A GENERAL POLYNOMIAL SIEVE

SHUHONG GAO AND JASON HOWELL

Abstract. An important component of the index calculus methods for finding discrete logarithms is the acquisition of smooth polynomial relations. Gordon and McCurley (1992) developed a sieve to aid in finding smooth Coppersmith polynomials for use in the index calculus method. We discuss their approach and some of the difficulties they found with their sieve. We present a new sieving method that can be applied to any affine subspace of polynomials over a finite field.

1. Introduction

The discrete logarithm problem in finite fields has been extensively studied in the last decade or so, due to its importance in several public-key cryptosystems. The fastest known algorithms for finding discrete logarithms are based on index calculus methods [1, 2, 3, 5, 7, 8, 9, 12, 14]. An important component of an index calculus algorithm for finite fields of small characteristic is the generation of smooth polynomial relations, i.e., equivalences where both polynomials factor completely over a small factor base, usually a set of all irreducibles up to a given degree. Special types of polynomial equivalences have been found by Coppersmith [3], Semaev [13] and Adleman [2]. More precisely, suppose that $m(x) \in \mathbb{F}_q[x]$ and $H(x, y) \in \mathbb{F}_q[x, y]$ are some fixed polynomials. Let

$$A(x) = r(x)m(x) + s(x) \tag{1}$$

and

$$B(x) = H(r(x), s(x)) \tag{2}$$

where $r(x)$ and $s(x)$ range over polynomials up to certain degree in $\mathbb{F}_q[x]$. It is desirable to find many or even all pairs of $r(x)$ and $s(x)$ such that both $A(x)$ and $B(x)$ are smooth. However, it is too time-consuming if one factors each polynomial and checks whether or not it is smooth. Thus we seek a different method to find all smooth polynomials of a given form.

One approach to finding smooth relations is to use a sieve. This is similar to Pomereance's sieve [11] for finding smooth integers. Consider a subspace of polynomials that are of a given form. We wish to determine which of these polynomials factor completely over a small factor base. Once we have an enumeration or array of these polynomials, we can keep track of the sum of degrees of known factors of the polynomials to help us determine which of these are likely to be smooth. To do this, we determine which of them are divisible by each irreducible g in our factor base. Then we move through the array and add the degree of each g to the positions corresponding to these polynomials.

Date: May 13, 1999.
This paper is dedicated to E. F. Assmus, Jr.

In a sieving approach to finding smooth elements, efficiency is based on the fact that only a limited number of simple operations are performed to find initial polynomials that are divisible by an element g in the factor base. Then the structure of the array is used to determine all the polynomials that are divisible by g and their corresponding positions in the array are appropriately marked. This is easily illustrated in the integer case [11], as the integers that are divisible by a given integer lie a fixed distance apart in sequential enumeration. The two main steps in a polynomial sieve are:

1. Finding one polynomial that is divisible by an irreducible g
2. Stepping through other polynomials in the array that are also divisible by g.

The latter step is not as easy as the integer case, there seems no obvious way to enumerate the polynomials in a subspace such that elements of a given residue class lie a fixed distance apart. Gordon and McCurley [6] use a Gray code to step through an array of polynomials. Their method works for a special class of polynomials such as Coppersmith's. Also, as Gordon and McCurley point out, their sieve may move erratically throughout the array, which could affect the overall performance of the approach. In the next section we describe their method. In section 3 we present a sieving method that can be applied to more general forms of polynomials. In addition, we show how to move smoothly throughout the array.

2. The Gordon and McCurley polynomial sieve

Gordon and McCurley's method works for polynomials of the form (1) in $\mathbb{F}_2[x]$ where both $r(x)$ and $m(x)$ are fixed and $s(x)$ varies. The goal is to find all (or many) $s(x) \in \mathbb{F}_2[x]$ up to a certain degree such that $A(x)$ is smooth. In [6], Gordon and McCurley applied this approach to a special case of these polynomials due to Coppersmith [3]. Suppose that $f(x) = x^n + f_1(x) \in \mathbb{F}_2[x]$ is irreducible where $f_1(x)$ has a small degree. Coppersmith uses equivalences in $\mathbb{F}_{2^n} \cong \mathbb{F}_2[x]/(f(x))$ of the form

$$B(x) \equiv A(x)^{2^k} \mod f(x) \tag{3}$$

where

$$\begin{aligned} A(x) &= r(x)x^h + s(x) && (4)\\ B(x) &= r(x)^{2^k} f_1(x) x^{h2^k - n} + s(x)^{2^k}, && (5) \end{aligned}$$

and both $r(x), s(x)$ are arbitrary of degrees up to some bound. Their sieving approach finds, for a fixed $r(x)$, all $s(x)$ in a specified range such that $A(x)$ is divisible by a fixed irreducible g. Once this is done for each irreducible g in the factor base, one can find which $s(x)$ (coupled with the fixed $r(x)$) produce a smooth $A(x)$. Then, with a pair $r(x), s(x)$ that produce a smooth A, the corresponding B polynomial is tested for smoothness. If it is smooth, we factor both polynomials and thus have a smooth polynomial relation. One can execute this sieve for a subrange of the polynomials $r(x)$ until we have found enough smooth relations. In computational experiments that were done in [6], the sieve enabled Gordon and McCurley to complete most of the computations for the index calculus method in fields of order 2^n for various values of n up to around 500.

For a fixed $r(x)$, $A(x)$ is uniquely determined by $s(x)$. Gordon and McCurley consider the mapping $A(x) \mapsto s(2)$ as an enumeration of all $A(x)$ with $s(x)$ of degree less than t. Each polynomial corresponds to a position in the array of 2^t elements.

Let g be an irreducible in a factor base. An initial $s(x)$ such that $A(x)$ is divisible by g is $s(x) = r(x)m(x) \mod g$. Once this $s(x)$ is located in the array, a Gray code is used to step through to the next $s(x)$ such that $A(x) \equiv 0 \mod g$. This is accomplished by setting $s(x) := s(x) + x^{l(i)}g$, where $l(i)$ is the position of the least significant bit in the binary representation of i, which is used as the counter for this loop. For each such $s(x)$, we will add $\deg(g)$ to the position $s(2)$. Once we do this for all g in our factor base, we have an array whose positions correspond to the sum of the degrees of known factors (not counting multiplicities) of $A(x)$ for the fixed $r(x)$ and the range of $s(x)$. If the entry in a position is large, the corresponding $A(x)$ is likely to be smooth.

Gordon and McCurley point out that the memory access patterns for the sieve seem somewhat chaotic, as consecutive $s(x)$ from the Gray code may not lie closely or in an increasing order in the array. It may jump back and forth in a random fashion. This could affect the performance of the sieve on certain types of processors. Also, it is not obvious how this sieve work for polynomials other than $A(x)$ with $r(x)$ and $m(x)$ fixed. Gordon and McCurley remark that a sieve for the Coppersmith polynomials $B(x)$ in (5) would require taking roots of polynomials, which may be too expensive.

3. A general polynomial sieve

The Gordon and McCurley sieve leads our interest into finding a more general polynomial sieve. In particular, we wish to develop a sieving method that will work for any affine subspace of polynomials, not just the Coppersmith polynomials.

Let $\phi_0(x), \phi_1(x), \dots, \phi_b(x)$ denote fixed (not necessarily distinct) polynomials in $\mathbb{F}_q[x]$. Consider the polynomial

$$A(x) = \phi_0(x) + a_1\phi_1(x) + a_2\phi_2(x) + \cdots + a_b\phi_b(x). \tag{6}$$

where $a_i \in \mathbb{F}_q$, $1 \le i \le b$. Note that these polynomials $A(x)$ form an affine linear space of dimension b, and any affine linear space is of this form. When $m(x)$ is fixed, the polynomial $r(x)m(x) + s(x)$ is a special case of (6), as also are the polynomials of Coppersmith [3] and Semaev [13]. In practice, we want to find all smooth $A(x)$. Let g be an element of the factor base with $\deg(g) = t$. Essentially, we wish to determine the conditions on the a_i such that

$$A(x) = \phi_0(x) + a_1\phi_1(x) + a_2\phi_2(x) + \cdots + a_b\phi_b(x) \equiv 0 \mod g(x). \tag{7}$$

Reduce each $\phi_i(x)$ modulo $g(x)$, say

$$\phi_i(x) \equiv \sum_{j=0}^{t-1} g_{ij}x^j \mod g, \tag{8}$$

where $g_{ij} \in \mathbb{F}_q$. Then

$$A(x) \equiv \sum_{i=0}^{b} a_i \sum_{i=0}^{t-1} g_{ij}x^j \equiv \sum_{j=0}^{t-1} \left(\sum_{i=0}^{b} a_i g_{ij} \right) x^j \mod g, \tag{9}$$

where $a_0 = 1$. Now (9) implies

$$\sum_{i=0}^{b} a_i g_{ij} = 0, \quad 0 \le j \le t-1. \tag{10}$$

Since we know each of the g_{ij} (as the ϕ_i are fixed), we can find all $A(x)$ that are divisible by g by solving the linear system (10). Note that this system has t equations and b unknowns. In cases where $t < b$ this system will be underdetermined, so the solutions can be expressed in terms of arbitrary parameters, say $\alpha_1, \dots, \alpha_v$. Thus there are q^v such polynomials $A(x)$ that are divisible by g. Running through all of the choices for $\alpha_1, \dots, \alpha_v$ will give us these polynomials.

To represent all the polynomials $A(x)$ in an array, we assume that each element of $\mathbb{F}_q$ is represented as an integer between 0 and $q-1$. This array has q^b elements. A polynomial $A(x)$ in (6) corresponds to the position j where

$$j = a_1 + a_2 q + \cdots + a_b q^{b-1} = (a_b a_{b-1} \cdots a_2 a_1)_q.$$

So a_b is the most significant digit. As we proceed through the array by assigning the parameters, we can add t to the position in the array corresponding to these polynomials. Table 1 presents a brief outline of this approach when used for all irreducibles in a factor base of degree up to b. Obviously, if b is large then we can

TABLE 1. The general polynomial sieve algorithm

```
Initialize array:   $s[i] := 0$   for $0 \le i \le q^b - 1$.
for each irreducible $g$, say of degree $t$ in the factor base
        compute $\phi_i \equiv \sum_{j=0}^{t-1} g_{ij} x^j \mod g,\ 0 \le i \le b$
        solve the linear system $\sum_{i=0}^{b} a_i g_{ij} = 0,\quad 0 \le j \le t-1$
        for each solution $A = (a_b a_{b-1} \cdots a_1 a_0)_q$  set $s[A] := s[A] + t$
```

set the sieve to go over a certain range of possibilities (e.g., fix some of the a_i). This will allow us to sieve over any subspace of polynomials. So it is easy to use parallel or distributed computer networks to sieve a large space.

Also, most of the steps involved in this sieve are relatively simple. The reductions modulo g are not too costly as the degree of g will be small. The linear system we need to solve will always have at most t rows and should be easy to solve, as t is small.

We need to specify the order in which the sieve proceeds through the array. Recall that the a_i are really digits in the base q representation of a position in the array. By choosing the a_i with the largest indices (most significant bits) to be the arbitrary parameters, as we run through all possibilities for the a_i we will always increase our position (index) in the array. This is best illustrated by an example.

Example 3.1. *Let $q = 2$ and let $\phi_i(x) \in \mathbb{F}_2[x]$ be given by*

$$\begin{array}{rcl}\phi_0(x) & = & 0 \\ \phi_1(x) & = & x^7 + x^5 + 1 \\ \phi_2(x) & = & x^{21} + x^{16} + x^{12} + x^4 + x^3 + x^2 \\ \phi_3(x) & = & x^{33} + x^{29} + x^{28} \\ \phi_4(x) & = & x^5 + x^4 + x^2 + x + 1 \\ \phi_5(x) & = & x^{11} + x^6 + x^5 + x^4 + 1 \\ \phi_6(x) & = & x^{25} + x^{20} + x^{15} \\ \phi_7(x) & = & x^{19} + x^{17} + x^{13} + x^{11} + x^7 + x^5 + x^3 \\ \phi_8(x) & = & x^{42} + 1.\end{array}$$

Denote $A = (a_1, a_2, \ldots, a_8)$. Let $g = x^4 + x^3 + 1$. Reducing each $\phi_i \mod g$ we have the following system of linear equations:

$$\begin{bmatrix}1&1&0&1&1&0&0&0\\0&1&1&0&0&0&1&1\\1&0&0&1&0&0&0&0\\1&1&0&0&0&0&0&0\end{bmatrix} A^T = \mathbf{0}. \tag{11}$$

When performing Gaussian elimination, we always pivot on the leftmost available column at each step. The row-reduced form of the coefficient matrix is

$$\begin{bmatrix}1&0&0&0&1&0&0&0\\0&1&0&0&1&0&0&0\\0&0&1&0&1&0&1&1\\0&0&0&1&1&0&0&0\end{bmatrix} \tag{12}$$

which leads to solutions of the form

$$\begin{array}{rcl}a_1 & = & a_5 \\ a_2 & = & a_5 \\ a_3 & = & a_5 + a_7 + a_8 \\ a_4 & = & a_5\end{array}$$

where a_5, a_6, a_7, a_8 are arbitrary in $\mathbb{F}_2$.

To step through the array, recall that position j in the array corresponds to the binary representation $a_8 a_7 \cdots a_2 a_1$ of j. Table 2 illustrates how we would proceed through the array in regards to the previous example. We simply run through the integers from 0 to $2^4 - 1$ in order to assign values to the four parameters.

In general it is easy to see that choosing the most significant bits to be the arbitrary parameters will ensure that we always step 'to the right' in the array. This avoids the chaotic memory-access patterns that arise in using Gray code as in Gordon and McCurley's sieve. As long as the parameters are chosen in this way, we will have this smooth movement.

When we are done with this particular g, we then move on to the next element of the factor base and proceed the same way. Setting $g = x^5 + x^2 + 1$, we arrive at

TABLE 2. Stepping through the array for $g = x^4 + x^3 + 1$

integer	$a_8a_7a_6a_5$	$a_8a_7a_6a_5a_4a_3a_2a_1$	position
0	0000	00000000	0
1	0001	00011111	31
2	0010	00100000	32
3	0011	00111111	63
4	0100	01000100	68
5	0101	01011011	91
6	0110	01100100	100
7	0111	01111011	123
8	1000	10000100	132
9	1001	10011011	155
10	1010	10100100	164
11	1011	10111011	187
12	1100	11000000	192
13	1101	11011111	223
14	1110	11100000	224
15	1111	11111111	255

new conditions on the a_i, in particular

$$\begin{aligned} a_1 &= a_5 \\ a_2 &= a_6 + a_8 \\ a_3 &= a_5 + a_6 \\ a_4 &= a_5 \\ a_7 &= a_8. \end{aligned}$$

Hence we will let a_5, a_6, a_8 be arbitrary in $\mathbb{F}_2$. Table 3 gives the same sort of procession as Table 2, only with the new conditions. Again it is important to note

TABLE 3. Stepping through the array for $g = x^5 + x^2 + 1$

integer	$a_8a_6a_5$	$a_8a_7a_6a_5a_4a_3a_2a_1$	position
0	000	00000000	0
1	001	00011101	29
2	010	00100110	38
3	011	00111011	59
4	100	11000010	194
5	101	11011111	223
6	110	11100100	228
7	111	11111001	249

the smooth progression through the array.

Another important implementation note in the case where $q = 2$ is that determining the dependent a_i from the arbitrary parameters can be thought of as simply a series of exclusive-or operations on binary vectors. Consider an implementation of the sieve where we arrive at the set of solutions

$$\begin{aligned} a_0 &= a_3 + a_5 + a_6 \\ a_1 &= a_3 + a_4 + a_5 \\ a_2 &= a_4 + a_5 + a_6. \end{aligned}$$

Note that we can write this as the matrix equation

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} \tag{13}$$

or as

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = a_3 \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + a_4 \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + a_5 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + a_6 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}. \tag{14}$$

Thus when we are ready to determine the vector $\mathbf{v} = [a_0, a_1, a_2]^T$, we can do so by adding scalar multiples of the vectors in (14). Since everything is binary, adding vectors is equivalent to the XOR operation (denote this operation by $\oplus$). Thus, when we increment the integer that corresponds to our selection of arbitrary parameters, we need only XOR $\mathbf{v}$ with the vectors whose coefficients have changed from the previous integer, i.e., the vectors whose coefficients are bits that get toggled in the incrementing of the integer. For example, note that if we start at the integer 5, we have

	$a_6a_5a_4a_3$	bits	$a_2a_1a_0$
5	0101	-	100
6	0110	a_3, a_4	$100 \oplus 011 \oplus 110 = 001$
7	0111	a_3	$001 \oplus 011 = 010$

This allows for an even more efficient way of proceeding through the array.

We comment on how to sieve polynomials of the form $H(r(x), s(x))$ where $H(x, y) \in \mathbb{F}_q[x, y]$ is a fixed polynomial. In general, the equation

$$H(r(x), s(x)) \equiv 0 \mod g \tag{15}$$

is not a system of linear equations in the coefficients of $r(x)$ and $s(x)$. One needs special methods to find some initial solutions. Suppose that $r_0(x)$ and $s_0(x)$ are found such that $H(r_0(x), s_0(x)) \equiv 0 \mod g$. since $H(x, y)$ is a polynomial, for any $r(x)$ and $s(x)$ such that

$$r(x) \equiv r_0(x) \mod g \tag{16}$$

$$s(x) \equiv s_0(x) \mod g \tag{17}$$

we have that (15) holds. Note that (16) and (17) are just special cases of (6). So our method can be applied to step through to all $r(x), s(x)$ pairs that satisfy (16) and (17).

Finally, Gordon [4] points out that one important practical improvement they used was Odlyzko's idea of forcing factors into A(x) and B(x). They did that by solving linear equations. This method fits nicely into our method: one just adds more equations to the system for each g that forces the polynomials $A(x)$ and $B(x)$ being divisible by small elements in the factor base. That idea is similar to Pollard's lattice sieve for integers [10].

4. Conclusions

We have developed a polynomial sieve for finding smooth relations. It works for any affine linear space of polynomials over finite fields, including those arising in the algorithms of Adleman, Coppersmith and Semaev. We showed how to step smoothly through the array in the sieving process. The method is suitable for parallel and distributed computer networks. A carefull implementation is needed to determine how well this sieve performs in practice, as well as how it compares to Gordon and McCurley's sieve. It should be mentioned that our sieve will not affect the asymptotic running time of the index calculus methods in [2, 3, 13].

Acknowledgement. We thank Daniel M. Gordon for sharing his ideas with us and thank two anonymous referees for their useful comments.

References

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

[2] ________________, *The function field sieve,* Algorithmic number theory, Lec. Notes in Comp. Sci. **877** 1994, Springer-Verlag, 108-121.

[3] D. Coppersmith, *Fast evaluation of logarithms in fields of characteristic two,* IEEE Trans. Inform. Theory, IT -30, 587-594,1984.

[4] D. M. Gordon, Email communication, 1999.

[5] D. M. Gordon, *Discrete logarithms in $GF(p)$ using the number field sieve*, SIAM J. Disc. Math., **6** (1993), 124-138.

[6] D. M. Gordon and K. S. McCurley, *Massively parallel computation of discrete logarithms,* Advances in Cryptology - Crypto '92, Lec. Notes Comp. Sci. **740** 1993, Springer-Verlag, New York, 312-323.

[7] K. S. McCurley, *The discrete logarithm problem,* Proc. Symposia Applied Mathematics, AMS, 1990.

[8] A. M. Odylzko, *Discrete logarithms in finite fields and their cryptographic significance,* Advances in Cryptology: Proceedings of Eurocrypt '84. Lecture notes in Computer Science **209** pgs. 224-314.

[9] A. M. Odlyzko, *Discrete logarithms and smooth polynomials*, pp. 269-278, in Finite Fields: Theory, Applications, and Algorithms, (Gary L. Mullen and Peter Jau-Shyong Shiue, eds.), Contemporary Mathematics *168*, AMS, 1994.

[10] J. M. Pollard, "The lattice sieve," in *The Development of the Number Field Sieve* (A. K. Lenstra and H.W. Lenstra, Jr., Eds.), Lecture Notes in Mathematics, no. 1554, Springer-Verlag, 1993, 43–49.

[11] C. Pomerance, *The quadratic sieve factoring algorithm,* Advances in Cryptology - Crypto '84, Lec. Notes Comp. Sci. **209** 1985, Springer, New York, 169-182.

[12] O. Schirokauer, D. Weber, and T. Denny, *Discrete logarithms: the effectiveness of the index calculus method,* Algorithmic Number Theory, Lec. Notes Comp. Sci. **1122** (1996), Springer, Berlin, 337-361.

[13] I. A. Semaev, *An algorithm for evaluation of discrete logarithms in some nonprime finite fields,* Preprint, 1994.

[14] A. E. Western and J. C. P. Miller, *Tables of indices and primitive roots,* Royal Society Mathematical Tables, vol. 9, Cambridge Univ. Press, 1968.

Department of Mathematical Sciences, Clemson University, Clemson, SC 29634 USA
E-mail address: SGAO@MATH.CLEMSON.EDU, HOWELL@MATH.CLEMSON.EDU