

**DISCRETE LOGARITHMS
IN FINITE FIELDS**

by

Dean Phillip Reiff

B.A., University of Colorado at Boulder, 1980

B.A., University of Colorado at Denver, 1987

A thesis submitted to the
University of Colorado at Denver
in partial fulfillment
of the requirements for the degree of
Master of Science
Applied Mathematics

1996

This thesis for the Master of Science

degree by

Dean Phillip Reiff

has been approved

by

William Cherowitzo

Sylvia Lu

Stan Payne

Date

Reiff, Dean Phillip (M.S., Applied Mathematics)

Discrete Logarithms in Finite Fields

Thesis directed by Associate Professor William Cherowitzo

ABSTRACT

Given a finite field F_q of order q , and g a primitive element of F_q , the discrete logarithm base g of an arbitrary, non-zero $y \in F_q$ is that integer x , $0 \leq x \leq q-2$, such that $g^x = y$ in F_q . The security of many real-world cryptographic schemes depends on the difficulty of computing discrete logarithms in large finite fields. This thesis is a survey of the discrete logarithm problem in finite fields, including: some cryptographic applications (password authentication, the Diffie-Hellman key exchange, and the ElGamal public-key cryptosystem and digital signature scheme); Niederreiter's proof of explicit formulas for the discrete logarithm; and algorithms for computing discrete logarithms (especially Shank's algorithm, Pollard's ρ -method, the Pohlig-Hellman algorithm, Coppersmith's algorithm in fields of order 2^n , and the Gaussian integers method for fields of prime order).

This abstract accurately represents the content of the candidate's thesis. I recommend its publication.

Signed _____
William Cherowitzo

ACKNOWLEDGEMENT

I would like to thank my adviser, Bill Cherowitzo, for all the help and guidance that he has graciously given me. I would also like to thank Sylvia Lu and Stan Payne for their service on my thesis committee.

Contents

1	INTRODUCTION	1
2	CRYPTOGRAPHIC APPLICATIONS	3
3	EXPLICIT FORMULAS	6
4	SQUARE ROOT ALGORITHMS	9
4.1	BABY-STEP GIANT-STEP METHOD	9
4.2	POLLARD'S ρ -METHOD	10
4.3	POHLIG-HELLMAN ALGORITHM	12
5	INDEX CALCULUS ALGORITHMS	14
5.1	COPPERSMITH'S ALGORITHM	17
5.2	GAUSSIAN INTEGERS METHOD	20
5.3	OTHER DEVELOPMENTS	23
6	CONCLUSION	25
	REFERENCES	26

1 INTRODUCTION

Let F_q be a finite field of order q , where $q = p^n$ (p prime), and let $F_q^* = F_q - \{0\}$. Given g , a primitive element of F_q , and an arbitrary $y \in F_q^*$, the *discrete logarithm* of y base g is defined as

$$\log_g y = x \iff g^x = y \text{ in } F_q \text{ and } 0 \leq x \leq q - 2.$$

(Notice that g primitive \Rightarrow existence of x , and $0 \leq x \leq q - 2 \Rightarrow$ uniqueness of x .) A fundamental difference between discrete logarithms and real logarithms is that the magnitude of y gives us no information about the magnitude of x .

Gauss [15] referred to the discrete logarithm as the *index* of a number (a nomenclature that is still sometimes used), and he noted (for the case q a prime) some of its basic properties:

- Calculations with discrete logarithms are performed modulo $q - 1$.
- $\log_g (y_1 y_2) \equiv \log_g y_1 + \log_g y_2 \pmod{q - 1}$
- $\log_g \left(\frac{y_1}{y_2}\right) \equiv \log_g y_1 - \log_g y_2 \pmod{q - 1}$
- Change of base: suppose Γ is another primitive element of F_q and we know $\log_g \Gamma = \gamma$. Γ and g both primitive $\Rightarrow (\gamma, q - 1) = 1$
 $\Rightarrow \exists \bar{\gamma}$ such that $\gamma \bar{\gamma} \equiv 1 \pmod{q - 1} \Rightarrow g = \Gamma^{\bar{\gamma}}$ in F_q .
Therefore $\log_g y = x \Leftrightarrow y = g^x = \Gamma^{\bar{\gamma}x}$ in $F_q \Leftrightarrow \log_\Gamma y \equiv \bar{\gamma}x \pmod{q - 1}$.
Multiplying the last congruence by γ gives $\log_g y \equiv \log_g \Gamma \cdot \log_\Gamma y \pmod{q - 1}$.

The most obvious method of finding the discrete logarithm of y is to simply keep raising g to different powers until we find the specific exponent x such that $g^x = y$ in F_q . However, if q is very large, this method is computationally infeasible, and while faster algorithms have been developed, the discrete logarithm problem remains intractable (for almost all $y \in F_q^*$) in very large fields F_q . This intractability makes the discrete logarithm problem useful in cryptographic applications, and many such applications have been developed, and implemented, in the last two decades. These real-world cryptographic implementations have raised the stakes on the discrete log problem, making the research of faster algorithms to solve the problem a matter of financial, and even national, security – it's now imperative to know in how large a field the problem is solvable.

There is also interest in the discrete logarithm problem in other environments – e.g. in the group of points on an elliptic curve over a finite field, or in the multiplicative (non-cyclic) group of units in Z_n , where n is a composite integer. Even in F_q^* , some cryptographic applications allow g to be non-primitive, so that the order of the group $\langle g \rangle$, and the mere existence of $\log_g y$ for arbitrary $y \in F_q^*$, are unknown. However, the case where g is primitive in a finite field environment has received the most attention, and I have limited the scope of this thesis to that case. I provide a survey of the discrete logarithm problem, including algorithms for computing discrete logarithms, explicit formulas for the discrete logarithm, and some of the most important cryptographic schemes whose security depends on the intractability of the discrete logarithm problem.

Throughout this thesis, \log_2 and \ln will denote real logarithms to the bases 2 and e , respectively. Additionally, q will continue to denote the order of a finite field, and g will always be assumed to be a primitive element. I refer to fields of prime order as ‘prime fields’, and to all other finite fields as ‘fields of prime power order’.

2 CRYPTOGRAPHIC APPLICATIONS

Discrete logarithms in large finite fields F_q come into play in cryptography because they appear to have the attributes of a *one-way function*. That is, while the discrete logarithm problem is intractable, its inverse function (viz. discrete exponentiation) is relatively easy to compute. Using the binary representation of an exponent x , $0 \leq x \leq q - 2$, the ‘square-and-multiply’ method can compute $g^x \in F_q$ with at most $2 \log_2 q$ multiplications and modular reductions. For example,

$$g^{19} = g^{10011_2} = g^{10000_2 + 10_2 + 1_2} = (((g^2)^2)^2 \cdot g)^2 \cdot g,$$

and computational blowup can be avoided by taking a modular reduction after each multiplication.

Below, I present just a few of the most widely used cryptographic schemes whose security depends on the difficulty of computing discrete logarithms. A safe field size in which to implement these schemes is $q \approx 10^{150} \sim 10^{300}$, depending on the importance of the secret, how long it needs to remain secret (e.g. an embarrassing diplomatic correspondence may need to remain secret for decades), and the computing resources available to those who might want to compromise the secret. (For comparison, there are approximately 10^{51} atoms in our planet, and the age of the universe is approximately 10^{17} seconds.) As will be seen in later sections, there are other aspects of the field F_q that can effect the ease with which discrete logarithms can be found in it. Most importantly, it is necessary that $q - 1$ have at least one large prime factor.

For those who haven’t seen much cryptography before, it may not be obvious how we can talk about messages (i.e. strings of characters of a language) as being integers, or even more generally, as elements of a finite field. An easy way to make the conversion to integers is simply to instruct our computer to take the sequence of 0-1 bits it had been interpreting as a character string, and to instead interpret that sequence of bits to be a large integer (stored in binary). In order to fix a maximum for the size of integers we want to deal with, it may be necessary to break the message into a number of sub-messages of a fixed size. Thus, for example, if we break a message into blocks of 64 8-bit characters, we can limit the size of our integers to be less than $2^{512} \approx 10^{154}$. Similarly, the bits of the message can be interpreted to be the coefficients of an element in a field F_{2^n} , while for fields F_{p^n} , p odd, we can convert a binary message to p -ary coefficients – e.g. convert the message 0111 to $2x + 1 \in F_{3^n}$.

The simplest cryptographic application that relies on the difficulty of computing discrete logarithms concerns user authentication – for example, verifying passwords on a multi-user computer. It would be very risky to have a file stored in the computer that contained every user’s password, yet the computer needs some way to verify the legitimacy of a password. Instead of storing the password x_i for each user i , we can choose a finite field F_q and a primitive element g , and store the values $g^{x_i} = y_i \in F_q$ for each user i . (g , and the modulus for F_q also need to be stored, but they can be the same for all users.) To authenticate a user’s password, the computer first calculates g^{x_i} , then compares the result for a match on the stored file. However, even if someone gains access to the stored file, in order to impersonate user i , they would first have to calculate the discrete logarithm of y_i in F_q .

The problem with classical, private-key cryptography has always been that, if two users wanted to communicate privately over a public, insecure channel, they first needed, in some private and secure way, to agree on a shared secret key, which they would both use to encipher and decipher their messages to each other. Furthermore, this problem is compounded for a large system of users, since a different secret key is needed for each pair of users in the system. This problem was eliminated in 1976, when Diffie and Hellman [11] invented the concept of a public-key cryptosystem. In their scheme, a finite field F_q and a primitive element g are publicly agreed upon. Each user i chooses an integer private key x_i , $2 \leq x_i \leq q - 2$, which they keep secret, but makes publicly known their public key $y_i = g^{x_i} \in F_q$. To encipher and decipher messages to each other, two users A and B use the key $g^{x_A x_B}$, which they can each calculate as $(y_B)^{x_A}$ or $(y_A)^{x_B}$, respectively. An ability to find discrete logarithms in F_q would clearly break the system, and for some special cases of q , it has been shown [6, 24] that any method of breaking the system is computationally equivalent to the discrete logarithm problem.

In 1985, ElGamal [12] proposed a public-key cryptosystem and digital signature scheme in which the public and private keys are the same as in the Diffie-Hellman system. User B can send a message $m \in F_q$ to user A by choosing an integer k , $2 \leq k \leq q - 2$ (and it’s important to choose a different k for each message), then sending the pair $(g^k, m y_A^k)$ to user A . User A knows $-x_A \pmod{q - 1}$, and can calculate y_A^{-k} as $(g^k)^{-x_A}$ to recover the message $m = (m y_A^k) y_A^{-k}$.

To implement the ElGamal digital signature scheme, we must restrict ourselves to a field of prime order p (actually this can be extended to fields of prime power order if we use an integer representation of field elements where necessary – see e.g. the description in [32]). To sign a message m , $1 \leq m \leq p - 1$,

a user A will provide a pair of integers (r, s) , $1 \leq r, s \leq p - 1$, that satisfy the following properties: a knowledge of x_A is necessary to produce the signature; anyone who knows m (including a judge in a court of law) can use y_A to verify the signature; and any alteration of the message m after the signature was produced will nullify the signature. To calculate r and s , user A chooses an integer k , $1 \leq k \leq p - 2$, such that $(k, p - 1) = 1$ (again, we must use a different k for each message), and calculates

$$\begin{aligned} r &\equiv g^k \pmod{p}, \text{ and} \\ s &\equiv k^{-1}(m - x_A r) \pmod{p - 1} \quad ((k, p - 1) = 1 \Rightarrow \exists k^{-1} \pmod{p - 1}). \end{aligned}$$

Thus, these r, s satisfy

$$g^m \equiv g^{x_A r + k s} \equiv (g^{x_A})^r (g^k)^s \equiv (y_A)^r r^s \pmod{p},$$

so to verify the signature, anyone can calculate g^m and $(y_A)^r r^s \pmod{p}$, and check that they are equal. In 1994, the National Institute of Standards and Technology [28] adopted a slightly modified version of this scheme as the Digital Signature Standard to be used by U.S. government agencies.

3 EXPLICIT FORMULAS

While worthless for actually calculating discrete logarithms (it would be faster to generate the whole field), it is fascinating that there exists explicit formulas for discrete logarithms in finite fields. Wells [42] gave an explicit polynomial form for discrete logarithms in prime fields, and Mullen and White [26] found an explicit form for discrete logarithms in fields of prime power order. Niederreiter [29] has provided much simpler proofs of these formulas, which I present below, that also generalizes the formula of Wells to fields of prime power order.

Let $q = p^n, n \geq 1$, be the order of the field. Some of the simplicity of Niederreiter's proof is achieved by noting that it suffices to determine $\log_g y$ modulo the characteristic p . To see this, notice that we can represent $\log_g y = \sum_{i=0}^{n-1} x_i p^i$, where $x_i \in \{0, 1, \dots, p-1\} \forall i$. Assume we can find $x_0 \equiv \log_g y \pmod{p}$. Then $\log_g y = x_0 + c_1 p$, for some integer $c_1 \geq 0$.

Let $y_1 = (y g^{-x_0})^{\frac{q}{p}}$. Then $y_1 = (g^{x_0 + c_1 p - x_0})^{\frac{q}{p}} = g^{c_1 q} = g^{c_1}$. So $c_1 = \log_g y_1$.

Continuing this process, we can, by assumption, find $x_1 \equiv \log_g y_1 \pmod{p}$,

Then $\log_g y = x_0 + c_1 p = x_0 + (x_1 + c_2 p)p = x_0 + x_1 p^1 + c_2 p^2$.

Let $y_2 = (y_1 g^{-x_1})^{\frac{q}{p}} = (y g^{-x_0 - x_1 p})^{\frac{q}{p^2}} = (g^{c_2 p^2})^{\frac{q}{p^2}} = g^{c_2}$. So $c_2 = \log_g y_2$.

$x_2 \equiv \log_g y_2 \pmod{p}$

\vdots

$x_{n-1} \equiv \log_g y_{n-1} \pmod{p}$.

Thus, we can obtain $\log_g y = x_0 + x_1 p + x_2 p^2 + \dots + (x_{n-1} + c_n p)p^{n-1}$,

where $0 \leq \log_g y \leq q-2 \Rightarrow c_n = 0$.

Niederreiter's proof uses the convention that $0^0 = 1$.

Lemma 3.1 *For integers $j \geq 0$ we have*

$$\sum_{c \in F_q} c^j = \begin{cases} 0, & \text{if } j = 0, \text{ or } j \not\equiv 0 \pmod{q-1} \\ -1, & \text{otherwise.} \end{cases}$$

Proof

If $j = 0$, we count to q modulo p (since $0^0 = 1$).

If $j \equiv 0 \pmod{q-1}$ and $j \neq 0$, we count to $q-1$ modulo p (since $0^j = 0$).

If $j \not\equiv 0 \pmod{q-1}$,

$$\sum_{c \in F_q} c^j = \sum_{c \in F_q^*} c^j = \sum_{i=0}^{q-2} (g^i)^j = \sum_{i=0}^{q-2} (g^j)^i = \frac{g^{j(q-1)} - 1}{g^j - 1} = \frac{1 - 1}{g^j - 1} = 0. \quad \square$$

Lemma 3.2 *If $q \geq 3$ and k is any integer with $0 \leq k \leq q - 1$, then*

$$\sum_{\substack{c \in F_q \\ c \neq 1}} \frac{c^k}{1 - c} \in F_p, \text{ and the sum is congruent to } k \pmod{p}$$

Proof

Let S_k be the sum on the left. If $k = 0$, then

$$\begin{aligned} S_0 &= \sum_{\substack{c \in F_q \\ c \neq 1}} \frac{c^0}{1 - c} = \sum_{\substack{c \in F_q \\ c \neq 1}} \frac{1}{1 - c} \\ &= \sum_{c \in F_q^*} \frac{1}{c} = \sum_{d \in F_q^*} d \text{ (just permuting the terms in the sum)} \\ &= \sum_{d \in F_q} d = 0 \quad (\text{by lemma 1, with } j = 1). \end{aligned}$$

If $1 \leq k \leq q - 1$, then

$$\begin{aligned} S_k &= S_k - 0 = S_k - S_0 \\ &= \sum_{j=1}^k (S_j - S_{j-1}) = \sum_{j=1}^k \sum_{\substack{c \in F_q \\ c \neq 1}} \frac{c^{j-1}(c-1)}{1-c} = - \sum_{j=1}^k \sum_{\substack{c \in F_q \\ c \neq 1}} c^{j-1} \\ &= - \sum_{j=1}^k (\sum_{\substack{c \in F_q \\ c \neq 1}} c^{j-1} + 1 - 1) = - \sum_{j=1}^k (\sum_{c \in F_q} c^{j-1} - 1) \\ &= - \sum_{j=1}^k (0 - 1) \quad (\text{by lemma 1, since } j - 1 \leq k - 1 < q - 1) \\ &= k, \text{ and } k \text{ an integer} \Rightarrow k \in F_p. \quad \square \end{aligned}$$

From lemma 3.2, the following definition is immediate:

$$T_k := \sum_{\substack{c \in F_q \\ c \neq 0,1}} \frac{c^k}{1 - c} \equiv k \pmod{p} \quad \text{for } q \geq 3 \text{ and } 1 \leq k \leq q - 1. \quad (3.0.1)$$

(i.e. $T_k = S_k - \frac{0^k}{1-0} = S_k$, since $k > 0$.) As will be seen, equation (3.0.1) is the beautifully simple key to Niederreiter's proofs. Theorem 3.1 proves the result of Mullen and White [26].

Theorem 3.1 For any $y \in F_q^*$, $q \geq 3$, we have

$$\log_g y \equiv -1 + \sum_{j=1}^{q-2} \frac{y^j}{g^{-j} - 1} \pmod{p}$$

where on the right-hand side of the congruence there is an element of F_p .

Proof

Put $k = \log_g y + 1$ in equation (3.0.1), and note that

$$c^{\log_g y} = (g^{\log_g c})^{\log_g y} = (g^{\log_g y})^{\log_g c} = y^{\log_g c} \quad \forall c \in F_q^*.$$

Then we get

$$\begin{aligned} \log_g y &\equiv -1 + \sum_{\substack{c \in F_q \\ c \neq 0,1}} \frac{c^{\log_g y+1}}{1-c} \equiv -1 + \sum_{\substack{c \in F_q \\ c \neq 0,1}} \frac{c^{\log_g y}}{c^{-1}-1} \pmod{p} \\ &\equiv -1 + \sum_{\substack{c \in F_q \\ c \neq 0,1}} \frac{y^{\log_g c}}{c^{-1}-1} \equiv -1 + \sum_{\substack{c \in F_q^* \\ c \neq 1}} \frac{y^{\log_g c}}{g^{-\log_g c}-1} \pmod{p} \\ &\equiv -1 + \sum_{j=1}^{q-2} \frac{y^j}{g^{-j}-1} \pmod{p} \quad (\text{where } c \neq 1 \Leftrightarrow j \neq 0). \quad \square \end{aligned}$$

If we define the discrete logarithm slightly differently, we get another form, which is a generalization of Wells [42] result to fields of prime power order. Let $\text{Log}_g y$ be the unique integer x such that $y = g^x$ and $1 \leq x \leq q-1$. (Note that $\text{Log}_g y = \log_g y$, except that $\text{Log}_g 1 = q-1$, while $\log_g 1 = 0$.)

Theorem 3.2 For any $y \in F_q^*$, $q \geq 3$, we have

$$\text{Log}_g y \equiv \sum_{j=1}^{q-2} \frac{y^j}{1-g^j} \pmod{p},$$

where on the right-hand side of the congruence there is an element of F_p .

Proof

Put $k = \text{Log}_g y$ in equation (3.0.1), and note that $c^{\text{Log}_g y} = y^{\text{Log}_g c} \quad \forall c \in F_q^*$.

Then we get

$$\begin{aligned} \text{Log}_g y &\equiv \sum_{\substack{c \in F_q \\ c \neq 0,1}} \frac{c^{\text{Log}_g y}}{1-c} \equiv \sum_{\substack{c \in F_q \\ c \neq 0,1}} \frac{y^{\text{Log}_g c}}{1-g^{\text{Log}_g c}} \pmod{p} \\ &\equiv \sum_{j=1}^{q-2} \frac{y^j}{1-g^j} \pmod{p} \quad (\text{where } c \neq 1 \Leftrightarrow j \neq q-1). \quad \square \end{aligned}$$

4 SQUARE ROOT ALGORITHMS

In this section I present three algorithms for computing discrete logarithms, each of which can achieve a running time of order roughly $\bar{p}^{\frac{1}{2}}$, where \bar{p} is the largest prime factor of $q - 1$. That's not fast enough to be useful in large, arbitrary finite fields, but if $q - 1$ has no large prime factors, these algorithms can be very practical.

Several examples in this section use the field F_{37} generated by $g = 5$, so I list F_{37}^* here:

x	5^x	x	5^x	x	5^x
0	1	12	10	24	26
1	5	13	13	25	19
2	25	14	28	26	21
3	14	15	29	27	31
4	33	16	34	28	7
5	17	17	22	29	35
6	11	18	36	30	27
7	18	19	32	31	24
8	16	20	12	32	9
9	6	21	23	33	8
10	30	22	4	34	3
11	2	23	20	35	15

4.1 BABY-STEP GIANT-STEP METHOD

Let $m = \lceil (q - 1)^{\frac{1}{2}} \rceil$. This algorithm, which is attributed to Shanks [38], makes use of the fact that we can express $\log_g y$ as $x = i + jm$, with $0 \leq i, j < m$. To find i and j , we first pre-compute a list of pairs (i, g^i) for $0 \leq i < m$, and sort the list according to the second component. Then for $0 \leq j < m$, we calculate yg^{-jm} until we find a j such that $yg^{-jm} = g^i$ for some i in the list. Thus, for this i and j , $y = g^{i+jm}$.

For example, if $q = 37$ ($m = 6$), and $g = 5$, we pre-compute and sort the list: $(i, g^i) \rightarrow (0,1)(1,5)(3,14)(5,17)(2,25)(4,33)$.

Now, to find the \log_5 of $y = 2$, we

let $j = 0$ and compute $2 \cdot 5^{-0 \cdot 6} = 2$, and

let $j = 1$ and compute $2 \cdot 5^{-1 \cdot 6} = 2 \cdot 5^{30} = 17$.

Since 17 appears as a second component in the list, corresponding to $i = 5$, we have $\log_5 2 = 5 + 1 \cdot 6 = 11$. \square

To achieve a running time of order roughly $\overline{p}^{\frac{1}{2}}$, we can use the algorithm in several stages, where in each stage we compute a logarithm in a subgroup of F_q^* of order not greater than the approximate size of \overline{p} . I believe Pollard [35] was the first to propose this multi-stage idea, and (while it's obviously not worth the trouble for a field this small) to show how this can be done, I outline the procedure for finding $\log_5 2$ in F_{37} in 3 stages. Note that $37 - 1 = 3^2 \cdot 2^2$.

STAGE 1 in a subgroup of order 3: We use the algorithm to find that the logarithm of y^{12} to the base g^{12} is 2, so $y^{12} = (g^{12})^2$. Taking 12^{th} roots, we have $y = g^2(g^3)^{k_1}$, $0 \leq k_1 \leq 11$ (since g^3 is a 12^{th} root of unity), so $x = 2 + 3k_1$.

STAGE 2 in a subgroup of order 3: To find k_1 , let $y_2 = yg^{-2} = (g^3)^{k_1}$, and use the algorithm to find the logarithm of y_2^4 to the base $(g^3)^4$. Since the logarithm is 0, we have $(g^3)^{4 \cdot 0} = y_2^4$, so taking 4^{th} roots gives us $y_2 = (g^3)^0(g^9)^{k_2} = (g^3)^0(g^3)^{3k_2}$, $0 \leq k_2 \leq 3$, so $k_1 = 0 + 3k_2$.

STAGE 3 in a subgroup of order 4: Use the algorithm to find that the logarithm of y_2 ($= y_2 \cdot (g^3)^{-0}$) to the base g^9 is 1. Thus, $k_2 = 1 \Rightarrow k_1 = 3 \Rightarrow x = 11$. \square

4.2 POLLARD'S ρ -METHOD

One drawback of Shanks algorithm is the need to sort and store a list of size $\overline{p}^{\frac{1}{2}}$. Pollard [35] introduced a probabilistic algorithm which eliminates the need for such storage. To find $\log_g y$, we first divide F_q^* into three sets S_1 , S_2 , and S_3 , that are approximately the same size. Define a sequence r_i from F_q^* by $r_0 = 1$, and for $i \geq 0$,

$$r_{i+1} = \begin{cases} yr_i & \text{if } r_i \in S_1 \\ r_i^2 & \text{if } r_i \in S_2 \\ gr_i & \text{if } r_i \in S_3. \end{cases}$$

Thus, every element in the sequence has the form $r_i = y^{a_i}g^{b_i}$, where $a_0 = b_0 = 0$, and for $i \geq 0$,

$$a_{i+1} = \begin{cases} a_i + 1 \pmod{q-1} & \text{if } r_i \in S_1 \\ 2a_i \pmod{q-1} & \text{if } r_i \in S_2 \\ a_i & \text{if } r_i \in S_3, \text{ and} \end{cases}$$

$$b_{i+1} = \begin{cases} b_i & \text{if } r_i \in S_1 \\ 2b_i \pmod{q-1} & \text{if } r_i \in S_2 \\ b_i + 1 \pmod{q-1} & \text{if } r_i \in S_3. \end{cases}$$

The sequence r_i behaves like a random walk in F_q^* , so we expect to find an i of size approximately $q^{\frac{1}{2}}$, such that $r_i = r_{2i}$ (see [35] for a more detailed discussion on the expected value of this i).

To find $\log_g y$, we run through a sequence of 6-tuples $(r_i, a_i, b_i, r_{2i}, a_{2i}, b_{2i})$, calculating each one from the previous one (which is quite easy, since r_i , r_{2i-1} , and r_{2i} are calculated by simple field multiplications), until we reach a case such that $r_i = r_{2i}$ (i.e. $y^{a_i} g^{b_i} = y^{a_{2i}} g^{b_{2i}}$). Let $m \equiv a_i - a_{2i} \pmod{q-1}$ and $n \equiv b_{2i} - b_i \pmod{q-1}$, then

$$y^m = g^n \text{ in } F_q. \quad (4.2.1)$$

Now, we use the Extended Euclidean Algorithm to find a λ and μ such that $d = \gcd(m, q-1) = \lambda m + \mu(q-1)$. Thus $\lambda m \equiv d \pmod{q-1}$, so raising equation (4.2.1) to the λ power gives us $y^d = g^{\lambda n}$, and λn must be of the form dk (since the left hand side is a d^{th} power). Taking d^{th} roots gives us

$$y = g^k (g^{\frac{q-1}{d}})^j, \quad 0 \leq j \leq d-1,$$

and we can simply try each possible value of j until we find an equality (for m random, we expect large values of d to be rare).

As an example, we find $\log_5 17$ in F_{37} .

Let $S_1 = \{1, 2, \dots, 12\}$, $S_2 = \{13, 14, \dots, 24\}$, and $S_3 = \{25, 26, \dots, 36\}$.

We calculate (but don't store):

$$\begin{aligned} (r_1 = 17, & \quad a_1 = 1, \quad b_1 = 0, \quad r_2 = 30, \quad a_2 = 2, \quad b_2 = 0) \\ (r_2 = 30, & \quad a_2 = 2, \quad b_2 = 0, \quad r_4 = 34, \quad a_4 = 3, \quad b_4 = 1) \\ (r_3 = 2, & \quad a_3 = 2, \quad b_3 = 1, \quad r_6 = 3, \quad a_6 = 6, \quad b_6 = 4) \\ (r_4 = 34, & \quad a_4 = 3, \quad b_4 = 1, \quad r_8 = 11, \quad a_8 = 14, \quad b_8 = 8) \\ (r_5 = 22, & \quad a_5 = 3, \quad b_5 = 2, \quad r_{10} = 34, \quad a_{10} = 16, \quad b_{10} = 8) \\ (r_6 = 3, & \quad a_6 = 6, \quad b_6 = 4, \quad r_{12} = 3, \quad a_{12} = 32, \quad b_{12} = 18) \end{aligned}$$

Then we have $3 = 5^4 17^6 = 5^{18} 17^{32}$, so $17^{26} = 5^{-14} = 5^{22}$.

$d = \gcd(26, 36) = 2 = 7 \cdot 26 - 5 \cdot 36$, so raising to the 7^{th} power gives us $17^2 = 5^{22 \cdot 7} = 5^{154}$. Taking square roots gives $17 = 5^5 (5^{18})^j$, $j = 0$ or 1 . Trying $j = 0$ gives us the equality $17 = 17$, so $\log_5 17 = 5$. \square

By a procedure similar to that outlined in the previous subsection, Pollard's ρ -method can be used in several stages, in subgroups of F_q^* , to yield a running time of order $\overline{p}^{\frac{1}{2}}$. It's possible, especially in a subgroup of very small size, that the algorithm will give us a useless equation (e.g. $y^a g^b = y^{a+q-1} g^b$). If this happens, we can simply change the initial conditions a_0 and b_0 , to try the algorithm again with a different r_0 .

Pollard [35] also proposed a λ -method, which can find a discrete logarithm that is known to lie in an interval $x_1 \leq x \leq x_2$ in time $O((x_2 - x_1)^{\frac{1}{2}})$. However, to my knowledge, no one has figured out a way to pre-determine such an interval.

4.3 POHLIG-HELLMAN ALGORITHM

Although this method is known as the Pohlig-Hellman [34] algorithm (who first published it), they credit Roland Silver as having independently discovered it earlier. We begin by finding the prime factorization

$$q - 1 = \prod_{i=1}^k p_i^{n_i}.$$

We will reduce the problem of finding $x = \log_g y$ into k subproblems, where each subproblem will be solved by finding n_i discrete logarithms in a subgroup of size p_i .

For each i , $1 \leq i \leq k$, we will find $x_i \equiv x \pmod{p_i^{n_i}}$, then use the Chinese Remainder Theorem to find x . For each i , let

$$x_i = \sum_{j=0}^{n_i-1} x_{i,j} p_i^j, \quad 0 \leq x_{i,j} \leq p_i - 1,$$

and note that $\forall i$, $x = x_i + c_i p_i^{n_i}$, for some $c_i \in \mathbb{Z}$. Then, since $g^{q-1} = 1$, we have the following equalities in F_q^* :

$$(y)^{\frac{q-1}{p_i}} = (g^x)^{\frac{q-1}{p_i}} = g^{(\sum_{j=0}^{n_i-1} x_{i,j} p_i^j + c_i p_i^{n_i}) \frac{q-1}{p_i}} = (g^{\frac{q-1}{p_i}})^{x_{i,0}}, \quad 0 \leq x_{i,0} \leq p_i - 1 \quad (4.3.1)$$

$$(y \cdot g^{-x_{i,0}})^{\frac{q-1}{p_i^2}} = g^{(\sum_{j=1}^{n_i-1} x_{i,j} p_i^j + c_i p_i^{n_i}) \frac{q-1}{p_i^2}} = (g^{\frac{q-1}{p_i^2}})^{x_{i,1}}, \quad 0 \leq x_{i,1} \leq p_i - 1 \quad (4.3.2)$$

$$(y \cdot g^{-x_{i,0} - x_{i,1} p_i})^{\frac{q-1}{p_i^3}} = g^{(\sum_{j=2}^{n_i-1} x_{i,j} p_i^j + c_i p_i^{n_i}) \frac{q-1}{p_i^3}} = (g^{\frac{q-1}{p_i^3}})^{x_{i,2}}, \quad 0 \leq x_{i,2} \leq p_i - 1 \quad (4.3.3)$$

⋮

$$(y \cdot g^{-(\sum_{j=0}^{n_i-2} x_{i,j} p_i^j)})^{\frac{q-1}{p_i^{n_i}}} = g^{(x_{i,n_i-1} p_i^{n_i-1} + c_i p_i^{n_i}) \frac{q-1}{p_i^{n_i}}} = (g^{\frac{q-1}{p_i^{n_i}}})^{x_{i,n_i-1}}, \quad 0 \leq x_{i,n_i-1} \leq p_i - 1 \quad (4.3.4)$$

These equalities basically describe the algorithm. That is, to find x_i , we run through equations (4.3.1) through (4.3.4), first calculating the left hand side of an equation (using previously found results), then finding $x_{i,j}$ as its discrete logarithm to the base $g^{\frac{q-1}{p_i}}$. For small p_i , we can find $x_{i,j}$ by trial and error, while for large p_i , we can use one of the previously mentioned algorithms of this section to find these logarithms.

(Notice that if $q - 1 = 2^s t$, t odd, then equations (4.3.1) through (4.3.4), with $p_i = 2$, give us a very easy way to extract the s least significant bits in the binary representation of x . For more on the security of individual bits in discrete logarithms, see [5, 33, 22].)

As an example, we find $\log_5 17$ in F_{37} : We have $q - 1 = 2^2 \cdot 3^2$.

For $p_1 = 2$:

$$\begin{aligned} -1 &= 17^{\frac{36}{2}} = (5^{\frac{36}{2}})^{x_{1,0}} = -1^{x_{1,0}} \Rightarrow x_{1,0} = 1 \\ 1 &= (17 \cdot 5^{-1})^{\frac{36}{4}} = (5^{\frac{36}{2}})^{x_{1,1}} = -1^{x_{1,1}} \Rightarrow x_{1,1} = 0 \end{aligned}$$

So $x_1 = x_{1,0} + x_{1,1} \cdot 2 = 1 + 0 \cdot 2 = 1 \Leftrightarrow x \equiv 1 \pmod{4}$.

For $p_2 = 3$:

$$\begin{aligned} 26 &= 17^{\frac{36}{3}} = (5^{\frac{36}{3}})^{x_{2,0}} = 10^{x_{2,0}} \Rightarrow x_{2,0} = 2 \\ 10 &= (17 \cdot 5^{-2})^{\frac{36}{9}} = (5^{\frac{36}{3}})^{x_{2,1}} = 10^{x_{2,1}} \Rightarrow x_{2,1} = 1 \end{aligned}$$

So $x_2 = x_{2,0} + x_{2,1} \cdot 3 = 2 + 1 \cdot 3 = 5 \Leftrightarrow x \equiv 5 \pmod{9}$.

Finally, we use the Chinese Remainder Theorem to find that $x = 5$. □

5 INDEX CALCULUS ALGORITHMS

The fastest method for computing discrete logarithms is known as the index calculus method. The basic ideas in the algorithm appear as early as 1922 in the work of Kraitchik [19, pp.119–123]. Adleman [1] analyzed the running time of the algorithm for the case q a prime, and Hellman and Reyneri [18] extended the algorithm to the case $q = p^n$, for fixed p and $n \rightarrow \infty$. The running time of the algorithm has the form

$$L[q, \alpha, c] = \exp((c + o(1))(\ln q)^\alpha (\ln \ln q)^{1-\alpha}), \quad 0 < \alpha < 1,$$

and c a constant ($o(1) \rightarrow 0$, as $q \rightarrow \infty$), which is known as *subexponential* (if α were 0, the time would be polynomial in $\ln q$; if α were 1, it would be fully exponential in $\ln q$). Since the algorithm uses operations in a ring embedded in the finite field, its description is different for different types of fields. To begin, I describe the basic version of the algorithm, which has run time generally $O(L[q, \frac{1}{2}, c])$, with c a little too large to be practical in large fields. In later subsections, we will look at faster versions of the method.

For the case F_p , p prime, whose elements are represented by the set $\{0, 1, \dots, p-1\}$, with arithmetic performed modulo p : Let S (the *factor base*), be the set of all prime integers less than or equal some bound b . An element of F_p^* is said to be *smooth* with respect to b if, in the ring \mathbb{Z} , all of its factors are contained in S . The algorithm proceeds in three stages. In the first stage, we take a random integer z in $[1, p-2]$, calculate $g^z \pmod{p}$, and see if $g^z \pmod{p}$ is smooth. If it is smooth, say

$$g^z \pmod{p} = \prod_{i=1}^{|S|} p_i^{\alpha_i}, \quad p_i \in S,$$

then we get an equation in the discrete logarithms to the base g :

$$z \equiv \sum_{i=1}^{|S|} \alpha_i \cdot \log_g p_i \pmod{p-1}, \quad (5.0.1)$$

where z and all of the α_i are known. We continue this process until we have generated more than $|S|$ equations. In stage two of the algorithm, we solve the system of equations (5.0.1) to find a unique solution for the $\log_g p_i$. Then in stage three, we are able to find the discrete logarithm of any $y \in F_p^*$. To do this,

we take z 's at random again, until we find a z such that $yg^z \pmod{p}$ is smooth. When we find such a z , we get an equation

$$\log_g y \equiv -z + \sum_{i=1}^{|S|} \alpha_i \cdot \log_g p_i \pmod{p-1},$$

where everything on the right hand side is known.

For example, in F_{37} with $g = 5$, let $S = \{2, 3\}$.

STAGE 1:

Try $z = 7$: $5^7 \equiv 18 \pmod{37} = 2 \cdot 3^2 \Rightarrow \log_5 2 + 2 \cdot \log_5 3 \equiv 7 \pmod{36}$.

Try $z = 6$: $5^6 \equiv 11 \pmod{37}$. Not smooth.

Try $z = 14$: $5^{14} \equiv 28 \pmod{37}$. Not smooth.

Try $z = 31$: $5^{31} \equiv 24 \pmod{37} = 2^3 \cdot 3 \Rightarrow 3 \cdot \log_5 2 + \log_5 3 \equiv 31 \pmod{36}$.

STAGE 2:

Subtracting 3 times the first equation from the second, and using $5^{-1} \equiv 29 \pmod{36}$, we get the solution $\log_5 2 = 11$ and $\log_5 3 = 34$.

STAGE 3:

Suppose we want to find $\log_5 17$.

Try $z = 24$: $17 \cdot 5^{24} \equiv 35 \pmod{37}$. Not smooth.

Try $z = 15$: $17 \cdot 5^{15} \equiv 12 \pmod{37} = 2^2 \cdot 3$

$\Rightarrow \log_5 17 \equiv -15 + 2 \cdot 11 + 1 \cdot 34 \equiv 5 \pmod{36}$. □

The algorithm works basically the same for the case F_{p^n} , $n \gg p$, whose elements are the set of all polynomials over F_p of degree less than n , with multiplication performed modulo a fixed polynomial $f(x)$ of degree n that is irreducible over F_p . We take S to be the set of all irreducibles in the ring $F_p[x]$ whose degree does not exceed some bound b , and call an element of $F_{p^n}^*$ smooth if (in $F_p[x]$) all of its factors are in S . To see why this simple extension of the algorithm doesn't yield a subexponential running time for arbitrary p and n , consider the case F_{p^2} . We need the products of elements in the factor base to give us a significant percentage of the field elements, but an $ax + b \in F_{p^2}$ can have at most one linear polynomial factor. (In section 5.2 we will see a way to get around this problem.)

Notice that, in general, there is a tradeoff inherent in our choice of how large to make the factor base S . By increasing the size of S , we increase the probability that a random field element is smooth, so the task of generating equations in stage one can be accomplished faster. However, that will also increase the number of unknowns (i.e. the discrete logarithms of all elements in S) in stage two, thus making it more difficult to solve the system of equations. Therefore, we seek to bound $|S|$ to the size that will balance these considerations. On a related topic, notice that the generation of equations in stage one can be done by

a large network of processors working in parallel, with very little communication between them, while the linear algebra in stage two is difficult to parallelize.

For large fields, the algorithm requires the solution of very large, but extremely sparse (and not too prone to exhibit linear dependence), systems of linear equations. Furthermore, the columns corresponding to the smaller irreducibles in the factor base will tend to be the most dense, while the columns corresponding to the larger irreducibles will be the most sparse. To solve the system, we can use a very effective method known as ‘structured Gaussian elimination’ to try to eliminate the sparsest columns and the most dense rows first. This can reduce the system by as much as 95% (especially if we have generated many more equations than unknowns). The reduced system is still sparse enough to be solved by using other sparse matrix techniques, so the entire system can be solved in time roughly $|S|^2$ (see [20, 30]). (To get a better sense of the sizes involved, a system arising from $|F_p| \approx 10^{58}$ had $|S| \approx 100,000$ and an average 15.5 non-zeros per equation. Structured Gaussian elimination reduced it to around 6,000 unknowns, and an average 80.5 non-zeros per equation.)

The fact that we need to solve the system of equations mod $q - 1$ is not too troubling. We can try to perform Gaussian elimination as though we were in a field, and possibly succeed. We will have a problem though, if we have to perform it on a column in which every entry is non-invertible mod $q - 1$. At worst, we can solve the equations modulo the prime divisors of $q - 1$, use Hensel’s lemma to lift the solutions to the appropriate powers of the divisors, then use the Chinese Remainder Theorem to find the solution mod $q - 1$. However, McCurley [25] pointed out a simpler possibility: Assume we are in column j ; have a non-zero entry in position jj (if not, switch rows); and want to introduce a zero in position ij . We use the Extended Euclidean Algorithm to find integers d, r , and s , such that $d = ra_{ij} + sa_{jj}$. Now replace row j by $r \cdot (\text{row } i) + s \cdot (\text{row } j)$, and replace row i by $(\frac{a_{ij}}{d}) \cdot (\text{row } j) - (\frac{a_{jj}}{d}) \cdot (\text{row } i)$. This will leave a zero in position ij , and d in position jj . If after doing this (possibly many times in the same column), d is invertible mod $q - 1$, then we’ve overcome the problem. In the least, we will have simplified the problem.

Analyzing index-calculus algorithms depends on knowing the probabilities that random field elements are smooth. These probabilities are now pretty well understood for the relative sizes of $|S|$ that are of interest. In prime fields F_p with smoothness bound $b = L[p, \frac{1}{2}, \beta]$, the work of Canfield, Erdős, and Pomerance [7] shows that we will probably have to test around $L[p, \frac{1}{2}, \frac{\alpha}{2\beta}]$ random integers between 1 and p^α to find one integer that is smooth. In fields F_{2^n} , Odlyzko [30] has shown that we will have to test approximately $\exp((1 + o(1))\frac{k}{b} \ln(\frac{k}{b}))$ degree k polynomials to find one that has all of its irreducible factors of degree b or

less. (For other fields F_{p^n} , see the survey article [31].) In general, the larger the magnitude of the field element we are testing, the less likely it will be smooth. Therefore, an effective way to speed up the algorithm is to reduce the magnitude of the field elements that we test for smoothness (but this must be done in such a way that we can still get an equation among the discrete logarithms of the irreducibles in S).

5.1 COPPERSMITH'S ALGORITHM

Coppersmith [8] described and analyzed his algorithm for fields F_{2^n} , though it can be generalized to other fields F_{p^n} of prime power order, with p growing slowly as $p^n \rightarrow \infty$ (it cannot be applied in prime fields F_p). Whereas in the basic version of the index-calculus algorithm we would try to factor elements of degree $< n$ into irreducibles of degree $\leq b \approx n^{\frac{1}{2}} \ln^{\frac{1}{2}} n$, in Coppersmith's algorithm we will take $b \approx n^{\frac{1}{3}} \ln^{\frac{2}{3}} n$ and test elements of degree $\leq n^{\frac{2}{3}} \ln^{\frac{1}{3}} n$ for smoothness. Thus, the algorithm achieves a run time of the form $L[2^n, \frac{1}{3}, c]$, where c oscillates to as high as 1.5874.

Coppersmith's algorithm was inspired by an idea that Blake, Fuji-Hara, Mullin, and Vanstone [4] referred to as 'systematic equations', in which we try to generate some of the equations for stage one by raising the irreducibles in S to a power of 2. For example, suppose we are in the field F_{2^7} defined by $f(x) = x^7 + x + 1$, and S contains all irreducible polynomials of degree at least as large as 2. Then $x^8 \equiv x^2 + x \pmod{f(x)}$, and if we raise $x + 1 \in S$ to the 8th power, we get an equation relating the discrete logarithms of irreducibles in S :

$$\begin{aligned} (x + 1)^{2^3} &= x^{2^3} + 1 \text{ (since we are in a field of characteristic 2)} \\ &\equiv x^2 + x + 1 \pmod{f(x)} \in S \\ \Rightarrow 8 \log_g(x + 1) &\equiv \log_g(x^2 + x + 1) \pmod{2^7 - 1}. \end{aligned}$$

Unfortunately, we can get at most $\frac{1}{2}$ of the equations we need by this method (see [30, sec. 5.1]). (Blake, et.al. also describe a method wherein we use the Extended Euclidean Algorithm to reduce the problem of finding a polynomial of degree $\approx n$ that is smooth, into the more probable task of finding two polynomials of degree $\approx \frac{n}{2}$ that are both smooth.)

Before describing Coppersmith's algorithm, we need to define some more notation. Let the irreducible polynomial that defines F_{2^n} be $f(x) = x^n + f_1(x)$. We need $\deg(f_1(x)) < n^{\frac{2}{3}}$, and heuristically we expect to be able to find an $f_1(x)$ of degree $\approx \log_2 n$. If we are given the task of finding discrete logarithms in a representation of the field that doesn't meet this requirement, Zierler [43] has shown that it is relatively easy to find the isomorphism between the representation we are given, and any representation that we choose. So we can

use our preferred $f(x)$, then transfer our results back to the given representation.

Let $k \in \mathbb{Z}$ such that $2^k \approx n^{\frac{1}{3}} \ln^{-\frac{1}{3}} n$, let $h = \lceil \frac{n}{2^k} \rceil$ ($\approx n^{\frac{2}{3}} \ln^{\frac{1}{3}} n$), and let $B \approx n^{\frac{1}{3}} \ln^{\frac{2}{3}} n$. To generate an equation in stage one, we choose a $u_1(x)$ and $u_2(x)$ of degrees $\leq B$, with $(u_1(x), u_2(x)) = 1$, and set

$$\begin{aligned} w_1(x) &= u_1(x)x^h + u_2(x), \text{ and} \\ w_2(x) &\equiv w_1(x)^{2^k} \pmod{f(x)}. \end{aligned}$$

If $w_1(x)$ and $w_2(x)$ are both smooth, then since

$$\log_g w_2(x) \equiv 2^k \log_g w_1(x) \pmod{2^n - 1},$$

we get an equation relating the discrete logarithms of elements in S . Furthermore,

$$\begin{aligned} w_2(x) &\equiv u_1(x^{2^k})x^{h2^k} + u_2(x^{2^k}) \pmod{f(x)} \\ &= u_1(x^{2^k})x^{h2^k - n} f_1(x) + u_2(x^{2^k}), \end{aligned}$$

and $h2^k - n$ equals some c , $0 \leq c < 2^k$, so both $w_1(x)$ and $w_2(x)$ have degrees $\leq O(n^{\frac{2}{3}})$. Also, given the probability that both $w_1(x)$ and $w_2(x)$ will be smooth, B is large enough that we should have enough relatively prime pairs $(u_1(x), u_2(x))$ to generate more than $|S|$ equations, where the bound on S is $b \approx n^{\frac{1}{3}} \ln^{\frac{2}{3}} n$.

For an example, in F_{2^7} defined by $f(x) = x^7 + x + 1$, let $2^k = 2$, $h = 4$, and $B = 3$. To generate one equation, let $u_1(x) = 1$ and $u_2(x) = x$. Then

$$\begin{aligned} w_1(x) &= (1)x^4 + x, \text{ and} \\ w_2(x) &\equiv w_1(x)^2 = (x^4 + x)^2 = x^8 + x^2 \equiv (x^2 + x) + x^2 \equiv x \pmod{f(x)}. \end{aligned}$$

Since $x^4 + x = x(x+1)(x^2+x+1)$, we get the equation:

$$\log_g x \equiv 2(\log_g x + \log_g(x+1) + \log_g(x^2+x+1)) \pmod{2^7 - 1}. \quad \square$$

Fortunately, we can test polynomials for smoothness before we try to factor them. Coppersmith noted that to test whether a $w(x)$ is smooth with respect to the bound b , we can check the necessary (though not sufficient) condition:

$$w'(x) \prod_{i=\lceil \frac{b}{2} \rceil}^b (x^{2^i} - x) \equiv 0 \pmod{w(x)},$$

where $w'(x)$ is the formal derivative of $w(x)$. That is, $x^{2^i} - x$ is the product of all irreducible polynomials over F_2 whose degree divides i , so if $a(x) \in S$ and $a(x)^k \parallel w(x)$, then $a(x) \mid \prod_{i=\lceil \frac{b}{2} \rceil}^b (x^{2^i} - x)$, and $a(x)^{k-1} \mid w'(x)$. The test is not a sufficient test of smoothness because polynomials of degree $> b$ which appear to an even degree in $w(x)$ will also appear to that degree in $w'(x)$ (since the

field has characteristic 2). However, this situation will occur only rarely, and since polynomials that pass the test still have to be factored, the only harm is a little wasted work. (See also [30] for a different smoothness test, and [17] for a procedure that sieves through u_1, u_2 pairs, taking those that will make $w_1(x)$ smooth.)

In general, the third stage of index-calculus algorithms is so much faster than the other two stages, that not much attention is paid to it. Once you know the discrete logarithms of the irreducibles in the factor base, you have essentially solved the problem in that field, since finding $\log_g y$ for any $y \in F_q^*$ is as simple as finding one smooth $yg^z \in F_q^*$, where z is known. However, with the factor base as small as it is in Coppersmith's algorithm, it would take a long time ($O(\exp(n^{\frac{2}{3}} \ln^{\frac{1}{3}} n))$) to find a random yg^z that is smooth. Therefore, we start by trying to find a yg^z whose irreducible factors $v_i(x)$ all have degree $\leq n^{\frac{2}{3}} \ln^{\frac{1}{3}} n$ (where $\deg(yg^z) < n \Rightarrow$ there are less than n factors $v_i(x)$). We then use a method similar to that used in the first stage of the algorithm, applied recursively (with decreasing bounds), to compute the discrete logarithms of the $v_i(x)$, thus allowing us to calculate $\log_g y \equiv -z + \sum_i \log_g v_i(x) \pmod{2^n - 1}$.

In each recursive step, we get an equation relating the discrete logarithm of an irreducible $v(x)$ in terms of the discrete logarithms of ($< n$) smaller degree irreducibles. After a finite number of recursions, these smaller degree irreducibles will be elements of the factor base, whose discrete logarithms we already know. Each recursive step involves finding a $w_1(x)$ and $w_2(x) \equiv w_1(x)^{2^k} \pmod{f(x)}$, for some k , similar to the procedure used in the first stage of the algorithm, but with the additional condition $v(x) | w_1(x)$. (The coefficients of acceptable $u_1(x)$ and $u_2(x)$ that comprise $w_1(x)$ can be determined by a set of $\deg(v(x))$ linear equations.) If $\frac{w_1(x)}{v(x)}$ and $w_2(x)$ both factor into irreducibles of degree not exceeding some bound $b < \deg(v(x))$, say

$$w_1(x) = v(x) \prod_i s_i(x)^{\alpha_i} \quad \text{and} \quad w_2(x) = \prod_j t_j(x)^{\beta_j},$$

then taking discrete logarithms base g gives us the equation

$$\sum_j \beta_j \log_g t_j(x) \equiv 2^k (\log_g v(x) + \sum_i \alpha_i \log_g s_i(x)) \pmod{2^n - 1},$$

so we can find $\log_g v(x)$ in terms of the $\log_g s_i(x)$ and $\log_g t_j(x)$.

The ideas of Blake, et.al. were successfully implemented [4, 27] in the field $F_{2^{127}}$, and Coppersmith and Davenport [8, 9] implemented Coppersmith's algorithm in $F_{2^{127}}$ (as alluded to earlier, an implementation is deemed successful if it can determine the discrete logarithms of all irreducibles in the factor base

S , without necessarily bothering to find an arbitrary $\log_g y$). These were quite significant at the time, since both Mitre Corporation and Hewlett-Packard Corporation had working models of the Diffie-Hellman key exchange in this field. Using a number of enhancements to Coppersmith's algorithm that Odlyzko [30] had proposed, Gordon and McCurley [17] successfully implemented Coppersmith's algorithm in $F_{2^{227}}$, $F_{2^{313}}$, and $F_{2^{401}}$ (including some success in parallelizing the linear algebra of stage two). They were also able to complete the stage one equation generation in the field $F_{2^{503}}$, but because they didn't have sole access to the computers they were using, they were unable to complete stage two of the algorithm.

5.2 GAUSSIAN INTEGERS METHOD

Coppersmith, Odlyzko, and Schroepel [10] proposed three algorithms for finding discrete logarithms in prime fields F_p , each of which has a run time $L[p, \frac{1}{2}, c = 1]$ for the first and second stages, and a run time of $L[p, \frac{1}{2}, c = \frac{1}{2}]$ for stage three. One of these algorithms, which Coppersmith, et.al. called the Gaussian integers method, has proved to be the most practical, and the most important in the way it has spurred the development of other discrete logarithm algorithms. LaMacchia and Odlyzko [21] successfully implemented the algorithm in a prime field of order $\approx 10^{58}$ (and went halfway through stage two with a prime $\approx 10^{67}$). Although their primary motivation was to obtain empirical results on sparse matrix techniques [20], the implementation with a 58 digit prime also broke an authentication scheme that Sun Microcomputers, Inc. had implemented in that field as part of their Network File System (and which had been incorporated into release 4.0 of UNIX System V).

The Gaussian integers method was inspired by a subexponential algorithm that ElGamal [13] invented for finding discrete logarithms in fields F_{p^2} , $p \rightarrow \infty$. The idea is to perform the index-calculus algorithm in an isomorphic copy of F_{p^2} . If m is a quadratic nonresidue mod p , then (p) is a prime ideal in the ring of quadratic integers $I(\sqrt{m})$, and $F_{p^2} \cong I(\sqrt{m})/(p)$, (e.g. one representation is $\{a\sqrt{m} + b \mid 0 \leq a, b < p - 1; a, b \in \mathbb{Z}\}$). We can find the isomorphism $\phi : F_{p^2} \rightarrow I(\sqrt{m})/(p)$ by adding multiples of p to the coefficients of the irreducible polynomial $f(x)$ that defines F_{p^2} (this doesn't change the representation of F_{p^2}) until the square-free part of the discriminant of $f(x)$ is equal to m . (To use ElGamal's own example: to find the isomorphism from F_{17^2} defined by $f(x) = x^2 + x + 6$ onto $I(\sqrt{-3})/(17)$, we find that $x = 8 + 3\sqrt{-3}$ is a root of $x^2 + (1 - 17)x + (6 + 85) = 0$, so we may let $\phi(ax + b) = a(8 + 3\sqrt{-3}) + b$.) Since the isomorphism preserves products, it also preserves logarithms (to the

base $\phi(g)$). The factor base contains the fundamental unit (i.e. the unit which generates the entire group of units in $I(\sqrt{m})$), and one quadratic prime with prime norm from each associate class in $I(\sqrt{m})$ whose norm is less (in absolute value) than some bound b . (A similar procedure [14] can be used for other fields F_{p^n} with $p \rightarrow \infty$ and fixed $n \neq 1, 2$.)

In the Gaussian integers method, we will use a very simple mapping of F_p to a subset of $\mathbb{Z} \times \mathbb{Z}$. Let r be a small negative integer that is also a quadratic residue modulo p — preferably $r \in \{-1, -2, -3, -7, -11, -19, -43, -67, -163\}$, so we can work in a unique factorization domain, though the algorithm can be modified to work in a non-UFD. (If $p \equiv 1 \pmod{4}$, we can take $r = -1$.) Let W be an integer such that $W^2 \equiv r \pmod{p}$, and let w represent the complex number \sqrt{r} . Find two integers $T, V < \sqrt{p}$ such that $T^2 \equiv rV^2 \pmod{p}$, and let $p' = T + Vw$. To simplify matters, I limit my discussion to the case where $T^2 - rV^2 = p$. Then the norm $N(p') = p \Rightarrow p'$ is prime $\Rightarrow (p')$ is a maximal ideal of $\mathbb{Z}[w]$ and $\mathbb{Z}[w]/(p')$ is isomorphic to F_p . In fact, $\phi : \mathbb{Z}[w]/(p') \rightarrow F_p$ defined by $\phi(e + fw) = e + fW \pmod{p}$ is an isomorphism. Choose a complex prime $G = a_1 + a_2w$ that generates the group of units $(\mathbb{Z}[w]/(p'))^*$ — this G will be the new base for logarithms. Mapping complex numbers $e + fw$ to real numbers $e + fW$, and the base $G = a_1 + a_2w$ to $g = a_1 + a_2W$, preserves logarithms. Let the smoothness bound $b = L[p, \frac{1}{2}, \frac{1}{2}]$, and we let the factor base contain the integer V , all real primes $\leq b$, and all complex primes $x + yw \in \mathbb{Z}[w]$ whose norm is $\leq b$ (including real primes that factor into two complex primes — we can remove redundancy from the factor base before we solve for unknowns in stage two).

Rather than look for random smooth numbers to generate equations in stage one, we will sieve through pairs of small (positive or negative) integers (c_1, c_2) looking for pairs that make $c_1V - c_2T$ smooth with respect to the real primes in the factor base. (A pair (kc_1, kc_2) , for constant k , will give us the same equation as (c_1, c_2) , so we should avoid using such multiples, e.g. by using only relatively prime pairs.) For each (c_1, c_2) that makes $c_1V - c_2T$ smooth, we check to see if $c_1 + c_2w$ is smooth (i.e. factors completely) with respect to the complex primes in the factor base. (Since the norm of a product of quadratic integers is the product of their norms, we can test $c_1 + c_2w$ for smoothness by testing its norm for smoothness with respect to the norms of complex elements in the factor base.) If $c_1 + c_2w$ is also smooth, then we can get an equation among the discrete logarithms base G of elements in the factor base, because

$$\begin{aligned} c_1V - c_2T &= V(c_1 + c_2w) - c_2(T + Vw) \\ &\equiv V(c_1 + c_2w) \pmod{p'}. \end{aligned}$$

To sieve through (c_1, c_2) pairs, we fix a c_1 value, and allocate (and initialize to zeros) an array C indexed by all the possible c_2 values we want to try. For each real prime p_i in the factor base, $p_i \nmid T$, compute $d_1 \equiv c_1 V T^{-1} \pmod{p_i}$, and add $\ln p_i$ to all array elements $C[c_2]$ such that $c_2 \equiv d_1 \pmod{p_i}$. (These are those c_2 such that $p_i \mid (c_1 V - c_2 T)$.) Do the same for $d_2 \equiv c_1 V T^{-1} \pmod{p_i^2}$, adding $\ln p_i$ to all array elements $C[c_2]$ such that $c_2 \equiv d_2 \pmod{p_i^2}$, and the same mod p_i^3 , mod p_i^4 , \dots , until we reach the highest power j such that $p_i^j \leq b$. Additionally, for those c_2 that are congruent to $d_j \pmod{p_i^j}$, we test whether $c_1 V - c_2 T$ has higher powers of p_i as factors, and add appropriate multiples of $\ln p_i$ to $C[c_2]$ if it does (i.e. one $\ln p_i$ for each additional power). After doing this for each real prime in the factor base, we compare the value in a $C[c_2]$ to the value of $\ln(c_1 V - c_2 T)$. If $\ln(c_1 V - c_2 T) \approx C[c_2]$, then $c_1 V - c_2 T$ is probably smooth, whereas $\ln(c_1 V - c_2 T) > C[c_2]$ indicates that $c_1 V - c_2 T$ probably has factors not in the factor base. Since c_1 and c_2 are small, and $T, V < \sqrt{p}$, the probabilities of smoothness are high, and we should get enough equations by sieving through positive integers $c_1, c_2 \leq b$.

To find an individual $\log_g y$ in stage three, we take random z , as in the basic index-calculus algorithm, and try to find a yg^z that is smooth with respect to (real) primes $< L[p, \frac{1}{2}, 2]$ (say $yg^z = \prod_i u_i$). To find the discrete logarithm of a u_i , we can again use sieving to look for c_1, c_2 such that $\frac{c_1 V - c_2 T}{u_i}$ is smooth with respect to the real primes in the factor base (say $\frac{c_1 V - c_2 T}{u_i} = \prod_j p_j^{\alpha_j}$), and for which $c_1 + c_2 w$ is smooth with respect to the complex primes in the factor base (say $c_1 + c_2 w = \prod_k q_k^{\beta_k}$). Then

$$u_i = \frac{c_1 V - c_2 T}{\prod_j p_j^{\alpha_j}} \equiv \frac{V(c_1 + c_2 w)}{\prod_j p_j^{\alpha_j}} \equiv \frac{V \prod_k q_k^{\beta_k}}{\prod_j p_j^{\alpha_j}} \pmod{p'}$$

and taking logarithms gives us $\log_G u_i$ ($= \log_g u_i$) in terms of the discrete logarithms of elements in the factor base.

For an example of stage one in F_{37} , we can let $r = -1$, so $w = i$ and $W = 6$. Let $T = 1$ and $V = 6$, so $p' = 1 + 6i$ and $\phi : \mathbb{Z}[i]/(p') \rightarrow F_{37}$ is an isomorphism defined as $\phi(e + fi) = e + 6f$. Let the generator $G = -1 + i$ be the base for logarithms in $\mathbb{Z}[i]/(p')$. Although the bound on the factor base is less than 3, I'm going to include 3 in order to better illustrate the sieving process. Thus, we take the factor base to be $S = \{6, 2, 3, -1 + i, -1 - i, 1 - i, 1 + i\}$. (It's interesting to notice that we can already deduce the discrete logarithm base G of every element in S : i is the fundamental unit in $\mathbb{Z}[i]/(p')$ and $|\langle i \rangle| = 4 \Rightarrow \log_G i = 9$, $\log_G(-1) = 18$, and $\log_G(-i) = 27$. $6 \equiv i \pmod{p'} \Rightarrow \log_G 6 = 9$. $-1 + i = G \Rightarrow \log_G(-1 + i) = 1$

$\Rightarrow \log_G(-1-i) = 10$, $\log_G(1-i) = 19$, and $\log_G(1+i) = 28$.
 $2 = (-1+i)(-1-i) \Rightarrow \log_G 2 = 11$, and $3 = \frac{6}{2} \Rightarrow \log_G 3 = 9 - 11 \equiv 34 \pmod{36}$.)

I will sieve through all pairs (c_1, c_2) with $c_1 = 2$ and $c_2 = 0, 1, 2, 3$, so we initialize to zero a four element array C indexed from 0 to 3. (Here also, in order to better illustrate what's going on in the sieving process, I increase the bound on the powers of the moduli to $p_i^j \leq 8$.) Since $T^{-1} = 1$ for any modulus, we have:

$$d_1 \equiv c_1 VT^{-1} = 2 \cdot 6 \cdot 1 \equiv 0 \pmod{2} \Rightarrow \text{add } \ln 2 \approx .69 \text{ to } C[0], \text{ and } C[2].$$

$$d_2 \equiv c_1 VT^{-1} = 2 \cdot 6 \cdot 1 \equiv 0 \pmod{4} \Rightarrow \text{add } \ln 2 \approx .69 \text{ to } C[0].$$

$$d_3 \equiv c_1 VT^{-1} = 2 \cdot 6 \cdot 1 \equiv 4 \pmod{8} \Rightarrow \text{no additions.}$$

$$d_1 \equiv c_1 VT^{-1} = 2 \cdot 6 \cdot 1 \equiv 0 \pmod{3} \Rightarrow \text{add } \ln 3 \approx 1.1 \text{ to } C[0], \text{ and } C[3].$$

Test whether 9 is a factor of $c_1 V - 0 \cdot T$: $9 \nmid 12 \Rightarrow$ no further additions to $C[0]$.

Test whether 9 is a factor of $c_1 V - 3T$: $9 \mid 9 \Rightarrow$ add $\ln 3 \approx 1.1$ to $C[3]$.

Test whether 27 is a factor of $c_1 V - 3T$: $27 \nmid 9 \Rightarrow$ no further additions to $C[3]$.

Now we compare:

$$\ln(2 \cdot 6 - 0 \cdot 1) \approx 2.485 \approx C[0] = 2.48 \Rightarrow \text{probably smooth.}$$

$$\ln(2 \cdot 6 - 1 \cdot 1) \approx 2.398 > C[1] = 0 \Rightarrow \text{probably not smooth.}$$

$$\ln(2 \cdot 6 - 2 \cdot 1) \approx 2.303 > C[2] = .69 \Rightarrow \text{probably not smooth.}$$

$$\ln(2 \cdot 6 - 3 \cdot 1) \approx 2.197 \approx C[3] = 2.2 \Rightarrow \text{probably smooth.}$$

$2 \cdot 6 - 0 \cdot 1 = 12 = 2^2 \cdot 3$ is (real) smooth, and $2 + 0i = (1+i)(1-i)$ is (complex) smooth, so from $(c_1 = 2, c_2 = 0)$ we get the equation:

$$2 \log_G 2 + \log_G 3 \equiv \log_G 6 + \log_G(1+i) + \log_G(1-i) \pmod{36}.$$

$2 \cdot 6 - 3 \cdot 1 = 9 = 3^2$ is (real) smooth, but $2 + 3i$ is not (complex) smooth, so we don't get an equation from $(c_1 = 2, c_2 = 3)$. \square

5.3 OTHER DEVELOPMENTS

The Gaussian integers method led to the development of an integer factoring algorithm known as the 'number field sieve', which Gordon [16] then adapted back to the problem of computing discrete logarithms in prime fields F_p . Gordon's number field sieve algorithm has a running time of $L[p, \frac{1}{3}, c \approx 2.0800]$, and a similar adaptation by Schirokauer [37] yields a slightly lower $c \approx 1.9229$. Weber [41] successfully implemented the algorithm in prime fields of order 10^{25} and 10^{40} . Adleman [3] developed a more general 'function field sieve' for all fields F_{p^n} such that $\ln p \leq n^{\epsilon(n)}$, where $0 < \epsilon(n) < 0.98$, and $\epsilon(n)$ approaches zero as $n \rightarrow \infty$. The function field sieve has a run time of the form $L[p^n, \frac{1}{3}, c]$, for some $c > 0$.

With the exception of the early work of Adleman [1] and Hellman and Reyneri [18], all of the index-calculus algorithms mentioned so far rely on unproven,

heuristic assumptions in the analysis of their running times. Pomerance [36] provides rigorously proved algorithms for fields F_q , where q is prime, or a power 2^n . In both cases, the algorithm has a worst case run time $L[q, \frac{1}{2}, \sqrt{2}]$ for stages one and two, and $L[q, \frac{1}{2}, \sqrt{\frac{1}{2}}]$ for stage three. Lovorn [23] has provided rigorously proved algorithms for fields F_{p^n} , with $\ln p \leq n^{0.98}$, and running time $L[p^n, \frac{1}{2}, c]$, for some $c > 0$.

The long-standing question of whether there exists a subexponential algorithm for computing discrete logarithms in all finite fields F_q was finally answered in the affirmative by Adleman and DeMarrais [2]. Their algorithm has a heuristic run time $L[q, \frac{1}{2}, c]$, for some $c > 0$.

6 CONCLUSION

Before closing, I should mention two more papers whose results are of theoretical interest. Shor [39] has given a probabilistic algorithm for computing, on a quantum computer, discrete logarithms in prime fields F_p . The algorithm has a run time which is polynomial in $\ln p$. Sorenson [40] gives parallel algorithms for computing discrete logarithms in prime fields F_p , and fields F_{p^n} (p small), using probabilistic boolean circuits of depth that is polynomial in $\ln p$, and size that is subexponential.

I should also stress, at this point, that there is no proof that computing discrete logarithms is hard — there could yet emerge some new idea that would make the problem easy. Consider, for example, that any cyclic group of order $q - 1$ is isomorphic to the additive cyclic group Z_{q-1} . The discrete logarithm problem in $(Z_{q-1}, +)$ becomes: find x such that $x \cdot g \equiv y \pmod{q-1}$, where g primitive $\Rightarrow (g, q-1) = 1$, so we can easily find $g^{-1} \pmod{q-1}$ using the Extended Euclidean Algorithm. Thus, an easy method of finding an isomorphism from (F_q^*, \cdot) to $(Z_{q-1}, +)$ would make the discrete logarithm problem easy.

More realistically, developments will probably continue along the line of those discussed in section 5.3. Some of the open questions that might be answered in the near future include:

- Does there exist an algorithm for all finite fields F_q with a heuristic running time $L[q, \frac{1}{3}, c], c > 0$?
- Does there exist an algorithm for all finite fields F_q with a rigorously proved running time $L[q, \frac{1}{2}, c], c > 0$?
- Do there exist algorithms with rigorously proved run times $L[q, \frac{1}{3}, c], c > 0$?
- Do there exist algorithms with (heuristic or rigorous) run time $L[q, \frac{1}{3}, 1]$?
- How practical are the newer algorithms? The algorithms are becoming increasingly complicated, and it's not yet clear whether their smaller (expected) running times can be attained in large field implementations.

References

- [1] L.M. ADLEMAN, A subexponential algorithm for the discrete logarithm problem with applications to cryptography, *Proc. of the 20th Annual IEEE Symposium on Foundations of Computer Science* (1979), 55-60.
- [2] L.M. ADLEMAN AND J. DEMARRAIS, A subexponential algorithm for discrete logarithms over all finite fields, *Math. Comp.*, **61** (1993), 1-15.
- [3] L.M. ADLEMAN, The function field sieve, pp. 108-121 in *Proc. Algorithmic Number Theory: First International Symposium, ANTS-I*, Lecture Notes in Computer Science #877, Springer-Verlag, 1994.
- [4] I.F. BLAKE, R. FUJI-HARA, R.C. MULLIN, AND S.A. VANSTONE, Computing logarithms in finite fields of characteristic two, *SIAM J. Algebraic Discrete Methods* **5** (1984), 276-285.
- [5] M. BLUM AND S. MICALI, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. of Comput.* **13** (1984), 850-864.
- [6] B. DEN BOER, Diffie-Hellman is as strong as discrete log for certain primes, pp. 530-539 in *Advances in Cryptology – CRYPTO '88*, Lecture Notes in Computer Science #403, Springer-Verlag, New York, 1990.
- [7] E.R. CANFIELD, P. ERDÖS, AND C. POMERANCE, On a problem of Oppenheim concerning Factorisatio Numerorum, *J. Number Theory* **17** (1983), 1-28.
- [8] D. COPPERSMITH, Fast evaluation of logarithms in fields of characteristic two, *IEEE Transactions on Information Theory* **30** (1984), 587-594.
- [9] D. COPPERSMITH AND J.H. DAVENPORT, An application of factoring, *J. Symbolic Computation* **1** (1985), 241-243.
- [10] D. COPPERSMITH, A. ODLYZKO, AND R. SCHROEPEL, Discrete logarithms in $GF(p)$, *Algorithmica* **1** (1986), 1-15.
- [11] W. DIFFIE AND M. HELLMAN, New directions in cryptography, *IEEE Trans. Info. Theory* **22** (1976), 644-654.

- [12] T. ELGAMAL, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Info. Theory* **31** (1985), 469-472.
- [13] T. ELGAMAL, A subexponential-time algorithm for computing discrete logarithms over $GF(p^2)$, *IEEE Trans. Info. Theory* **31** (1985), 473-481.
- [14] T. ELGAMAL, On computing logarithms over finite fields, pp.396-402 in *Advances in Cryptology – CRYPTO '85*, Lecture Notes in Computer Science #218, Springer, 1986.
- [15] K.F. GAUSS, *Disquisitiones Arithmeticae*, Leipzig, Fleischer, 1801. Translation into English by Arthur A. Clarke, reprinted by Springer-Verlag, New York, 1986.
- [16] D.M. GORDON, Discrete logarithms in $GF(p)$ using the number field sieve, *SIAM J. Discr. Math.* **6** (1993), 124-138.
- [17] D.M. GORDON AND K.S. MCCURLEY, Massively parallel computation of discrete logarithms, pp. 312-323 in *Advances in Cryptology – CRYPTO '92*, Lecture Notes in Computer Science #740, Springer, 1993.
- [18] M.E. HELLMAN AND J.M. REYNERI, Fast computation of discrete logarithms in $GF(q)$, pp. 3-13 in *Advances in Cryptology: Proceedings of CRYPTO '82*, Plenum Press, 1983.
- [19] M. KRAITCHIK, *Théorie des nombres*, Vol. 1, Gauthier-Villars, Paris, 1922.
- [20] B.A. LAMACCHIA AND A.M. ODLYZKO, Solving large sparse linear systems over finite fields, pp. 109-133 in *Advances in Cryptology – CRYPTO '90*, Lecture Notes in Computer Science #537, Springer, 1991.
- [21] B.A. LAMACCHIA AND A.M. ODLYZKO, Computation of discrete logarithms in prime fields, *Designs, Codes, and Cryptography* **1** (1991), 46-62.
- [22] D.L. LONG AND A. WIGDERSON, The discrete logarithm hides $O(\log n)$ bits, *SIAM J. Comput.* **17** (1988), 363-372.
- [23] R. LOVORN, Rigorous, subexponential algorithms for discrete logarithms over finite fields, Ph.D. thesis, University of Georgia, May 1992.
- [24] U.M. MAURER, Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms, pp. 271-281 in *Advances in Cryptology – CRYPTO '94*, Lecture Notes in Computer Science #839, Springer, 1994.

- [25] K.S. McCURLEY, The discrete logarithm problem, pp. 49-74 in *Cryptography and Computational Number Theory*, C. Pomerance, ed., *Proc. Symp. Appl. Math* **42** Amer. Math. Soc., 1990, pp. 49-74.
- [26] G. MULLEN AND D. WHITE, A polynomial representation for logarithms in $GF(q)$, *Acta Arithmetica* **47** (1986), 255-261.
- [27] R. MULLIN, E. NEMETH, AND N. WEIDENHOFER, Will public key crypto systems live up to their expectations? pp. 193-196 in *Proc. 1984 International Conf. on Parallel Processing*, IEEE Press, 1984.
- [28] NATIONAL INSTITUTE FOR STANDARDS AND TECHNOLOGY, Digital Signature Standard, FIPS PUB 186, Computer Systems Laboratory, Gaithersburg, MD., May 19, 1994.
- [29] H. NIEDERREITER, A short proof for explicit formulas for discrete logarithms in finite fields, *Applicable Algebra in Eng., Comm., and Comp.* **1** (1990), 55-57.
- [30] A.M. ODLYZKO, Discrete logarithms in finite fields and their cryptographic significance, pp. 224-314 in *Advances in Cryptology – Eurocrypt '84*, Lecture Notes in Computer Science #209, Springer-Verlag, 1985.
- [31] A.M. ODLYZKO, Discrete logarithms and smooth polynomials pp. 269-278 in *Finite Fields: Theory, Applications and Algorithms*, G.L. Mullen and P. Shiue, eds., American Math. Society, Contemporary Math Series #168, 1994.
- [32] P. VAN OORSCHOT, A comparison of practical public key cryptosystems based on integer factorization and discrete logarithms, pp. 289-322 in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, ed., IEEE Press, New York, 1992.
- [33] R. PERALTA, Simultaneous security of bits in the discrete log, pp. 62-72 in *Advances in Cryptology – Eurocrypt '85*, Lecture Notes in Computer Science #219, Springer-Verlag, New York, 1986.
- [34] S.C. POHLIG AND M.E. HELLMAN, An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Trans. Info. Theory* **24** (1978), 106-110.
- [35] J.M. POLLARD, Monte Carlo methods for index computation (mod p), *Math. Computation* **32** (1978), 918-924.

- [36] C. POMERANCE, Fast, rigorous factorization and discrete logarithm algorithms, pp. 119-143 in *Discrete Algorithms and Complexity*, (Proc. of the Japan-U.S. Joint Seminar, June 4, 1986, Kyoto, Japan) D.S. Johnson, T. Nishizaki, A. Nozaki, H.W. Wilf, eds., Academic Press, 1987.
- [37] O. SCHIROKAUER, Discrete logarithms and local units, *Phil. Trans. Royal Soc. London, A* **345** (1993), 409-423.
- [38] D. SHANKS, Class number, a theory of factorization, and genera, pp. 415-440 in *1969 Number Theory Institute, Proc. Symposium Pure Mathematics*, v.20, American Mathematical Society, 1971.
- [39] P.W. SHOR, Algorithms for quantum computation: discrete logarithms and factoring, pp. 124-134 in *Proc. 35th Annual Symposium on Foundations of Computer Science*, IEEE Press, 1994.
- [40] J.P. SORENSON, Polylog depth circuits for integer factoring and discrete logarithms, *Information and Computation*, **110** (1994), 1-18.
- [41] D. WEBER, An implementation of the general number field sieve to compute discrete logarithms mod p , pp. 95-105 in *Advances in Cryptology – Eurocrypt '95*, Lecture Notes in Computer Science #921, Springer-Verlag, New York, 1995.
- [42] A.L. WELLS, JR., A polynomial form for logarithms modulo a prime, *IEEE Trans. Info. Theory* **30** (1984), 845-846.
- [43] N. ZIERLER, A conversion algorithm for logarithms on $GF(2^n)$, *J. Pure Appl. Algebra* **4** (1974), 353-356.