 0 0 1 1	1 1 1 1 0 1 0 1 0 1 0 1 0 1	1 1 0 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1 0 1 0 1 0 1 1 0 1	1 1 0 0 0 0 1 0 1 0 1 0 1 0 1	1 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1
1 0 1 0 1 1 1 1 1 0 1	1 0 0 0 0 1 1 1 0 0 1 0 1 0 1	1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1
1 0 0 1 1 1 1 1 1 1 0 1	1 0 0 0 0 0 1 1 1 1 0 0 1 0 1	1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 1 0 0 1 0 0 0 1 1	1 1 1 1 0 1 0 1 0 1 0 1 0 1	1 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 1 1	1 0 0 0 0 0 0 1 1 0 1 0 1 0 1	1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1
1 1 1 1 0 1 1 1 0 0 1	1 1 1 1 1 0 0 1 0 1 0 1 0 1	1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 1 0 1 1 0 1	1 0 0 0 0 1 1 1 0 0 1 0 1 0 1	1 1 1 0 1 1 1 1 0 0 0 0 0 0 0 1
1 0 1 0 0 1 0 0 1 0 1	1 0 0 0 0 0 1 1 1 1 0 0 1 0 1	1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 0 0 1 0 1 0 1	1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1	1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1
1 1 0 0 0 1 0 1 0 1 1	1 1 1 0 0 1 1 1 0 0 1 0 1 0 1	1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 1
1 0 0 0 0 1 0 0 1 1 1	1 0 0 0 0 1 0 1 1 1 0 0 1 0 1	1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1
1 0 1 0 0 1 1 0 1 1 1	1 1 1 1 0 0 0 1 1 1 1 0 0 1 0 1	1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1
1 0 0 0 1 1 1 0 1 1 1	1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 0 1	1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 1
1 1 1 0 0 0 0 1 1 1 1	1 1 0 0 0 1 0 1 0 0 0 3 1 1	1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 1
1 1 0 0 0 0 1 1 1 1 1	1 0 0 0 0 1 1 1 0 0 0 3 1 1	1 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1
1 0 1 0 0 0 1 1 1 1 1	1 1 1 0 0 1 1 1 0 0 0 3 1 1	1 0 1 0 0 1 0 1 0 0 0 1 0 0 0 1
1 0 1 0 0 1 0 1 1 1 1	1 1 0 0 0 0 1 1 1 0 0 0 3 1 1	1 0 0 1 0 1 0 1 3 0 1 0 1 0 0 1
1 0 0 1 1 0 0 1 1 1 1	1 1 1 1 1 0 0 0 1 0 0 0 1 1	1 1 1 0 1 0 1 1 3 0 1 0 1 0 0 1
1 1 0 0 0 1 1 1 1 1 1	1 0 0 0 0 1 0 1 0 1 0 0 0 1 1	1 0 1 1 1 1 1 3 0 1 0 1 0 0 1
1 0 0 1 0 1 1 1 1 1 1	1 1 0 0 1 1 0 1 0 1 0 0 0 1 1	1 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1
1 1 0 0 1 1 1 1 1 1 1	1 0 0 0 1 1 1 1 1 0 0 0 1 1	1 1 0 1 0 1 0 1 1 1 0 1 1 0 0 1
1 0 0 1 0 0 0 0 0 0 1	1 0 1 0 0 0 0 0 1 0 1 0 1 1	1 0 0 1 1 1 1 1 0 1 1 1 0 0 0 1



14









FINITE FIELD TECHNIQUES FOR SHIFT REGISTERS  
WITH APPLICATIONS TO RANGING PROBLEMS AND  
CRYPTOGRAPHY

Final Report

Project No. 106-16-02

Prepared for

The Department of Communications  
under DDS Contract No. OSU 82-00090

by

I. Blake, R. Fuji-Hara, R. Mullin and S. Vanstone  
University of Waterloo

Scientific Authority

B. Bryden

Communications Research Center.

### 1. Introduction.

Given the finite field  $GF(q)$ , a generator  $\alpha$  and an integer  $x$ ,  $0 \leq x \leq q-2$ , there is a simple and fast procedure for computing  $\alpha^x$ .

Write

$$x = \sum_{i=0}^t \beta_i 2^i \text{ where } \beta_i \in \{0,1\}, 0 \leq i \leq t.$$

Then

$$\alpha^x = \alpha^{\sum_{i=0}^t \beta_i 2^i} = \prod_{i=0}^t \alpha^{\beta_i 2^i}.$$

Thus computing  $\alpha^x$  requires the repeated application of the squaring operation. The number of operations is approximately  $O(\log_2 x)$ . This operation is commonly referred to as exponentiation.

The inverse operation to exponentiation does not appear to be nearly as simple. This is the problem we have considered.

### THE DISCRETE LOGARITHM PROBLEM.

Let  $\alpha$  be a generator for the finite field  $GF(q)$ . Given  $\beta \in GF(q)$ ,  $\beta \neq 0$ , find the integer  $x$ ,  $0 \leq x \leq q-2$ , such that  $\beta = \alpha^x$ .

The integer  $x$  is called the logarithm of  $\beta$  to the base  $\alpha$  and is written  $x = \log_\alpha \beta$ . We can certainly find  $\log_\alpha \beta$  by computing successive powers for  $\alpha$ . For large  $q$  the approach is computationally infeasible, and hence the discrete logarithm problem is more accurately stated by including the term "computationally feasible".

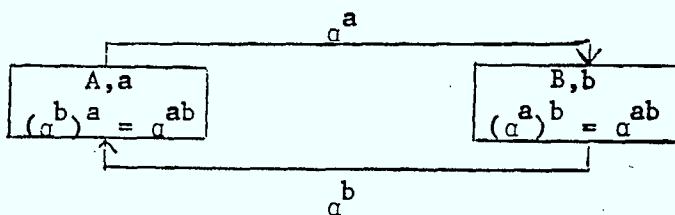
The problem of finding logarithms in finite fields is important in a variety of situations including the estimation of elapsed clock cycles between two states of a linear feedback shift register. Recently, finite field exponentiation has received much attention as a cryptographic scheme and has been used as the basis for a public key distribution system. We will consider this particular application in more detail in section 2.

In this report we consider various aspects of the discrete logarithm problem. In section 3 we present results on the Euclidean Algorithm and apply these results to some simple algorithms for finding logs. In section 4 we give an improved algorithm which is subexponential and random time. In section 5 test results of this algorithm in  $GF(2^{61})$  are given. In section 6 we briefly describe an algorithm due to L. Adleman [1] and compare it with the one in section 4. In section 7 we discuss the field  $GF(2^{127})$ . Section 8 briefly covers the problem of field representations.

## 2. Cryptographic Significance of Discrete Logs.

In 1976 W. Diffie and M. Hellman [6] proposed a scheme for public key distribution which is based on the assumed intractability of computing discrete logarithms in large fields.

Two parties A and B are using a cryptographic system which requires that both A and B share a common key K. Part of the security of such a system is the passing of the key between the two parties. Suppose that a finite field  $\text{IF}$  and a generator  $a$  for  $\text{IF}$  are public. A generates a random integer  $a$  and B generates a random integer  $b$ . A sends B  $a^a$  and B sends A  $a^b$ . A computes  $(a^b)^a$  and B computes  $(a^a)^b$ . A and B now share the common key  $a^{ab}$ . This scheme is depicted in the following diagram.



A tapper on the channel sees  $a^a$  and  $a^b$  but unless this tapper can compute logarithms in  $\text{IF}$  he cannot (in any obvious way) obtain  $a^{ab}$ .

This system is referred to as a Public Key Distribution System (PKDS). This PKDS has been implemented by various corporations. Hewlett-Packard [15] has designed and built a VLSI chip for exponentiation in  $GF(2^{127})$ . The field is generated by the primitive trinomial  $f(X) = X^{127} + X + 1$ . Mitre corporation [14] has a software implementation using the same field and representation. Both of these systems

are designed for key passing for the DES (Data Encryption Standard) system. Collin's Radio, a supplier to members of the Federal Reserve banking system in the U.S.A., has implemented a PKDS and, recently, Fujitsu (Japan) has announced that it will adopt PKDS. Fujitsu's implementation will be different than those cited above. It intends to use exponentiation in  $GF(p)$ ,  $p$  a prime number of about 100 digits.

As a second application, Pohlig and Hellman [11] suggest the following.

Let  $\mathbb{F}$  be a finite field  $GF(q)$  and  $k$  be a positive integer,  $0 \leq k \leq q-2$ , such that  $(k, q-1) = 1$ . We can now use exponentiation in  $\mathbb{F}$  to form a symmetric cryptosystem ([15]). If the message  $M$  is an element of  $\mathbb{F}$ ,  $M \neq 0$ , then the cipher text  $C$  is formed as

$$M^k \equiv C \pmod{\mathbb{F}}$$

where this notation means that  $M^k$  is reduced modulo  $q$ , if  $q$  is a prime number, and modulo the generating polynomial if  $q$  is a power ( $\geq 2$ ) of a prime number. In order to decipher we require the integer  $d$  such that

$$kd \equiv 1 \pmod{q-1}.$$

If taking logs in  $\mathbb{F}$  is easily done then this system is insecure. We know of no implementations of this technique.

We indicate one final application of exponentiation in a finite field. Two parties A and

B wish to communicate. Messages are elements in  $GF(q)$  and  $GF(q)$  is public. Party A generates a random integer  $a$ ,  $(a, q-1) = 1$  and B generates  $b$ ,  $(b, q-1) = 1$ . If A wants to send message  $M$  to B, A transmits  $M^a$ . B, having received  $M^a$ , sends A  $M^{ab}$ . A now sends B  $M^b$  since A can compute  $(M^a)^b$ . B determines the message by applying  $b^{-1}$ . A tapper will see  $M^a$ ,  $M^{ab}$ , and  $M^b$ . Suppose the tapper has an efficient method of computing logarithms to the base  $a$  in this field. It may appear at first glance that this does not help a great deal since an obvious way to find  $a$  and  $b$  is to compute logarithms in the field to the base  $M^a$  deducing  $b$  from  $M^{ab}$  and then finding  $b^{-1}$ . This is, of course, not the case since for some integer  $t$ ,  $M = a^t$ . If the tapper can compute logs

to the base  $\alpha$  then he can determine  $a\alpha t = s_1$ ,  $a\beta t = s_2$  and  $b\beta t = s_3$  and, hence, deduce  $a$  and  $b$ . Thus the complexity of this method is no greater than taking logarithms to a fixed base.

At least one other application of finite field exponentiation to voting protocols appears in the literature. The interested reader is referred to the paper by R. Lipton and A. Wigderson [7] for details.

Because of these applications it is of some interest to consider possible approaches to computing logarithms in a finite field in a feasible manner. It was observed by S. Pohlig and M. Hellman [11] that if  $q-1$  is highly composite then the complexity of finding logarithms in  $GF(q)$  is greatly reduced. Because of actual implementations of PKDS it is of interest to consider finite fields of characteristic 2. For these reasons the focus of this report is on finite fields of the form  $GF(2^n)$  where  $2^n-1$  is a prime. The first few values of  $n$  for which such a field exists are 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127 and 521. In a previous report ([3]) it was shown that computing logarithms in  $GF(2^n)$ ,  $n \leq 31$  can be done efficiently. It is the purpose of this report to establish that computing logarithms in  $GF(2^n)$ ,  $n \leq 127$  is feasible.

We conclude this section with some data which illustrate the sizes of the fields we are considering.

Let  $\alpha$  be a generator in  $GF(2^n)$ . Suppose we determine the log of an element  $\beta$  by the naive approach of computing successive powers of  $\alpha$  until we find  $\beta$ . Suppose we use a feedback shift register which will compute 1 billion consecutive powers of  $\alpha$  per second. The following Table 1 illustrates the time (in years) to exhaust  $GF(2^n)$  for various values of  $n$ .

Table 1

n	The Number of Elements in $GF(2^n)$	Time (in years)
31	2, 147, 483, 647	$.68 \times 10^{-7}$
61	2, 305, 843, 009, 213, 693, 951	73.12
89	6 1 8 9 7 0 0 1 9 6 4 2 6 9 0 1 3 7 4 4 9 5 6 2 1 1 2	$19.6 \times 10^9$
107	1 6 2 2 5 9 2 7 6 8 2 9 2 1 3 3 6 3 3 9 1 5 7 8 0 1 0 2 8 8 1 2 7	$5 \times 10^{15}$
127	1 7 0 1 4 1 1 8 3 4 6 0 4 6 9 2 3 1 7 3 1 6 8 7 3 0 3 7 1 5 8 8 4 1 0 5 7 2 7	$5 \times 10^{21}$

3. The Euclidean Algorithm and its Applications.

We first consider the Euclidean Algorithm for integers. The greatest common divisor of two integers  $a$  and  $b$  is denoted by  $(a, b)$ .

THE EUCLIDEAN ALGORITHM FOR INTEGERS.

Given integers  $a$  and  $b$  (not both 0) there exist integers  $s$  and  $t$  such that

$$as + bt = (a, b).$$

The most practical method for determining  $s$  and  $t$  is the following. Let

$$\begin{aligned}s_{-1} &= 1, t_{-1} = 0, r_{-1} = a \\ s_0 &= 0, t_0 = 1, r_0 = b\end{aligned}$$

$$\text{and for } i \geq 1, \text{ if } q_i = \left\lfloor \frac{r_{i-2}}{r_{i-1}} \right\rfloor \text{ then } \left. \begin{aligned}r_i &= r_{i-2} - q_i r_{i-1}, \quad 0 \leq r_i < r_{i-1} \\ s_i &= s_{i-2} - q_i s_{i-1} \\ t_i &= t_{i-2} - q_i t_{i-1},\end{aligned}\right\} \quad (A)$$

Equations (A) have the matrix formulation

$$\begin{bmatrix} r_{i-1} \\ r_{i-2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & q_i \end{bmatrix} \begin{bmatrix} r_i \\ r_{i-1} \end{bmatrix} \quad (B)$$

$$\begin{bmatrix} s_{i+1} & s_i \\ t_{i+1} & t_i \end{bmatrix} = \begin{bmatrix} s_i & s_{i-1} \\ t_i & t_{i-1} \end{bmatrix} \begin{bmatrix} -q_{i+1} & 1 \\ 1 & 0 \end{bmatrix} \quad (C)$$

From (B) and (C) we deduce that

$$\begin{bmatrix} r_i \\ r_{i-1} \end{bmatrix} = \begin{bmatrix} s_i & t_i \\ s_{i-1} & t_{i-1} \end{bmatrix} \begin{bmatrix} r_{-1} \\ r_0 \end{bmatrix} = \begin{bmatrix} s_i & t_i \\ s_{i-1} & t_{i-1} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

From (C) we deduce that the  $t_i$  alternate in sign. Since

$$t_{i+1} = t_{i-1} - q_{i+1}t_i$$

and

$$r_{i-1} = q_{i+1}r_i + r_{i+1}$$

then

$$r_i t_{i+1} = r_i t_{i-1} - r_i q_{i+1} t_i$$

and

$$t_i r_{i+1} = t_i r_{i-1} - t_i q_{i+1} r_i.$$

Hence,

$$|t_{i+1}|r_i + |t_i|r_{i+1} = |t_{i-1}|r_i + |t_i|r_{i-1}, i \geq 0.$$

and

$$a = |t_{i-1}|r_i + |t_i|r_{i-1}, i \geq 0. \quad (D)$$

From (D) we observe that if  $r_{i-1} > \sqrt{a}$  then  $|t_i| < \sqrt{a}$ . Hence, if there exists an index  $j$  such that  $r_{j-1} > \sqrt{a}$  and  $r_j \leq \sqrt{a}$  then  $|t_j| < \sqrt{a}$ . This observation can be used as a basis for an algorithm to compute logarithms in  $GF(p)$ ,  $p$  is a prime number.

#### ALGORITHM.

Part A. Compute and store the logarithms of all integers  $x$ ,  $0 \leq x \leq \sqrt{p}$ . This part of the algorithm need only be done once but is by no means simple.

Part B. Given an integer  $b$ ,  $1 \leq b \leq p-2$  for which we want  $\log_a b$ , do the following

- (i) If  $b \leq \sqrt{p}$ , look up  $\log b$  in the database.
- (ii) If  $b > \sqrt{p}$  then find integers  $t$  and  $r$  (by the Euclidean Algorithm) such that  $tb + sp = r$  which is equivalent to  $tb \equiv r \pmod{p}$ ,  $|t| < \sqrt{p}$ ,  $r < \sqrt{p}$ . Compute  $\log b = \log r - \log |t|$ . Recall that  $\log (-1) = \frac{p-1}{2}$  and that  $\log r$  and  $\log |t|$  are in the database.

EXAMPLE. Consider GF(163) with generator 2.

x	$\log_2 x$
1	0
2	1
3	101
4	2
5	15
6	102

x	$\log_2 x$
7	73
8	3
9	40
10	16
11	47
12	103

(DATABASE)

Suppose we require the log of 76. Applying the Euclidean Algorithm to the integers 163 and 76 gives the table

$s_i$	$t_i$	$r_i$
1	0	163
0	1	76
1	-2	11

and hence,

$$(-2)(76) \equiv 11 \pmod{163}$$

and

$$\begin{aligned}\log_2 76 &= \log 11 - \log -2 \\ &= 47 - \{\log (-1) + \log 2\} \\ &= 47 - \{81 + 1\} \\ &= 127.\end{aligned}$$

For fields of the form  $GF(p^n)$  we will require the Euclidean Algorithm for polynomials. The statements are similar to those above but we include them for completeness.

#### THE EUCLIDEAN ALGORITHM FOR POLYNOMIALS.

Let  $A(x)$  and  $B(x)$  be polynomials over  $GF(q)$ . As for integers  $(A(x), B(x))$  is the greatest common divisor of  $A(x)$  and  $B(x)$ . There exist polynomials  $s(x)$  and  $t(x)$  over  $GF(q)$  such that

$$s(x)A(x) + t(x)B(x) = (A(x), B(x)).$$

An  $s(x)$  and  $t(x)$  can be determined by the algorithm

$$s_{-1}(x) = 1, t_{-1}(x) = 0, r_{-1}(x) = A(x)$$

$$s_0(x) = 0, t_0(x) = 1, r_0(x) = B(x)$$

⋮

$$s_i(x), \quad t_i(x), \quad r_i(x)$$

where

$$\left. \begin{array}{l} r_i(x) = r_{i-2}(x) - q_i(x)r_{i-1}(x) \\ s_i(x) = x_{i-2}(x) - q_i(x)s_{i-1}(x) \\ t_i(x) = t_{i-2}(x) - q_i(x)t_{i-1}(x) \end{array} \right\} i \geq 1.$$

The following properties are easily established by induction.

$$(1) \quad s_i(x)A(x) + t_i(x)B(x) = r_i(x), \quad i \geq -1.$$

$$(2) \quad \deg s_i(x) + \deg r_{i-1}(x) = \deg B(x)$$

$$(3) \quad \deg t_i(x) + \deg r_{i-1}(x) = \deg A(x).$$

From (1)  $t_i(x)B(x) + s_i(x)A(x) = r_i(x)$  which is equivalent to  
 $t_i(x)B(x) \equiv r_i(x) \pmod{A(x)}.$

From (3) and the fact that  $\deg r_i(x) < \deg r_{i-1}(x)$

$$\deg t_i(x) + \deg r_i(x) < \deg A(x).$$

One of the most important properties (for our purpose) of Euclid's Algorithm for polynomials is given in the next theorem.

**THEOREM 3.1.** Let  $\ell$  and  $k$  be non-negative integers with  $k \geq \deg(A(x), B(x))$  satisfying  $\ell + k = \deg A(x) - 1$ . Then there exists a unique index  $j$ , such that  $\deg t_j(x) \leq \ell$  and  $\deg r_j(x) \leq k$ .

*Proof.* The  $\deg r_i(x)$  is a strictly decreasing function of  $i$ . Define the index  $j$  by  $\deg r_{j-1}(x) \geq k+1$  and  $\deg r_j(x) \leq k$ . Since  $\deg t_j(x) + \deg r_{j-1}(x) = \deg A(x)$  then  $\deg t_j(x) + k + 1 \leq \deg A(x)$  or  $\deg t_j(x) \leq \ell$ .

In the algorithm of the next section we will apply this theorem in the case where  $(A(x), B(x)) = 1$ ,  $\ell = \lfloor \frac{\deg A(x)}{2} \rfloor$  and, thus,  $k \leq \ell$ .

Recall that  $[x]$  is the greatest integer less than  $x$ .

EXAMPLE. Consider the polynomials  $A(x) = x^9 + x + 1$  and  $B(x) = x^5 + x^2 + x + 1$  over  $GF(2)$ . The steps in the Euclidean Algorithm are displayed in the table.

$s_i(x)$	$t_i(x)$	$r_i(x)$	$q_i(x)$
1	0	$x^9 + x + 1$	
0	1	$x^5 + x^2 + x + 1$	
1	$x^4 + x + 1$	$x^4 + x^3 + x$	$x^4 + x + 1$
$x^3 + 1$	$x^7 + x^3 + x$	$x^3 + 1$	$x + 1$
$x^4 + x^3 + x$	$x^8 + x^7 + x^3 + x^2 + 1$	1	$x + 1$

From the table we get

$$(x^4 + x + 1)(x^5 + x^2 + x + 1) \equiv x^8 + x^7 + x^3 + x^2 + 1 \pmod{A(x)}$$

where  $\deg t_1(x) = 4 \leq \lfloor \frac{\deg A(x)}{2} \rfloor$  and  $\deg r_1(x) = 4 \leq \lfloor \frac{\deg A(x)}{2} \rfloor$ .

As a direct analogue of the algorithm given earlier in this section we describe an algorithm for computing logarithms in  $GF(p^n)$ ,  $n \geq 2$ . We will think of the elements in this field as polynomials over  $GF(p)$  of degree at most  $n-1$ . Let  $f(x)$  be the polynomial generating the field.

#### ALGORITHM.

Part A. Compute and store the logarithms of all elements in  $GF(p^n)$  having degree at most  $\lfloor \frac{n}{2} \rfloor$ .

Part B. Let  $b(x)$  be a nonzero element of  $GF(p^n)$  whose log is required.

- (i) If  $\deg b(x) \leq \lfloor \frac{n}{2} \rfloor$ , find  $\log b$  in the database.
- (ii) If  $\deg b(x) > \lfloor \frac{n}{2} \rfloor$ , apply the Euclidean Algorithm to  $f(x)$  and  $b(x)$  to find polynomials  $t(x)$  and  $r(x)$  such that  $t(x)b(x) \equiv r(x) \pmod{f(x)}$ ,  
 $\deg t(x) \leq \lfloor \frac{n}{2} \rfloor$  and  $\deg r(x) \leq \lfloor \frac{n}{2} \rfloor$ .

Compute  $\log b(x) = \log r(x) - \log t(x)$ .

EXAMPLE. Consider  $GF(2^7)$  generated by  $f(x) = x^7 + x + 1$ . Suppose we want to compute logarithms to the base  $x$ .

$\beta$	$\log_x \beta$
1	0
$x$	1
$x^2$	2
$1+x$	7
$x+x^2$	8
$x^2+x^3$	9
$1+x^2$	14
$x+x^3$	15

$\beta$	$\log_x \beta$
$1+x+x^2+x^3$	21
$1+x+x^3$	31
$1+x+x^2$	56
$x+x^2+x^3$	57
$1+x^3$	63
$1+x^2+x^3$	90
$x^3$	3

(DATABASE)

Suppose we want the logarithm of the element  $b(x) = 1 + x^3 + x^6$ .

Apply the Euclidean Algorithm.

$s_i(x)$	$t_i(x)$	$r_i(x)$
1	0	$f(x)$
0	1	$b(x)$
1	$x$	$x^4+1$
$x^2$	$x^3+1$	$x^3+x^2+1$

$$(x^3+1)b(x) \equiv x^3+x^2+1 \pmod{f(x)}.$$

$$\begin{aligned}\log_a b(x) &= \log_a (x^3+x^2+1) - \log_a (x^3+1) \\ &= 90 - 63 \\ &= 27.\end{aligned}$$

This can be checked easily since we know that  $x^9 = x^2 + x^3$  (from database) and so

$$\begin{aligned}x^{27} &= (x^2+x^3)^3 = x^6 + x^7 + x^8 + x^9 \\&= x^6 + x + 1 + x^2 + x + x^3 + x^2 \\&= x^6 + x^3 + 1.\end{aligned}$$

This algorithm is feasible for fields as large as  $GF(2^{31})$  (see [ 3]) but for fields much larger is infeasible. In section 4 we describe a modified version of this algorithm which is feasible for much larger fields.

#### 4. The Algorithm.

The algorithms of the previous section require approximately  $\sqrt{q}$  units in time and space for a field with  $q$  elements. For  $q$  as large as  $2^{61}$  this is not a realistic approach to computing logs. In this section we present a modified version of these algorithms which is random time and subexponential in  $\log q$ . We will describe the algorithm for  $GF(p^n)$ ,  $n \geq 2$  but it also applies to  $GF(p)$ ,  $p$  a prime by simply replacing polynomials by integers and irreducible polynomials by primes.

#### THE ALGORITHM. ( $GF(p^n)$ generated by $f(x)$ .)

Part A. Compute and store the logarithms of all irreducible polynomials of degree at most some constant  $b$ . The number  $b$  is determined by the size of the field. This part of the algorithm is by no means simple but it only needs to be done once.

Part B. We require the log of element  $b(x)$ .

- (i) If  $\deg b(x) \leq b$ , find  $\log b(x)$  in the database.
- (ii) If  $\deg b(x) > b$  then
  - (a) Set  $i = 1$ .
  - (b) Generate a random integer  $a_i$  and compute  $g_i(x) \equiv x^{a_i} b(x) \pmod{f(x)}$ .

(c) Apply Euclid's Algorithms to  $g_i(x)$  and  $f(x)$  to get polynomials  $t_i(x)$  and  $r_i(x)$  such that

$$t_i(x)g_i(x) \equiv r_i(x) \pmod{f(x)}$$

and  $\deg t_i(x)$  and  $\deg r_i(x) \leq \frac{n}{2}$ .

(d) Factor  $t_i(x) = \prod_{j=1}^t p_j^{e_j}(x)$  and  $r_i(x) = \prod_{h=1}^s q_h^{d_h}(x)$ .

If  $\deg p_j(x) \leq b$ ,  $1 \leq j \leq t$ , and  $\deg q_h(x) \leq b$ ,  $1 \leq h \leq s$  then compute (from the database)

$$\log b(x) = -a_i = \sum_{j=1}^t e_j \log p_j(x) + \sum_{j=1}^s d_j \log q_j(x).$$

Otherwise, set  $i = i+1$  and go to step (b).

Part A of the algorithm is what we refer to as constructing the database for the field. Our approach to this problem is as follows. Having selected the integer  $b$ , let the irreducible polynomials of degree at most  $b$  be  $p_1(x), p_2(x), \dots, p_t(x)$  and let  $x_i = \log p_i(x)$ ,  $1 \leq i \leq t$ . We attempt to find  $t$  linearly independent equations in the  $t$  unknowns  $x_i$ ,  $1 \leq i \leq t$ . One way to get these equations is by applying Part B of the algorithm to each  $p_j(x)$ ,  $1 \leq j \leq t$ . If, in step d, the  $t_i(x)$  and  $r_i(x)$  both factor into irreducibles of degree at most  $b$  we get an equation in the unknown  $x_j$ ,  $1 \leq j \leq t$ . We are most interested in the case where  $p = 2$  and  $p^n - 1$  is a prime. Finding the equations in this manner is essentially random and, hence, once  $t$  such equations are found and if  $2^n$  is reasonably large the probability that the equations are linearly independent is also large. To be more precise, suppose we have  $t-1$  independent equations and we add another random equation. What is the probability that this equation is linearly independent from the others? Since the coefficients of these equations are in  $GF(2^n - 1)$  the probability is

$$\frac{(2^n - 1)^t - (2^n - 1)^{t-1}}{(2^n - 1)^t} = \frac{2^n - 2}{2^n - 1}$$

which is close to one for even moderately sized  $n$ .

There are other techniques for deriving equations. We will discuss these in a later section.

Part B of the algorithm requires that we do much factoring. Various techniques for factoring polynomials over  $GF(p)$  are known (see [4], [5], [9], [12]). As our interests have mainly centred around  $GF(2^n)$  we have implemented the Berlekamp Algorithm [2] since it appears to be the most efficient method over  $GF(2)$ .

For a given polynomial  $b(x)$ , Part B of the algorithm may be executed many times before some  $g_i(x)$  gives a  $t_i(x)$  and  $r_i(x)$  which factor into the database. For example, in  $GF(2^{61})$  with  $b = 12$  tests indicate that on average Part B is required about 50 times for any given polynomial  $b(x)$ . This means that in most passes either  $t_i(x)$  or  $r_i(x)$  contains an irreducible factor with degree greater than  $b$ . Hence, when we start to factor  $t_i(x)$  and  $r_i(x)$  we attempt to find a large factor as quickly as possible. If we find one then we terminate the factoring process and start Part B again. To be more explicit, we define the simple part of a polynomial  $g(x)$  to be the product of the irreducible factors of  $g(x)$  which occur to the first power. Call this factor  $Sim(g(x))$ . The repeated factor of  $g(x)$  is defined to be  $Rep(g(x)) = g(x)/Sim(g(x))$ . For example, if  $g(x) = (1+x)^2(1+x+x^2)(1+x+x^3)$  then  $Sim(g(x)) = (1+x+x^2)(1+x+x^3)$  and  $Rep(g(x)) = (1+x)^2$ . When factoring  $t_i(x)$  and  $r_i(x)$  in Part B we first factor  $Sim(t_i(x))$  and then  $Sim(r_i(x))$ . If either one contains an irreducible factor of degree larger than  $b$  then, of course, no further factoring is necessary. It is a relatively simple matter to compute  $Sim(g(x))$  and look for its largest irreducible factor first. (The Berlekamp matrix technique for factoring cannot distinguish between polynomials which are irreducible and those which are powers of such. That is why  $sim(g(x))$  and  $rep(g(x))$  are computed.)

Part B of the algorithm is based on the tendency for polynomials to factor into irreducible factors of small degree. The asymptotic complexity of the algorithm for  $GF(2^n)$  (the case we are most interested in) has been computed by A. Odlyzko [10]. If  $p(n,k)$  is the probability that a polynomial of degree  $n$  over  $GF(2)$  factors into irreducible factors all of whose degrees are less than or equal to  $k$  then  $p(n,k)$  is asymptotically bounded below by

$\left(\frac{e}{n}\right) \frac{n}{k}^{(1+o(1))}$  and above by  $\left(\frac{2e}{n}\right)^k$  as  $n \rightarrow \infty$  and  $\frac{n}{2k} \rightarrow 0$ . Based on this, the random time complexity is  $R4\sqrt{cn\log n}$ , where  $R$  depends on the factoring algorithm employed.

### 5.2 Results in $GF(2^{61})$ .

We have implemented the algorithm of the previous section and tested it in the field  $GF(2^{61})$  generated by  $f(x) = x^{61} + x^5 + x^2 + x + 1$ . The program is in Fortran 77 with C subroutines and is running on a general purpose VAX11/780.

In table 1 we list the probability  $P(k)$  that two polynomials, each of degree at most 30, factor into irreducible factors all of degree at most  $k$ .  $S(k)$  is the number of irreducible polynomials of degree at most  $k$ . Table 1 includes  $S(k)/P(k)$  which is the expected number of passes of Part B in order to generate  $S(k)$  random linear equations in the  $S(k)$  unknowns which are the logs of the irreducible polynomials of degree at most  $k$ . For this case, we see that  $k = 12$  minimizes the number of expected runs to produce a database. Hence, we selected  $b = 12$  for the algorithm as applied to this field. This value of  $b$  requires a database consisting of 746 logarithms. The polynomials and their logs are given in Appendix A and each appears in octal form. For example, number 10 on the list is

57 1 7 0 4 1 2 1 5 4 2 3 2 6 2 5 2 3 7 6 0 3

The octal number 57 represents 1 0 1 1 1 1 and is the polynomial  $x^5 + x^3 + x^2 + x + 1$ . The binary representation of the logarithm is

1 1 1 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 1 0 1 1 0 0 0 1 0 0 1 1  
0 1 0 1 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 0 0 0 1 1

and the decimal representation of the logarithm is

2,171,194,306,592,735,107.

Rather than solve a system of 746 equations in 746 unknowns over  $GF(2^{61}-1)$  we decide to initially set  $b = 10$  which requires more runs to get the equations but the resulting system has only 226 unknowns. Having done this we used the algorithm to build the database up to  $b = 12$ . We estimate that such a database can be constructed in under two hours using our present

CANADIAN PROBABILITIES FOR GF(2\*\*61).

K	P(K)	S(K)	EXPECTED # OF RUNS
1	5.33462163E-14	1	1.87454719E+13
2	1.76871018E-12	2	1.13076751E+12
3	1.42920165E-10	4	2.79876531E+10
4	6.35613994E-09	7	1.10129734E+09
5	3.22381391E-07	13	40324908.2
6	5.33811982E-06	22	4121301.27
7	6.74428582E-05	40	593094.674
8	4.2333643E-04	70	165353.121
9	1.86019133E-03	126	67734.9679
10	5.77168417E-03	225	38983.4221
11	.0144081649	411	28525.4925
12	.0293335145	746	25431.6611
13	.0528409512	1376	26040.4094
14	.085727823	2537	29593.6595
15	.128359867	4719	36763.8274
16	.176843304	8799	49755.9127
17	.229773022	16509	71849.166
18	.285984158	31041	108540.977
19	.345031001	58635	169941.251
20	.40617628	111012	273309.904
21	.469073093	210870	449546.143
22	.533239803	401427	752807.645
23	.598365862	766149	1280402.26
24	.664015691	1465019	2206301.78
25	.729737155	2807195	3846857.71
26	.794775116	5387990	6779263.58
27	.857890338	10358998	12074967.6
28	.916752227	19945393	21756579.8
29	.966945476	38458183	39772855.8
30	.999999999	74248450	74248450.1

Table 1

routines.

The algorithm was run on 500 polynomials. The results of this sample are displayed in Figure 1.

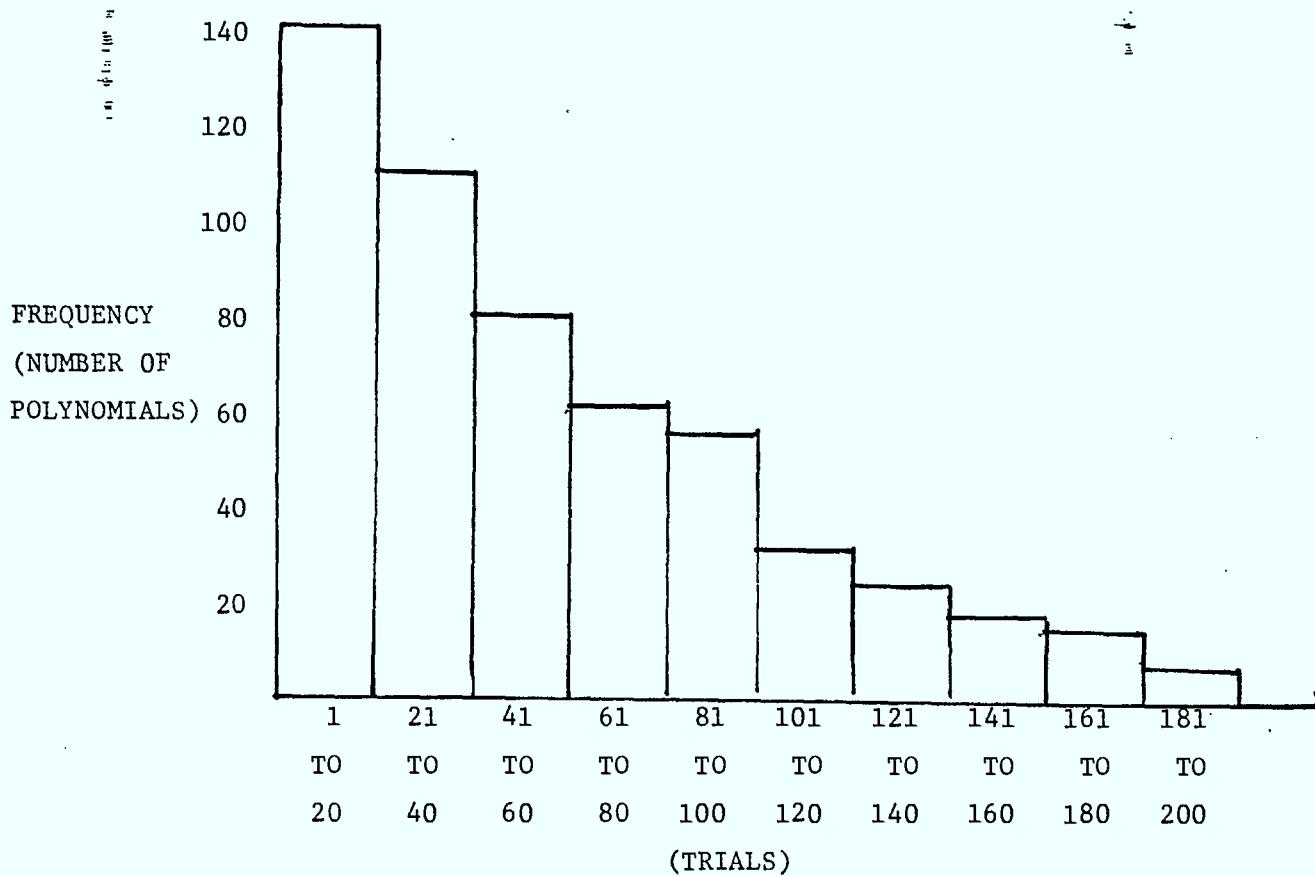


Figure 1.

Part B of the algorithm was modified slightly. Instead of generating a random integer  $a_i$  and computing  $x^{a_i} b(x)$  (we found that this was, in fact, slowing the program down) we computed  $(1+x+x^60)^{a_i} b(x)$ . We knew the logarithm of  $1 + x + x^{60}$  and this did not seem to affect the randomness we require. For the 500 polynomials we tested, the average number of passes through Part B of the algorithm was 56.01 with an average time to compute a log of under 5 seconds.

#### 6. Adleman's Algorithm

The best algorithm for computing discrete logarithm which appears in the literature is one due to L. Adleman [1]. For comparison purposes we describe the algorithm here. The algorithm is subexponential in the log of the number of elements in the field and random time.

Let  $p_1(x), p_2(x), \dots, p_t(x)$  be all irreducible polynomials of degree at most  $b$  over  $GF(p)$ . Define a polynomial  $a(x)$  to be smooth if

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

With each smooth polynomial  $a(x)$  associate a vector  $\overline{a(x)} = (e_1, e_2, \dots, e_t)$ .

We will only consider the algorithm in  $GF(2^n)$  where  $2^n - 1$  is a prime. Let  $a$  be a generator in  $GF(2^n)$ .

THE ALGORITHM. Let  $b(x)$  be an arbitrary nonzero element in  $GF(2^n)$  and let  $b$  be some fixed positive integer determined by  $n$ .

1. Generate smooth elements  $\overline{ba^s}, \overline{a^s}, \overline{a^s}, \dots, \overline{a^s}$  so that  $\overline{ba^s}$  is linearly dependent on  $\overline{a^s}, \overline{a^s}, \dots, \overline{a^s}$ . That is,

$$\overline{ba^s} = \sum_{i=1}^{\ell} \lambda_i \overline{a^s}$$

$$2. \log_a b = -s + \sum_{i=1}^{\ell} \lambda_i s_i.$$

This algorithm, as with the one of the previous section, requires factoring polynomials. It also requires that one solve a linear system of equations over  $GF(2^n - 1)$ . In practice it is likely better to compute the logarithms of all the  $p_i(x)$ ,  $1 \leq i \leq t$ , by initially solving a linear system of equations. Step 1 would then require that we find a smooth  $ba^s$  for some  $s$ . This algorithm and ours are very similar and essentially only differ in the use of the Euclidean algorithm.

Suppose we consider the expected number of runs to produce the database for this algorithm in the field  $GF(2^{61})$ . Table 2 lists probabilities and expected trials. The optimal database for this algorithm has  $b = 15$  and the expected number of runs is 628284. The algorithm of section 4 is about 25 times more efficient in constructing a database. Also, factoring two polynomials of degree at most 30 can be done more efficiently than factoring one of degree 60.

ADLEMAN PROBABILITIES FOR GF(2\*\*61).

K	P(K)	S(K)	EXPECTED # OF RUNS
1	8.20090524E-16	1	1.21937758E+15
2	8.81933415E-15	2	2.2677449E+14
3	2.4088587E-13	4	1.66053741E+13
4	6.16697535E-12	7	1.13507832E+12
5	2.87534365E-10	13	4.52119871E+10
6	6.49386181E-09	22	3.38781462E+09
7	1.45140186E-07	40	275595623
8	1.6240426E-06	70	43102317.6
9	1.2624515E-05	126	9980581.43
10	6.39169698E-05	225	3520191.91
11	2.48450225E-04	411	1654254.89
12	7.38423665E-04	746	1010260.15
13	1.84778808E-03	1376	744674.141
14	3.94471792E-03	2537	643138.509
15	7.51093409E-03	4719	628284.038
16	.0129724298	8799	678284.649
17	.0207338898	16509	796232.649
18	.0310329059	31041	1000260.82
19	.0441310341	58635	1328656.83
20	.0601228451	111012	1846419.61
21	.0789028551	210870	2672526.87
22	.100132689	401427	4008950.56
23	.12355915	766149	6200665.83
24	.148935705	1465019	9836586.87
25	.176080582	2807195	15942672.2
26	.204816723	5387990	26306396.9
27	.235001922	10358998	44080482
28	.266478225	19945393	74848115.6
29	.299022554	38458183	128612984
30	.332077077	74248450	223588002

Table 2

### 7. Results in $GF(2^{127})$

We are currently constructing the database for the finite field  $GF(2^{127})$  generated by  $f(x) = x^{127} + x + 1$ . In Tables 3 and 4 we list the probabilities and expected number of runs for our algorithm and the Adleman algorithm respectively. The optimal value of  $b$  for our algorithm is 20. This value requires that we compute a database containing 111,012 logs. Although this number is manageable, we have decided on  $b = 17$  in order to reduce the size of the linear system without increasing the expected number of runs drastically. The optimal value for the Adleman algorithm is  $b = 23$  with an expected number of runs of 5,549,006,000. In terms of expected runs the Adleman algorithm is about 100 times less efficient. This does not take into account the time required to factor.

Presently, our algorithm averages one trial in .86 seconds on the VAX 11/780. To produce enough random equations to construct the database would require about 2.28 years of machine time. This particular field exhibits a weakness which will considerably shorten the time required to get the necessary equations.

We define the square orbit of an element  $\beta$  in  $GF(2^n)$  to be the set

$$SO(\beta) = \{\beta^{2^s} : 0 \leq s \leq n-1\}.$$

If  $\alpha$  is the generator of the field (the log base) employed and  $SO(\alpha)$  contains polynomials of "low" degree then this representation of the field is said to exhibit orbital weakness. Orbital weakness can be exploited to get many equations for the database and reduced the number of trials.

In  $GF(2^{127})$  generated by  $f(x) = x^{127} + x + 1$  with log base  $\alpha$ ,  $f(\alpha) = 0$ , we have that

$$SO(\alpha) = \text{span}\{\alpha^2, \alpha^4, \dots, \alpha^{64}\}.$$

This is easily seen from the fact that for  $i \geq 7$

$$\alpha^{2^i} = (\alpha^2)^{2^{i-7}} = (\alpha^2 + \alpha)^{2^{i-7}} = \alpha^{2^{i-6}} + \alpha^{2^{i-7}}.$$

Since every element in  $SO(\alpha)$  is  $\alpha^{2^i}$  for some  $i$  then the logs of all elements in  $\text{span}\{1, \alpha, \alpha^2, \alpha^4, \dots, \alpha^{64}\}$  are also known. This follows since

$$(1+\alpha^{2^i}) = (1+\alpha)^{2^i} = \alpha^{(127)2^i}.$$

CANADIAN PROBABILITIES FOR GF(2\*\*127).

K	P(K)	S(K)	EXPECTED # OF RUNS
1	1.27141469E-32	1	7.8652544E+31
2	1.61023467E-30	2	1.24205499E+30
3	1.42003032E-27	4	2.81684126E+27
4	1.15313844E-24	7	6.07038995E+24
5	3.46321796E-21	13	3.75373429E+21
6	2.43083741E-18	22	9.05037906E+18
7	1.75009226E-15	40	2.28559379E+16
8	2.99279106E-13	70	2.33895379E+14
9	2.39477444E-11	126	5.26145586E+12
10	7.73424039E-10	225	2.90914154E+11
11	1.42517804E-08	411	2.88385022E+10
12	1.48157461E-07	746	5.03518348E+09
13	1.0660927E-06	1376	1.29069451E+09
14	5.46899677E-06	2537	463887639
15	2.19028276E-05	4719	215451634
16	7.12191038E-05	8799	123548311
17	1.96662481E-04	16509	83945854.4
18	4.71015488E-04	31041	65902291.5
19	1.00764675E-03	58635	58190035.4
20	1.96260912E-03	111012	56563479.1
21	3.54177251E-03	210870	59537985.4
22	5.96621694E-03	401427	67283339.5
23	9.45338803E-03	766149	81044911.9
24	.0142135998	1465019	103071637
25	.0204564492	2807195	137227872
26	.0283794933	5387990	189855046
27	.0381757153	10358998	271350462
28	.0500248252	19945393	398709899
29	.0640949758	38458183	600018684
30	.0805164802	74248450	922152208

Table 3

ADLEMAN PROBABILITIES FOR GF(2\*\*127)

DEGREE	PROBABILITY	TOTAL IRREDUCIBLES	EXPECTED RUNS
1	0.0000000	1	*****
2	0.0000000	2	*****
3	0.0000000	4	*****
4	0.0000000	7	*****
5	0.0000000	13	*****
6	0.0000000	22	*****
7	0.0000000	40	*****
8	0.0000000	70	*****
9	0.0000000	126	*****
10	0.0000000	225	*****
11	0.0000000	411	*****
12	0.0000000	746	*****
13	0.0000000	1376	*****
14	0.0000000	2537	677227100000.0000000
15	0.0000000	4719	188825700000.0000000
16	0.0000001	8799	69154690000.0000000
17	0.0000005	16509	31488570000.0000000
18	0.0000018	31041	17251720000.0000000
19	0.0000033	58635	11004140000.0000000
20	0.0000139	111012	7988899000.0000000
21	0.0000326	210870	6470914000.0000000
22	0.0000697	401427	5756010000.0000000
23	0.0001381	766149	5549006000.0000000
24	0.0002554	1465077	5737345000.0000000
25	0.0004452	2807173	6304890000.0000000
26	0.0007375	5387893	7305424000.0000000
27	0.0011680	10358933	8868630000.0000000
28	0.0017773	19946640	11222780000.0000000
29	0.0026095	38457232	14737250000.0000000
30	3.3185230	-2073236992	-624746700.0000000

Table 4.

THEOREM 7.1. Let  $f(x)$  be an irreducible polynomial of degree  $n$  over  $F = GF(q)$ . Let  $g(x)$  be an arbitrary polynomial over  $GF(q)$ . Let  $m(x)$  be any divisor of  $f(g(x))$ . Then the degree of  $m(x)$  is a multiple of  $n$ .

Proof. It is sufficient to prove the result for irreducible factors  $m(x)$ . Let  $m(x)$  be a factor of  $h(x) = f(g(x))$ , let  $K$  denote the splitting field of  $h(x)$  and let  $\alpha$  be a root of  $m(x)$  in  $K$ . Let  $L$  denote the subfield of  $K$  induced by  $\alpha$ . That is,  $L$  is the splitting field of  $m(x)$ . Since  $m(x) | h(x)$ , we have  $h(\alpha) = f(g(\alpha)) = 0$  so that  $g(\alpha)$  lies in the splitting field of  $f(x)$ , which is  $R = GF(q^n)$ . Also  $g(\alpha)$  lies in no proper subfield of  $GF(q^n)$  since all zeroes of  $f(x)$  generate this field. Since  $g(\alpha)$  lies in  $L$  then  $R \subset L$  and, therefore,  $n$  divides  $\dim[L:GF(q)]$ . Since  $m(x)$  is irreducible and  $L$  is its splitting field, we have that

$$\deg m(x) = \dim[L:GF(q)]$$

and the result follows.

The following example illustrates the use we make of orbital weakness and Theorem 7.1.

EXAMPLE. In  $GF(2^{127})$  generated by  $f(x) = 1 + x + x^{127}$  we know that  $x^2 = x + x^2$ . Consider the irreducible quintic  $g(x) = 1 + x^2 + x^5$ . Now,

$$\begin{aligned} g(x^{2^7}) &= g(x+x^2) \\ &= 1 + x^2 + x^4 + x^5 + x^6 + x^9 + x^{10} \\ &= (1+x^2+x^3+x^4+x^5)(1+x^3+x^5). \end{aligned}$$

But  $g(x^{2^7}) = [g(x)]^{2^7}$  and, thus,

$$2^7 \log g(x) = \log p_1(x) + \log p_2(x)$$

where

$$\begin{aligned} p_1(x) &= 1 + x^2 + x^3 + x^4 + x^5 \\ p_2(x) &= 1 + x^3 + x^5. \end{aligned}$$

This gives us a linear equation in the logs of polynomials of low degree.

Equations generated as in this example are referred to as systematic. The system of linear equations we require to construct the database in  $GF(2^{127})$  cannot be made up entirely of systematic equations. It is, however, very likely that a large portion of the equations can be systematic with the remaining equations being generated at random or by other means. We have only begun to generate systematic equations. There are 226 irreducible polynomials of degree at most 10. We have generated 133 linearly independent systematic equations. There are 126 irreducible polynomials of degree at most 9 and we have 90 linearly independent systematic equations. Another method for generating equations is what we call weight manipulation. It appears to be a powerful technique for giving linear equations. We have, as yet, not implemented it and so will not discuss it further.

8. Representations of  $GF(p^n)$ ,  $n \geq 2$ .

It is well known that any two finite fields with  $p^n$  elements are isomorphic. Let  $f(X)$  and  $h(X)$  be distinct irreducible polynomials of degree  $n$  over  $GF(p)$ . Let  $\mathbb{F}_1$  and  $\mathbb{F}_2$  be the finite fields with  $p^n$  elements generated by  $f(X)$  and  $h(X)$  respectively.  $\mathbb{F}_1$  and  $\mathbb{F}_2$  are said to be distinct representations of  $GF(p^n)$ . The algorithm of section 4 requires that a database of logarithms be constructed. These logarithms are given with respect to a given representation of the field. If the field representation is changed we can only compute logarithm in the new system if

1. a new database is constructed

or

2. the isomorphism between the two representations is established.

To make 2. more clear, consider the following. Let  $\alpha$  be a primitive element for a field  $\mathbb{F}_1$  of order  $p^n$  and let  $\beta$  be a primitive element for a field  $\mathbb{F}_2$  of order  $p^n$ . Suppose that a good algorithm for computing logarithms to the base  $\alpha$  in  $\mathbb{F}_1$  is known. Given an element  $X \in \mathbb{F}_2$  such that we require its logarithm to the base  $\beta$  we could do the following. Let  $m_\beta(X)$  be the minimum polynomial of  $\beta$  and let  $\alpha^i$  be a root of  $m_\beta(X)$  in  $\mathbb{F}_1$ . This polynomial can be used to define a field isomorphism

$$\sigma: \mathbb{F}_2 \rightarrow \mathbb{F}_1 \text{ where } \sigma(\beta^j) = (\alpha^i)^j, 0 \leq j \leq p^n - 2,$$

$\sigma$  can be represented by a linear transformation  $A$  between the bases  $B_1 = \{1, \beta, \beta^2, \dots, \beta^{n-1}\}$  and  $B_2 = \{1, \alpha, \alpha^2, \dots, \alpha^{n-1}\}$ . Now, for  $X \in \mathbb{F}_2$ , write  $X$  with respect to the basis  $B_1$  and denote it  $X_\beta$ . Thus,  $AX_\beta \in \mathbb{F}_1$ . Computing the logarithm of this element gives  $AX_\beta = \alpha^t$ , for some integer  $t$ , or  $X_\beta = A^{-1}\alpha^t = (A^{-1}\alpha)^t = \beta^{-it}$ . Hence,  $\log_\beta X = -i \log_\alpha AX_\beta$ .

There seems to be no general procedure for efficiently determining the isomorphism between two finite fields. The following result, which to the authors knowledge was first pointed out by J. Sacks [13], gives a useful result in this direction. It requires the use of  $p$ -polynomials or linearized polynomials.

A polynomial  $L(X)$  over  $GF(p)$  is said to be a *linearized polynomial* if and only if

$$L(X) = \sum_{i=0}^t \lambda_i X^{p^i}.$$

A polynomial  $A(X)$ , over  $GF(p)$  is said to be an *affine polynomial* iff  $A(X) = L(X) - a$  where  $L(X)$  is a linearized polynomial and  $a \in GF(p)$ .

For a thorough treatment of linearized and affine polynomials the reader is referred to the book by E. Berlekamp [2].

Affine polynomials have the nice property that their roots in  $GF(p^n)$  can be determined by solving a linear system of equations over  $GF(p)$ .

Suppose we have two fields  $\mathbb{F}_1$  and  $\mathbb{F}_2$  of order  $p^n$ . These fields are generated by  $f(X)$  and  $h(X)$  respectively. If we can find a root of  $f(X)$  in the field  $\mathbb{F}_2$  then isomorphism is easily established. Suppose that  $f(X)$  divides an affine polynomial  $a(X)$ . Then the roots of  $f(X)$  are among the roots of  $a(X)$ . We can find the roots of  $a(X)$  in  $\mathbb{F}_2$  by solving a linear system of equations over  $GF(p)$ . By substituting each of these roots into  $f(X)$  we can determine a root of  $f(X)$  in  $\mathbb{F}_2$ . If the degree of  $a(X)$  is not too large then

this method is feasible. For example, in  $GF(2^{127})$  generated by  $f(X) = X^{127} + X + 1$  we see that  $a(X) = X^{128} + X^2 + X$  is an affine polynomial which  $f(X)$  divides. Since the degree of  $a(X)$  is not large, we can determine isomorphism between any two finite fields of order  $2^{127}$  efficiently. The example we cited earlier of  $GF(2^{61})$  generated by  $f(X) = X^{61} + X^5 + X^2 + X + 1$  is somewhat different. The affine polynomial of least degree which  $f(X)$  divides has order  $2^{60}$ . This, of course, does not mean that isomorphism cannot be done efficiently. We need to find an irreducible polynomial of degree 61 which divides an affine polynomial of not too large a degree. We have not made a serious attempt to find such a polynomial.

#### 9. Conclusion

The results contained in this report establish that  $GF(2^{127})$  is not secure for a PKDS. It is our opinion that a PKDS is still a viable technique and still requires much consideration.

#### References

- [1] L. Adleman, *A superexponential algorithm for the discrete logarithm problem with applications to cryptography*, Proc. IEEE 20th Annual Symposium on Foundations of Computer Science (1979) 56-60.
- [2] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill Publishing Co. (1968).
- [3] I.F. Blake, R. Fuji-Hara, and S.A. Vanstone, *Finite field techniques for shift registers and encipherment*, Research Report 106-16, Department of Communication.
- [4] E.R. Berlekamp, *Factoring polynomials over large finite fields*, Mathematics of Computation Vol. 24 (1970) 713-735.
- [5] D.G. Cantor and H. Zassenhaus, *A new algorithm for factoring polynomials over finite fields*, Mathematics of Computation, Vol. 36 No. 154 (1981) 587-592.
- [6] Witfield Diffie and Martin E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory Vol. IT-22 No. 6 (1976).
- [7] R.J. Lipton and A. Wigderson, *Multi-party cryptographic protocols (extended abstract)*, preprint.

- [8] R.J. McEliece, *The Theory of Information and Coding: A Mathematical Framework for Communications*, Addison Wesley, Reading, Mass. (1977).
- [9] R.T. Moenck, *On the efficiency of algorithms for polynomial factoring*, Mathematics of Computation Vol. 31, No. 137 (1977) 235-250.
- [10] A. Odlyzko, (Private communication).
- [11] S.C. Pohlig and M.E. Hellman, *An improved algorithm for computing logarithms over GF( $p$ ) and its cryptographic significance*, IEEE Transactions on Information Theory Vol. IT-24, No. 1 (1978).
- [12] M.O. Rabin, *Probabilistic algorithms in finite fields*, SIAM J. Comput. Vol. 9, No. 2 (1980) 273-280.
- [13] J. Sacks, (Private Communication).
- [14] B. Shanning, *Data encryption with public key distribution*, Proc. EASCON (1979).
- [15] G.J. Simmons, *Secure communications and asymmetric cryptosystems*, AAAS Selected Symposium 69, Westview Publishing (1982).

APPENDIX A

Mar 21 15:19 1983 1090ct.746 Page 1

1	00003	010316066404231475500	57	00615	025774452403156514571
2	00007	017520407004732423076	58	00637	023053567302320125625
3	00013	157143622767314604674	59	00643	022572753152352770406
4	00015	032270127304724737712	60	00651	10265731777241437553
5	00023	136605717173772731555	61	00661	177051600550611234473
6	00031	112025170025776617606	62	00675	052634537556247145033
7	00037	035155773453253363572	63	00703	052275261035420270252
8	00045	170237001432145267663	64	00717	167630707640746224033
9	00051	117120362671665516740	65	00727	076402237745037503543
10	00057	170412154232625237603	66	00735	041530110710515714740
11	00067	006704664705477222615	67	00747	170716052160205300674
12	00073	152023761473523203223	68	00763	176467244241533156361
13	00075	163422201631020127150	69	00765	005037002044575445344
14	00103	055743220313560727413	70	00771	173627736110544104024
15	00111	174316240620662166537	71	01003	131542245577422525050
16	00127	051712125204554664374	72	01021	044437221402172412370
17	00133	136470333355543560464	73	01027	170731051775110401752
18	00141	174353441147425731122	74	01033	122041621367400264176
19	00147	136405757302321502123	75	01041	173646700766343330073
20	00155	025572021424606370213	76	01055	156471623352644775161
21	00163	160112033003341106051	77	01063	004267062041242444712
22	00165	070022357535246056423	78	01113	176577722767611630255
23	00203	071423470125567303036	79	01131	055447277530575470737
24	00211	040132211332431533654	80	01137	000022003664401417400
25	00217	002644226650632675321	81	01145	077416036434510377613
26	00221	000767544273600061613	82	01151	032746736424213541204
27	00235	155266664645776661020	83	01157	102616563173742520247
28	00247	142403014602433242112	84	01167	164152044577634601265
29	00253	11437444073414202623	85	01175	161570450446733656276
30	00271	115237547042317102043	86	01207	140655072055355360003
31	00277	172445622162047650147	87	01225	161514267270364666513
32	00301	123147423427413555231	88	01231	130660031715675152357
33	00313	006044257356175277461	89	01243	077577725037065401157
34	00323	063450162612014577672	90	01245	045234316652120214730
35	00325	003271061445605660052	91	01257	024266574055641130556
36	00345	170405152322177535344	92	01267	056020407332310123155
37	00357	106663707126110465751	93	01275	155455160461202156402
38	00361	030314347051325725713	94	01317	026163436572646325443
39	00367	100377371266552444040	95	01321	070120323550044451016
40	00375	156765724131522365731	96	01333	172264245104316542011
41	00433	052011057261556224312	97	01365	171107616233162361274
42	00435	051562037145116730574	98	01371	101154432337174004661
43	00453	047112343753120312151	99	01401	065576740357266262607
44	00455	167303753543103316101	100	01423	071152202404311457610
45	00471	031606640423147550004	101	01425	051557506575673263305
46	00477	144303477610611275753	102	01437	140044740161762347300
47	00515	174014554322705275063	103	01443	020513054175477145313
48	00537	027007203033225641336	104	01461	162251703151627213736
49	00543	135167014577124132633	105	01473	135743752204464740271
50	00545	123246317136557740443	106	01511	103226301610213651026
51	00551	042424023053364612113	107	01517	110145566501133413711
52	00561	134364327400556040125	108	01533	025034431263166711327
53	00567	123016653077743727614	109	01541	126547054540172476473
54	00573	154702510642571426302	110	01553	050207366132436046314
55	00607	154612045141652770367	111	01555	056747422065325431227
56	00613	160001346317201145150	112	01563	062366076316667076252

Mar 21 15:19 1983 losoct.746 Page 2

113	01577	060637725746045513525	169	02671	031754031012612037772
114	01605	074157247574042560023	170	02701	025637116225676200625
115	01617	050062233576763216025	171	02707	007122146245073064336
116	01641	103542742037154136105	172	02745	062175605707022352576
117	01665	023040622336305375510	173	02767	123231237524044074002
118	01671	010524250200402532464	174	02773	024574275035366631703
119	01707	004155644255416073551	175	03023	074032661131312354555
120	01713	123163001562727352046	176	03025	140057005236226460540
121	01715	125543254266233276453	177	03043	113400175254426061500
122	01725	102356124421535171627	178	03045	003462300160510663762
123	01731	122305167541372116276	179	03061	003073335450522302035
124	01743	075361135103202537055	180	03067	106565557716563032110
125	01751	035014553277415073476	181	03103	003521641662656067103
126	01773	174727600171404046714	182	03117	037617217672676220172
127	02011	157642753165630501655	183	03121	125144652025510743043
128	02017	157526365037510206513	184	03133	163014324755251222140
129	02033	016031213450753766071	185	03171	161062272535263064026
130	02035	171752363564775000526	186	03177	167135332360611010000
131	02047	053075511600036361575	187	03205	154142774634220130201
132	02055	154430717276014602577	188	03211	175276015740621030075
133	02065	004643576222372704201	189	03247	047537655635356530453
134	02107	057413046751103454720	190	03255	112033201070766265015
135	02123	035720316766354436742	191	03265	005433205564721457150
136	02143	076730324211516723057	192	03277	011247650662652412365
137	02145	163635376471325562125	193	03301	127756376007605724217
138	02157	164542313642647332004	194	03315	00600166061172455171
139	02201	025136304212406004741	195	03323	102732044600627242747
140	02213	021643523167450025342	196	03337	055476441277576054523
141	02231	002136022304766654070	197	03367	003276073576125276430
142	02251	157171163431522477054	198	03375	150071751437222532607
143	02257	115616724400476563323	199	03417	111635432306010221361
144	02305	014213431740031230224	200	03421	071477501511234654110
145	02311	157701374042667231722	201	03427	154753006407102441624
146	02327	060173640773554060140	202	03435	046045252024760173034
147	02347	154547214715024335233	203	03441	155222656576176530316
148	02355	103066255731527672705	204	03453	007372331545465460325
149	02363	123507252224545767772	205	03465	064307327771664714065
150	02377	046750616617372245335	206	03471	063010256725057740344
151	02413	057373671337473371671	207	03507	113073532321164756130
152	02415	052511163461624640517	208	03515	162433046350562656605
153	02431	021371322065354143555	209	03525	103774514746010623455
154	02437	050521512163376062060	210	03531	143550417321647712755
155	02443	073757137734547032640	211	03543	120404140611263006132
156	02461	050770357710705531072	212	03573	025122436340245507362
157	02475	105424642110007400324	213	03575	111142037617433552217
158	02503	153227015357732115354	214	03601	023244724651100030053
159	02527	1123334517723550466515	215	03607	113756770247151212141
160	02541	17220772000070755576	216	03615	07532066077314125711
161	02547	004502545272417613526	217	03623	023177115421507754401
162	02553	117414771046777531413	218	03651	117553624101554364056
163	02605	006335012400176641324	219	03661	107262425372460127171
164	02617	014716114400524623047	220	03705	067373363430307136507
165	02627	013246641455141256002	221	03733	072677767330027253163
166	02633	037671556577264575537	222	03753	171573521750326645574
167	02641	151465340413514563167	223	03763	060050221440120501536
168	02653	055605257244426536541	224	03771	006020754552601055737

225	03777	06167674727766651346	281	05205	113551636454166034662
226	04005	160670650351725673727	282	05221	014064746520105112401
227	04027	056133171710115211473	283	05235	004265035635412364724
228	04053	17374170553772620134	284	05247	121613442056572612623
229	04055	106755370764712626213	285	05253	073400524245207635170
230	04107	106276645113032520661	286	05263	017045372477041721315
231	04143	125051411100225350564	287	05265	156452056500751073552
232	04145	024446044016723140556	288	05325	065712300643067246603
233	04161	017412253614446725776	289	05337	006003424100262673705
234	04173	102567236207742275203	290	05343	045337254214104225014
235	04215	151423242477325414231	291	05351	005324701574362611077
236	04225	174635772037710054037	292	05357	136164273305550156212
237	04237	124744472556343700515	293	05361	027771705757204130634
238	04251	025577554615064065631	294	05373	106545110234767041775
239	04261	177662163161670246734	295	05403	157352631261423266362
240	04303	055111521615744073353	296	05411	001443601215507600451
241	04317	032756703254233175304	297	05421	105476454536036743326
242	04321	023211512143025770373	298	05463	075010524453103205466
243	04341	140736432130610613546	299	05477	107134526270670517157
244	04347	132412654175123636241	300	05501	077002025637753316075
245	04353	043457607451216271303	301	05513	154242540177171300216
246	04365	167321113356352755674	302	05531	032473303736370736705
247	04415	034014357110534657112	303	05537	045144337535466560313
248	04423	043215460320017211370	304	05545	020750375046125611066
249	04445	046532725247754265744	305	05557	014614563557130571631
250	04451	136644144775022442161	306	05575	166525413226251324162
251	04467	067146205456567141603	307	05607	057357400136545126210
252	04473	133534563360762132531	308	05613	102257472720772336163
253	04475	073330330270641106264	309	05623	123063417233070240760
254	04505	053162533267725126321	310	05625	075676271720110517710
255	04511	034063707553464405754	311	05657	01155211420227361536
256	04521	107351722111367241347	312	05667	156503777736457732503
257	04533	140146545011706272054	313	05675	071372026063615701756
258	04563	024252123262653341014	314	05711	130043233304125422005
259	04565	052332777346704424062	315	05733	154731222543431770400
260	04577	075403702501225234172	316	05735	164752627436423613412
261	04603	154453711222661600546	317	05747	032567557471010023607
262	04617	040770270356372560660	318	05755	051027067244366123215
263	04653	072056231757517633465	319	06013	107071710006511745735
264	04655	017260102323503001711	320	06015	025106155170571552200
265	04671	101264651530232436360	321	06031	155733215331574707011
266	04707	154764344641273767527	322	06037	042672532264457353430
267	04731	053432046510310156054	323	06061	121377043073252673235
268	04745	025717546563650215311	324	06127	100205344142260431345
269	04757	134023104300474172325	325	06141	142127137427077567000
270	04767	134015615112745725015	326	06153	006261246527603360564
271	05001	146374407004341431371	327	06163	177255107566176414163
272	05007	103341202466027715146	328	06165	111155446027301033107
273	05023	071613635476606270204	329	06205	015665145447622426345
274	05025	025305173030264142107	330	06211	005523005136125465465
275	05051	042634755456335306315	331	06227	136277135472204764046
276	05111	073637765440512233365	332	06233	127671654612070112316
277	05141	134574255127755010561	333	06235	130501521275367342271
278	05155	104632212551626207542	334	06263	125300071307722466614
279	05171	051637317417615102150	335	06277	103236235637340611301
280	05177	062353443111172457153	336	06307	022722726403064043262

337	06315	000670405272432250676	393	07461	037222777066100337700
338	06323	107246411003622754544	394	07467	007005560741601616202
339	06325	032732015752751037455	395	07535	036332346420551621117
340	06343	060134724036474226354	396	07553	066625200305276757636
341	06351	064616370403331276073	397	07555	002451336317360326441
342	06367	112762521360772211674	398	07565	131070442440534135537
343	06403	006456315730150316565	399	07571	151066326351531113158
344	06417	176675707722506743431	400	07603	045551674474732411634
345	06435	010410644653722667065	401	07621	007305471424206330377
346	06447	146371101137426646654	402	07627	057133102205603650401
347	06455	037432225630335310547	403	07633	127434753211665303550
348	06501	035060365735301370066	404	07647	166213302003443650156
349	06507	104340164041602650616	405	07655	106071004217267647625
350	06525	133637541557504274373	406	07665	043335043154321413331
351	06531	147517126626037601731	407	07715	033002543773534252057
352	06543	075151571567661744454	408	07723	112263363737400737475
353	06557	005305327635257430304	409	07745	103100767146727467554
354	06561	050354534276144140650	410	07751	044074057507267572567
355	06623	171361371033720414520	411	07773	064072566240354477715
356	06637	032457773313670317100	412	10011	143044000334404250517
357	06651	075756235254704461331	413	10027	003677441411525367014
358	06673	063160647264714322052	414	10041	156615340322631045210
359	06675	106222512501402750600	415	10063	013300221553555774441
360	06711	120571611703250002044	416	10065	061043467562753205421
361	06727	120375572224121515632	417	10077	011132561740745742477
362	06733	067510241577172211077	418	10115	102027201533551401467
363	06741	056320002005170533456	419	10123	176763037242334543217
364	06747	076705051550553111701	420	10151	172003347651463445473
365	06765	073256433435244617353	421	10167	127373715221232640541
366	06777	166365426550775124077	422	10173	030672317440265324223
367	07005	040240051330336715250	423	10175	114600674236137716646
368	07035	137476240460303740064	424	10201	144267471147037047743
369	07041	022212306600567645031	425	10213	125711200330354441671
370	07047	106613063227211176501	426	10231	050046757733543446477
371	07053	007317751532342226110	427	10243	015131537164306003476
372	07063	001723612022125743670	428	10245	146667237667420773603
373	07071	116445773305535601652	429	10317	177534163567766776336
374	07107	070046307451011347347	430	10321	146404541536255250477
375	07113	131701026460024302531	431	10353	066502025312201467073
376	07125	041541033276442504644	432	10355	013125224500417500267
377	07137	004617275613470715420	433	10377	074752176545060257304
378	07161	123117340263007325466	434	10407	074175062534443435764
379	07173	014002116320662674117	435	10437	070335506262042247726
380	07175	140743623624357123372	436	10443	132126741740502430256
381	07201	073275061742050230777	437	10461	064026747237451672626
382	07223	040156163675272315464	438	10467	010224443140543704701
383	07237	006453664335420545111	439	10473	171104567525050720643
384	07243	071772415011112140133	440	10517	030404466542507044730
385	07273	152403101623660741456	441	10527	161461553537006150077
386	07311	110460616602063662672	442	10541	112632036572364043271
387	07317	134720750071440531160	443	10553	007370450232156213705
388	07335	041354406402565744124	444	10555	056207413661427037134
389	07363	145154071657773442403	445	10571	054173745044007115264
390	07371	125372317474424153700	446	10603	015242006405307432645
391	07413	13623573162117773467	447	10605	167233711631215637452
392	07431	025265027601726601204	448	10621	100666434273606072474

Mar 21 15:19 1983 losoct.746 Page 5

449	10653	106143445364675236075	505	12147	160665431212455036027
450	10663	015005254422667632302	506	12153	156143632007424445172
451	10731	152071064215130043430	507	12165	120006765262645336077
452	10737	145776532246263401576	508	12177	170626137746475135240
453	10743	125526535057512033355	509	12211	116143453622254242574
454	10757	017152534020343534565	510	12241	145773200345660016745
455	10761	126164644437752576664	511	12247	057132740717225004242
456	11001	025115524423550355272	512	12255	145656271061050702734
457	11015	151546326717242221123	513	12265	170143114734721007655
458	11023	133515173063547622333	514	12315	105566512574453507512
459	11031	155757463244051676232	515	12323	061377156713125501244
460	11045	114662310022320736464	516	12331	174665637412502407530
461	11067	045427520623675031401	517	12337	07046670745535445774?
462	11073	050046424755735470220	518	12345	156721177756762311677
463	11075	137354606312471037500	519	12417	030630376514475370746
464	11103	167442707464235734547	520	12435	012353335667223160174
465	11105	006330450517051644700	521	12513	071665466013537266517
466	11147	135106460271366340544	522	12515	162713161657546634732
467	11155	147554516674375005670	523	12601	135225274267153535377
468	11163	106473554234511436400	524	12623	044216705777470451249
469	11177	175175541506376621224	525	12645	044121330135334500610
470	11265	113637377757365233761	526	12651	171636156403246972376
471	11271	073554577743606662670	527	12673	002141122414371735527
472	11301	034711722334600342514	528	12705	023021277636114453534
473	11313	163374735641513731110	529	12727	052075203457002110244
474	11411	123310111640625635032	530	12735	005331212025566041644
475	11417	003551101734111060053	531	12753	131732311174376367271
476	11433	162762644763344216114	532	12771	102524511010202116507
477	11435	021601021215567245715	533	13003	172657713715506312674
478	11441	076424642241156530170	534	13011	062231004443335741404
479	11463	170250255473704306234	535	13033	064077177741374503060
480	11471	135766707603016666322	536	13077	110132311371567762513
481	11477	174101615614174260052	537	13101	035565750252232567812
482	11515	072032143723547523315	538	13107	035534024775017127434
483	11545	162776745642562523044	539	13113	061167011076772513413
484	11561	112505316021145265365	540	13125	147357451771730727211
485	11631	051004742576032635500	541	13131	161243364643604558152
486	11637	027523025540560053455	542	13143	020701670146550022133
487	11643	006112422130622340712	543	13145	051153451123214623261
488	11651	153062342261626163144	544	13157	146715422115401063626
489	11657	030541171254260066454	545	13223	030204451410175546034
490	11673	000365136635134676574	546	13245	105405555617332156300
491	11703	005364074776233731733	547	13275	037353022642475257761
492	11721	157453651433302774152	548	13303	150022211371610752320
493	11727	074330670034056057324	549	13311	12115006010675377226
494	11735	034351670366344661676	550	13321	112372533057335510713
495	11763	073614365067265010526	551	13341	134446516272343042553
496	11765	061050507425725524326	552	13347	17452736633267751523
497	12007	163056645410272420654	553	13363	160247352312742652401
498	12023	122375175117135665540	554	13377	161160063414746322650
499	12061	110725322273770720472	555	13413	106402212507567231742
500	12067	175321574411140255026	556	13425	132034037660542327017
501	12111	136423002264610225611	557	13431	022616352120262622447
502	12117	031661323415031704721	558	13475	033311040327034643611
503	12133	044721705521530234301	559	13503	140076644041676017045
504	12135	024772223063775236405	560	13505	132344455446600062676

Mar 21 15:19 1983 10soct.746 Page 6

561	13517	041753342570506140632	617	14717	161202323657312504531
562	13527	176423233362075212357	618	14735	011275712066617065533
563	13535	171435571714516440543	619	14747	121475736577264600129
564	13563	050403714364534541220	620	14755	061617317246414760714
565	13565	031005005426747132214	621	14771	017267167001646641477
566	13571	004402405711342276763	622	15033	132760404054576140374
567	13611	041717141377525702755	623	15035	112777676166412515773
568	13617	126176704251735065506	624	15041	067075500631442414544
569	13627	065374530163634007213	625	15053	103073566633114361420
570	13635	162670304611634535035	626	15063	170466520007434236047
571	13655	056563423762240531515	627	15115	173127757256157616573
572	13663	057744153423077663027	628	15123	132234454572330545103
573	13677	013025057344542646676	629	15125	140125046620202002271
574	13701	145034050212774003217	630	15137	132473354635275703665
575	13737	101121175306340122217	631	15143	035505467224121021772
576	13773	147730561146305253760	632	15151	077566077625004421635
577	14007	047410404160626171641	633	15173	146241307330313571677
578	14015	144752651357036062737	634	15213	105725605534330420527
579	14037	157014416444501317403	635	15261	145761520660574077766
580	14043	037452720214642363734	636	15305	123541740074723521224
581	14061	117275533320043135202	637	15321	165640222716435024672
582	14067	016343051337104256566	638	15341	046613162564235654031
583	14111	076067420703610526771	639	15347	13045633767771647461
584	14127	000461116345237746556	640	15353	127142377672123154041
585	14135	173403731161025603236	641	15365	051746507372075151755
586	14155	067620347351131500221	642	15413	132444345777365334133
587	14171	056704363220710415646	643	15415	142735333132163251357
588	14177	030636363266515562554	644	15423	051642214473537077461
589	14203	046155634602746075225	645	15431	163162503467343433275
590	14221	130027211162406320645	646	15437	040462454107235137302
591	14227	012340464543633141621	647	15457	022732500557401477714
592	14241	157740733046401227634	648	15505	162526437541055615566
593	14271	041013651544444364474	649	15527	147404043175417264352
594	14313	014724435102661160032	650	15611	002311663777330315473
595	14315	103543171765000212712	651	15617	126110362475243000645
596	14357	122303120154777323520	652	15621	127753443114535614443
597	14361	103044303425613410233	653	15647	065560145517357005237
598	14373	127275413151550624273	654	15665	01246465432570421374
599	14405	123305217573465463136	655	15671	056724464442670655206
600	14411	040025313044113147456	656	15677	143316064307220247640
601	14433	135316104575670615530	657	15701	112261433610545040080
602	14455	066542012334204167276	658	15713	015232204317633236053
603	14465	050414032763553303266	659	15723	153360304510113505205
604	14501	101523677306665463672	660	15743	045240323703244651540
605	14513	131116075210003742235	661	15775	057243637360461465335
606	14537	144722401457062171545	662	16003	177257643006344635414
607	14545	142173613324735443141	663	16005	020660417053144245264
608	14557	170545316773102030430	664	16017	020234245732065537423
609	14573	016231013217451011422	665	16021	117405551650070601443
610	14601	035216471227005304610	666	16027	132606166733520030255
611	14613	027223143401477617770	667	16047	030241316656606124076
612	14631	066137432736666700673	668	16115	047645166644420112771
613	14661	141666135035472242121	669	16137	145030147627303213671
614	14667	061524247070041000126	670	16207	052104066151236540672
615	14675	162673066743064731257	671	16237	057127305374535126022
616	14711	037566723704735174641	672	16245	157234675314411563367

673	16267	027400773602556440022	729	17507	127670455056405476153
674	16273	163400426053360205635	730	17513	142033660433670023403
675	16305	157134122607315320212	731	17523	006752003705705157132
676	16311	117006076710423530636	732	17545	134476053402545223173
677	16317	123024160050070212707	733	17561	040002347741356276377
678	16327	010244620733016614356	734	17575	055263543553071260456
679	16353	124654356317707544042	735	17601	101037763160636037521
680	16355	115704354020755530566	736	17615	006045545675124451175
681	16363	135152046173004134321	737	17631	062652674433222165303
682	16401	020574143111302373000	738	17657	067130756353774241514
683	16407	075564507364073477357	739	17673	107436216567774235735
684	16443	113773557132613525207	740	17675	170331533732024567376
685	16457	075510154640221655636	741	17703	001237442013013324352
686	16475	166541163224575342160	742	17705	073461432464751703736
687	16503	150254764737567601245	743	17711	131573516035140170551
688	16521	021326371754477747611	744	17741	037611537054073670734
689	16533	154232306245500071740	745	17755	100325523127146416551
690	16535	102633542046003243514	746	17777	132162346055623713672
691	16547	144550122021607233254			
692	16565	035305602311422107462			
693	16571	157623226667501466147			
694	16603	021122010462572255221			
695	16605	153371135330320365176			
696	16611	121567670604775061321			
697	16621	063650653013662776306			
698	16647	134155112104770052226			
699	16663	175753702742231377223			
700	16701	113047355240251564376			
701	16757	000671057623733220373			
702	17007	107247120031664357043			
703	17025	114474621332712625730			
704	17031	127444557344203270715			
705	17057	042626056201567162326			
706	17073	037332162042555017650			
707	17075	067457763374713704625			
708	17105	106714634050134550002			
709	17121	073442126627251153503			
710	17135	063266555315262725530			
711	17141	053300530142177205704			
712	17147	140342261637230672306			
713	17163	117347737402500321266			
714	17217	007134422776236411320			
715	17227	003640050762233732566			
716	17233	063001676211131015254			
717	17271	053464246727633203451			
718	17277	047475574220321025446			
719	17315	177113474406306146345			
720	17323	003757560436004437330			
721	17343	144234137627325165551			
722	17361	012315063666105716045			
723	17367	031146253500316126415			
724	17403	047746335655622353107			
725	17421	074562234451035065317			
726	17433	101041126513720203570			
727	17447	031312457255102744760			
728	17471	146226640220502344105			

FUJI-HARA, R.  
--Finite field techniques for  
shift registers and encipherment...

LKC  
P91 .C655 F83 1983  
Finite field techniques for  
shift registers and  
encipherment : final report

DATE DUE  
DATE DE RETOUR

APR 10 1985

MAY - 1 1985

JUN 28 1985

01 2018

LOWE-MARTIN No. 1137

INDUSTRY CANADA / INDUSTRIE CANADA



220284

