ISSN (Print): 2476-8316
ISSN (Online): 2635-3490
Dutse Journal of Pure and Applied Sciences (DUJOPAS), *Vol. 5 No. 1a June 2019*

A Survey of Discrete Logarithm Algorithms in Finite Fields

[1*]Ali Maianguwa Shuaibu, [2]Sunday Babuba & [3]James Andrawus
[1,2,3]Department of Mathematics,
Federal University Dutse,
Jigawa State
Email: shuaibuali16@gmail.com

Abstract

The discrete logarithm in a finite group of large order has been widely applied in public key cryptosystem. In this paper, we investigate attempts to solve the discrete logarithm problem, leading towards finding the current computationally best algorithms for performing the backwards computation. Several proposed algorithms for computing discrete logarithms are known and we briefly discuss some of them. Furthermore, we present the most powerful general-purpose algorithm that is known today, called the index-calculus algorithm, and analyze its asymptotic performance. Finally, we discuss several technical issues that are important to the performance of the index-calculus algorithm, such as rapid methods to solve the systems of linear equations that arise in it.

Keywords: Collisions, Discrete logarithm, Elliptic curve, Finite field.

Introduction

The most obvious approach to breaking modern cryptosystems is to attack the underlying mathematical problem. Before the mid-1970s, the main applications for discrete logarithms were similar to those of ordinary logarithms in routine computations. Discrete logarithms allowed the use of easier additions in place of relatively hard multiplications. What was frequently used was Zech's logarithm (also called Jacobi's logarithm (Lidland & Niederreiter, 1983)), which is a modification of the ordinary discrete logarithm. In a finite field F with primitive element g, Zech's logarithm of an integer n is defined as the integer $Z(n) \bmod (q-1)$ which satisfies $g^{Z(n)} = 1 + g^n$. This provides a quick way to add elements given in terms of their discrete logarithms, except for boundary cases such as $g^m + g^n = g^m(1 + g^{n-m}) = g^{m+Z(n-m)}$.

Similar to ordinary logarithms, where slide rules and logarithm tables have been replaced by calculators, discrete logarithms in small or moderately large fields now rely on computer algebra systems for such routine applications.

Interest in discrete logarithms increased tremendously in the mid-1970s with the invention of public key cryptography (Stallings, 2011).It is noted that discrete exponentiation is easy to compute, but the discrete logarithm and its inverse, appeared difficult. This actually motivated the invention of the Diffe-Hellman key exchange protocol which is the first

*Author for Correspondence

practical public key cryptosystem(Odlyzko, 2013).The Diffe-Hellman problem is to compute g^{xy}, the key that the two parties to the Diffe-Hellman protocol obtained from the g^x and g^y that are visible to the eavesdropper. Although this problem has attracted a lot of attention from researchers, it has not been solved. It is still unknown whether the Diffe-Hellman problem is as difficult as the discrete logarithm problem for the most important cases of finite field and elliptic curve discrete logarithms (Bresson, Chevassut and Pointcheval, 2003).

Other crypto graphic systems different from the Diffe-Hellman crypto system whose security similarly depends on the intractability of the discrete logarithm problem have been proposed. Many of them can be used in settings other than finite fields.

Some basic definitions that are relevant to this study are presented as follows:

Definition 1
A group (G, ·) consists of a set G and an operation ·, satisfying

1. Associative: a ·(b ·c) = (a ·b) ·c,∀a,b,c ∈G

2. Identity element: ∃1∈G,∀a∈G: a ·1 = 1 ·a = a

3. Inverse element: ∀a∈G,∃a^{-1} ∈G: a ·a^{-1} = a^{-1} ·a = 1

4. Abelian (extra): ∀a,b ∈G: a ·b = b · a

The order of an element a of a group (G, ·) is n, iff n is the smallest positive number such that a · a ····· a = 1.

The set{a,a^2, ...,a^{n-1},a^n = 1}is called the group generated by a and denoted $\langle a \rangle$.

Definition 2 (Finite Field)
Let p be a prime, then $F_p = \mathbf{Z} / p\mathbf{Z}$ is a finite field with p elements. Every non-zero element has inverse for multiplication and thus: $F_p^*, \times$ has $p-1$ elements.

Definition 3 (Discrete Logarithm Problem)
Given a finite field F, a primitive element g of F, and a nonzero element h of F, the discrete logarithm of h to base g, written as $\log_g h$, is the least non-negative integer n such that $h = g^n$.

The value $\log_g h$ is unique modulo q−1, and $0 \le \log_g h \le q-2$. It is often convenient to allow it to be represented by any integer n such that $h = g^n$. The discrete logarithm of h to base g is often called the index of h with respect to the base g. In general, discrete logarithms defined on groups are usually called generic discrete logarithms(Odlyzko, 2013).

Definition 4
If G is a group (with multiplication as group operation), and $g \in G$ of finite order m, then for any element h of (g), the cyclic subgroup of G generated by g, the discrete logarithm of h to base g, written as $\log_g h$, is the least non-negative integer n such that $h = g^n$ (and therefore $0 \le \log_g h \le m-1$).

The difficulty of the discrete logarithm problem depends on the group G. It is called very easy if it can be solved using polynomial time algorithm, e.g. (Z_N, +). It is called hard if can be

solved using sub-exponential time algorithm, e.g. (F_p, ×). It is called very hard if it can be solved using exponential time algorithm, e.g. elliptic curve groups

According to Odlyzko (2013), the definition of a group discrete logarithm allows for consideration of discrete logarithms in finite fields when the base g is not primitive, provided the argument itself is in the group (g). This situation arises in some important applications, particularly in the U.S. government standard for the Digital Signature Algorithm (DSA). DSA operations are performed in a field F_P with p a prime (nowadays recommended to be at least 2048 bits). This prime p is selected so that $p-1$ is divisible by a much smaller prime r (specified in the standard to be of 160, 224, or 256 bits), and an element h of F_P is chosen to have multiplicative order r (say by finding a primitive element g of F_P and setting $h = g^{(p-1)/r}$). The main element of the signature is of the form h^s for an integer s, and ability to compute s would break DSA. DSA can be attacked either by using generic finite group discrete logarithm algorithms in the group $\langle \mathrm{h} \rangle$ or finite field algorithms in the field F_P (which can then easily yield a solution in $\langle \mathrm{h} \rangle$).

The basic properties of discrete logarithms such as the change of base formula, apply universally. However, it observed that many of the known discrete logarithm algorithms are only valid in finite fields. In General, discrete logarithms are easier to compute in finite fields, since they have a rich algebraic structure that can be exploited for cryptanalytic purposes. Most of the researches on discrete logarithms in other settings has been devoted to embedding the relevant groups inside finite fields in order to apply finite field discrete logarithm algorithms.

There is a close relationship between integer factorization and discrete logarithms in finite fields, as most of the algorithms in integer factorization have similar ones in discrete logarithms. In general, less attention has been devoted to discrete logarithms compared to integer factorization. This is as a result of the greater technical difficulty of the discrete logarithm problem as compared to integer factorization as well as less attention being devoted to discrete logarithm problems (Odlyzko, 2013).

Computation of Discrete Logarithms

In this section we give a few methods for computing discrete logarithms. In general, discrete logarithm computation appears to be relatively difficult when compared to factoring, and many of the algorithms in cryptosystems share the same kinds of underlying ideas. If we attempt to use the "brute-force" method of substituting each possible answer at a time, it will be very time consuming.

When one can solved discrete logarithm problems where the base is a given primitive root a , then the change-of-base formula gives us a simple linear congruence $\log_a x \log_x y = \log y (\mathrm{mod}\ p-1)$ and we can easily solve for $\log_x y$. Thus, we will assume throughout this work that we seek to compute a discrete logarithm whose base is a primitive root, since this is the most general case.

The Silver–Pohlig–Hellman algorithm

The discrete logarithm problem becomes easier to solve if the order of the element g is factored partially. The Silver–Pohlig–Hellman method(Pohlig and Hellman,1978) states as follows: Suppose that g is an element of finite order m in a group G, and m has m_1 and m_2 as factors such that $m = m_1 m_2$ with $\gcd(m_1, m_2) = 1$. Then the cyclic group $\langle g \rangle$ is the direct

product of the cyclic groups $\langle g^{m_2} \rangle$ and $\langle g^{m_1} \rangle$ of orders m_1 and m_2, respectively. If we determine $a = \log_{g^{m_2}}(w^{m_2})$ and $b = \log_{g^{m_1}}(w^{m_1})$, the Chinese Remainder Theorem (CRT) tells us that $\log_g w$ is determined completely, and in fact we obtain

$$\log_g w = b * x * m_1 + a * y * m_2 (\mathrm{mod}\, m),$$

where x and y come from the Euclidean algorithm computation of $\gcd(m_1, m_2) = 1$ i.e., $1 = xm_1 + ym_2$. This procedure extends easily to more than two relatively prime factors.

Given $|g| = m$, is a prime power, say $m = p^k$, then the computation of $\log_g w$ reduces to k discrete logarithm computations in a cyclic group of p elements. For example, if $r = p^{k-1}$ and $h = g^r$, $u = w^r$, then h has order p, and computing $\log_h u$ yields the reduction of $\log_g w (\mathrm{mod}\, p)$. This process can then be iterated to obtain reduction modulo p^2, and so on.

The above remarks, together with the results of the rho method show when the complete factorization of the order of g can be obtained. In fact discrete logarithms can be computed in not more than $r^{\frac{1}{2}}$ operations in the group, where r is the largest prime in the factorization process(Odlyzko, 2013).

It is noted that any function can be represented by a polynomial in a finite field. It turn out that such polynomials do turn out to have some esthetically pleasing properties for the discrete logarithm(see, Zhe-Xian 2008,Wells 1984).

Baby steps–giant steps algorithm

One way of solving the discrete logarithm problem quicker is the baby step-giant step method. Baby steps–giant steps algorithm states as follows: Suppose that G is a group and g is an element of G of finite order m. If $h \in \langle g \rangle, h = g^k$, and $w = \left\lceil m^{1/2} \right\rceil$, then k can be written as $k = aw + b$ for some a, b with $0 \le a, b < w$.

For instance, if we want to compute the discrete logarithm of x in $a^x = b (\mathrm{mod}\, p)$. First we select an integer k such that $k \ge \sqrt{p}$. Next, calculate all the a^x from $1 \le x \le k$ and record all values (This is called the baby step). Third, we apply $ba^{(-k)x}$ where $-k$ is the inverse of k until we arrive at a value that was already recorded in the baby step (This is called giant step). More specifically, we terminate when we find m and n such that $a^m \equiv l (\mathrm{mod}\, p)$, and $ba^{-kn} \equiv l (\mathrm{mod}\, p)$. Then we obtain $a^m \equiv ba^{-kn} (\mathrm{mod}\, p)$, which is $a^m a^{kn} \equiv b (\mathrm{mod}\, p)$. Therefore, we arrive at the final result $x = m + nk$.

Example 1(Shi, 2018).

Solve for x when $7^x \equiv 210 (\mathrm{mod}\, 359)$. First, we solve $\sqrt{p}$ √p to find $k, \sqrt{p} = \sqrt{359} \approx 18.95$. So we take $k = 19$.

x	1	2	. . .	. . .	13	14	. . .	. . .	19
7^x	7	49	. . .	. . .	309	9	. . .	. . .	124
$210(7)^{-19x}$	297	179	. . .	. . .	9	. . .	. . .	. . .	. . .

From the table above, we noticed that $7^{14} \equiv 9(\mathrm{mod}\,359)$, and also $210(7)^{13(-19)} \equiv 9(\mathrm{mod}\,359)$. Therefore, we conclude that $7^{14} \equiv 210(7)^{13(-19)}(\mathrm{mod}\,359)$, which can be rewritten as $7^{14}7^{13(19)} \equiv 210(\mathrm{mod}\,359)$. Hence, $7^{216} \equiv 210(\mathrm{mod}\,359)$. The discrete logarithm is 216.

The baby steps–giant steps technique has the advantage of being fully deterministic. Its major disadvantage is that it requires storage of approximately $m^{1/2}$ group elements. However, there is usually a tradeoff between space and time, where a smaller list is stored but more computations are required. The baby steps–giant steps algorithm extends easily to many cases where the discrete logarithm is restricted in some way (Odlyzko, 2013). For example, if it is known that $\log_g w$ lies in an interval of length n, the basic approach sketched above can be modified to find it in $O(n^{1/2})$ group operations (plus the usual sorting steps). Similarly, if the discrete $\log k$ is allowed to have only small digits when represented in some base (say, base 10), then the running time will be about the square root of the number of possibilities for k.

Pollard rho and kangaroo methods for discrete logarithms

Pollard invented two randomized methods for solving discrete logarithm problems in any group (Pollard, 1978). These methods are the rho method and the kangaroo (or lambda) technique. Pollard's rho method depend on the birthday paradox, considers when one takes a random walk on a completely connected graph of n vertices. Because the order of the group is finite, the sequence will ultimately reach an element that has occurred before. This is called a collision or a match. The advantage of this method is that the space requirements are small if one uses a clever method of detecting a collision. The problem of efficient collision detection of a pseudo-random walk in Pollard's rho method is presented in Wang and Zhang (2012).These discrete logarithm algorithms similar to the rho method depend on the Floyd algorithm for detecting cycles with little memory at some cost in running time, in that they compare x_{2i} to x_i, where x_i is the position of the random walk at time i.

Pollard described the application of the kangaroo algorithm to the discrete logarithm problem in the multiplicative group of units modulo a prime p. However, it is a generic discrete logarithm algorithm that will work in any finite cyclic group.

Suppose G is a finite cyclic group of order n which is generated by the element α, and one wish to find the discrete logarithm x of the element β to the base α. In other words, we need $x \in Z_n$ such that $\alpha^x = \beta$. The kangaroo algorithm allows us to search for x in some subset $\{a, \ldots, b\} \subset Z_n$, and this search may involve the entire range of possible logarithms by setting $a = 0$ and $b = n - 1$. Dummit (2016)notes that Pollard's rho algorithm for discrete logarithms is more efficient in this case.

Since the rho and kangaroo methods for discrete logarithms are probabilistic, they cannot guarantee a solution, but heuristics suggest, and experiments confirm, that both run in

expected time $O(m^{1/2})$, where m is the order of the group. This has the same computational effort as for the baby steps–giant steps algorithm. However, the rho and kangaroo methods have two advantages. One is that they use very little memory. Another one is that, as was first shown by van Oorschot and Wiener (1999), they can be parallelized, with essentially linear speedup, so that k processors find a solution about k times faster than a single one. We briefly explain the standard version of the rho method as follows.

Rho Algorithm for Discrete Logs

Partition the group $\langle g \rangle$ of order m into three roughly equal sets S_1, S_2 and S_3, using some property that is easy to test, such as the first few bits of a canonical representation of the elements of G. To compute $\log_g h$, define a sequence $w_0, w_1, \ldots$ by $w_0 = g$ and for $i > 0, w_{i+1} = w_i^2, w_i g$, or $w_i h$, depending on whether $w_i \in S_1, S_2$ or S_3. Then each w_i is of the form

$$w_i = g^{a_i} h^{b_i}$$

for some integers a_i, b_i. If the procedure of moving from w_i to w_{i+1} behaves like a random walk (as is expected), then in $O(m^{1/2})$ steps we will find i such that $w_i = w_{2i}$, and this will give a congruence

$$a_i + b_i \log_g h \equiv a_{2i} + b_{2i} \log_g h \pmod{m}.$$

Depending on the greatest common divisor of m and $b_i - b_{2i}$ this congruence will typically either yield $\log_g h$ completely, or give some stringent congruence conditions, which with the help of additional runs of the algorithm will provide a complete solution.

The low memory requirements and parallelizability of the rho and kangaroo algorithms have made them the methods of choice for solving general discrete logarithm problems. There is a substantial literature on various modifications, although they do not improve too much on the original parallelization observations of Kim *et al* (2010).

The rho method, as outlined above, requires knowledge of the exact order m of the group. The kangaroo method only requires an approximation to m. The kangaroo algorithm can also be applied effectively when the discrete logarithm is known to lie in a restricted range.

Index calculus algorithms

This section is devoted to a brief overview of index calculus algorithms for discrete logarithms. Shanks (Shanks, 1971) and Pollard (Pollard, 1978) methods mentioned earlier take exponential time, about $m^{1/2}$ for a group of order. However, the index calculus techniques are sub exponential, with running times closer to $\exp(\log m^{1/2})$ and even $\exp(\log m^{1/3})$, and they apply directly only to finite fields. That is why much of the research on discrete logarithms in other groups of cryptographic interest, such as on elliptic curves, is devoted to finding ways to reduce those problems to ones in finite fields. In the case of DSA mentioned earlier, the recommended size of the modulus p has increased very substantially, from 512 to 1024 bits when DSA was first adopted, to the range of 2024 to 3036 bits more recently. The FIPS 186-3 standard specifies bit lengths for the two primes p and r of (1024, 160), (2048, 224), (2048, 256), and (3072, 256). The relative sizes of p and r were selected to offer approximately equal levels of security against index calculus algorithms (p) and generic discrete logarithm

attacks (r). The reason for the much faster growth in the size of p is that with the sub exponential running time estimates, the effect of growing computing power is far more pronounced on the p side than on the r side. In addition, while there has been no substantial theoretical advance in index calculus algorithms in the last two decades, there have been numerous small incremental improvements. On the other hand, there has been practically no progress in generic discrete logarithm algorithms, except for parallelization (Odlyzko, 2013).

The basic idea of index calculus algorithms dates back to Kraitchik (Odlyzko, 2013), and is also key to all fast integer factorization algorithms. In a finite field F with $|F| = q$, and with primitive element g, if we find some elements $x_i, y_j \in F$ such that

$$\prod_{i=1}^{r} x_i = \coprod_{j=1}^{s} y_j$$

Then $$\sum_{i=1}^{r} \log_g x_i = \sum_{j=1}^{s} \log_g y_j \,(\mathrm{mod}\, q-1)$$

If enough equations are collected, this linear system can be solved for the $\log_g x_i$ and $\log_g y_j$ Singular systems are not a problem in practice, since typical computations generate considerably more equations than unknowns, and one can arrange for g itself to appear in the multiplicative relations.

To compute $\log_g w$ for some particular $w \in F$ with index calculus algorithms, it is often necessary to run a second stage that produces a relation involving w and the previously computed discrete logarithms. In some algorithms the second stage is far easier than the initial computation, in others it is of comparable difficulty. For some time now, the best index calculus algorithms for both integer factorization and discrete logarithms had running times of the form (Odlyzko, 1984):

$$\exp((c+o(1))(\log q)^{1/2}(\log\log q)^{1/2}) \quad as \;\; p \to \infty$$

for various constants $c > 0$, where q denotes the integer being factored or the size of the finite field. The first practical method that broke through this running time barrier was Coppersmith's algorithm (Coppersmith, 1984) for discrete logarithms in fields of size $q = 2k$ (and more generally, of size $q = p^k$ where p is a small prime and k is large). It had running time of approximately

$$\exp(\mathrm{C}(\log q)^{1/3}(\log\log q)^{2/3}) \quad ,$$

where the C varied slightly, depending on the distance from k to the nearest power of p, and in the limit as $k \to \infty$ it oscillated between two bounds (Odlyzko, 1984). The function field sieve of Adleman (1994) which also applies to fields with $q = p^k$ where p is relatively small, improves on the Coppersmith method, but has similar asymptotic running time estimate. The running time of Coppersmith's algorithm turned out to also apply to the number field sieve. This method, which uses algebraic integers, was developed for integer factorization byPollard and Hendrik Lenstra (Odlyzko, 2013), with subsequent contributions by many others. It was adopted for discrete logarithm computations in prime fields by Gordon (1993), with substantial improvements by other reseachers (Commeine and Semaev, 2006; Schirokauer,2010).

Smooth integers and smooth polynomials

The index calculus algorithms depend on a multiplicative splitting of some elements into smaller collection, such as integers or polynomials. This smaller collection usually consists of elements that by some measure (norm) are small. The importance of index calculus algorithms is to intelligently select elements from the large set at random, soas to maximize the chances they will have the desired type of splitting. Usually these elements that do have such splittings are called "smooth." There are rigorous analyses that provide estimates of how frequent elements in various domains are "smooth." These estimates exists for ordinary integers, algebraic integers, and polynomials over finite fields (Odlyzko, 2013).

Parse Linear Systems of Equations

Index calculus algorithms for discrete logarithms require the solution of linear equations modulo $q-1$, where q is the size of the field. Similar to the Silver–Pohlig– Hellman method, the Chinese Remainder Theorem reduces the problem to that of solving the system modulo primes r that divide $q-1$.

The linear algebra problems that arise in index calculus algorithms for integer factorization are very similar, but simpler, in that they are all just $\bmod 2$. The measure of how hard a discrete logarithm problem is can be ascertained if it is resistant to the Silver–Pohlig– Hellman attack. Hence $q-1$ has to have at least one large prime factor r, so that the linear system has to be solved modulo a large prime. This increases the complexity of the linear solution computation, and thus provides slightly higher security for discrete logarithm cryptosystems (Odlyzko, 2013).

A major factor that enables the solution of the very large linear systems that arise in index calculus algorithms is that these systems are very sparse. Those "smooth" elements do not involve too many of the "small" elements in the multiplicative relations. Usually the structured gaussian elimination method proposed in Odlyzko (1984) is applied first. It combines the relations in ways that reduce the system to be solved and do not destroy the sparsity too far. In this case, the conjugate gradient, the Lanczos, or the Wiedemann methods that exploit sparsity are used to obtain the final solution(LaMacchia and Odlyzko, 1991). For the extremely very large linear systems that are involved in recordsetting computations, distributed computation is required. The methods of choice, once structured gaussian elimination is applied, are the block Lanczos and block Wiedemann methods (Coppersmith, 1994). Some symbolic algebraic systems incorporate implementations of the sparse linear system solvers mentioned above. As a demonstration of the effiectiveness of the sparse methods, the record factorization of RSA768 produced 64 billion linear relations. These were reduced, using structured gaussian elimination, to a system of almost 200 million equations in about that many unknowns. This system was still sparse, with the average equation involving about 150 unknowns. The block Wiedemann method was then used to solve the resulting system (Kleinjung *et al*, 2010).

Conclusion

Discrete logarithms are quickly computable in a few special cases. However, no efficient method is known for computing them in general. Several important algorithms in base their security on the assumption that the discrete logarithm problem over carefully chosen groups has no efficient solution

Some of the obvious attacks on the cryptographic algorithms can be described in terms of the discrete logarithm problem. In particular if we think of encryption methods as hash functions, their inverses should be hard to compute. Discrete logarithm is roughly an inverse of the functions (Shi, 2018).

The discrete logarithm is not usually used during the encryption or decryption process, but the method of computing discrete logarithms would provide a viable attack on cryptographic protocols. An attacker can use discrete logarithm to directly find private keys so we can get the message easily from Diffie-Helman key exchange. Similarly for RSA, knowing how to compute the discrete logarithm allows attackers to find d_A and then obtain the plaintext message through $(x^{e_A})^{d_A} \pmod{N_A}$, given x^{e_A} and N_A.

There exist groups for which computing discrete logarithms is apparently difficult. In some cases, there is not only no efficient algorithm known for the worst case(e.g. large prime order subgroups of groups $(\mathbf{Z}_p)^\times$), but the average case complexity can be shown to be about as hard as the worst case using random self-reducibility.

At the same time, the inverse problem of discrete exponentiation is not difficult as it can be computed efficiently using exponentiation by squaring. This asymmetry is analogous to the one between integer factorization and integer multiplication. Both asymmetries (and other possibly one-way functions have been exploited in the construction of cryptographic systems.

Popular choices for the group G in discrete logarithm cryptography are the cyclic groups $(\mathbf{Z}_p)^\times$ such as ELGamal encryption, Diffie-Hellman key exchange, the Digital signature algorithm and cyclic subgroups of elliptic curves over finite fields.

Since there is no publicly known algorithm for solving the discrete logarithm problem in general, the first three steps of the number field sieve algorithm only depend on the group G, not on the specific elements of G whose finite log is desired. By precomputing these three steps for a specific group, one need only carry out the last step, which is much less computationally expensive than the first three, to obtain a specific logarithm in that group.

It turns out that much Internet traffic uses one of a handful of groups that are of order 1024 bits or less, e.g. cyclic groups with order of the Oakley primes specified in RFC 2409. The Logjam attack used this vulnerability to compromise a variety of Internet services that allowed the use of groups whose order was a 512-bit prime number, so called export grade.

The authors of the Logjam attack estimate that the much more difficult precomputation needed to solve the discrete logarithm problem for a 1024-bit prime would be within the budget of a large national intelligence agency such as the U.S. National Security Agency (NSA). The Logjam authors speculate that precomputation against widely reused 1024 DH primes is behind claims in leaked NSA documents that NSA is able to break much of current cryptography.

References

Adleman, L. M. (1994). The function field sieve, In Algorithmic number theory (Ithaca, NY, 1994), volume 877 of Lecture Notes in Comput. Sci., pages 108– 121. Springer, Berlin.

Bos, J. and Kaihara, M.E. (2009). Playstation 3 computing breaks 260 barrier: 112-bit prime ECDLP solved, online annoucement, http://lacal.epfl.ch/112bit prime.

Bresson, E., Chevassut, O. and Pointcheval, D. (2003). The group Diffie Hellman problems, In Selected areas in cryptography, volume 2595 of Lecture Notes in Comput. Sci., pages 325–338. Springer, Berlin.

Commeine, A. and Semaev, I. (2006). An algorithm to solve the discrete logarithm problem with the number field sieve, In Public key cryptography—PKC 2006, volume 3958 of Lecture Notes in Comput. Sci., pages 174–190. Springer, Berlin.

Coppersmith, D. (1994). Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm, *Math. Comp.*, 62(205):333–350.

Coppersmith, D. (1984). Fast evaluation of logarithms in fields of characteristic two, IEEE Trans. Inform. Theory, 30(4):587–594.

Dummit, E. (2016).Discrete Logarithms in Cryptography.https://math.la.asu.edu/~dummit/docs/cryptography_3_discrete_logarithms_in_cryptography.pdf

Gordon, D.M. (1993). Discrete logarithms in GF(p) using the number field sieve, SIAM J. *Discrete Math.*, 6(1):124–138.

Kim, J. H., Montenegro, R., Peres,Y. and Tetali, P. A. (2010). birthday paradox for Markov chains with an optimal bound for collision in the Pollard rho algorithm for discrete logarithm, *Ann. Appl. Probab.*, 20(2):495–521.

LaMacchia, B. and Odlyzko, A. (1991). Solving large sparse linear systems over finite fields, In Advances in Cryptology-CRYPT090, volume 537 of Lecture Notes in Computer Science, pages 109–133. Springer.

Lidland, R. and Niederreiter, H. (1983). Finite fields, volume 20 of Encyclopedia of Mathematics and its Applications, Addison-Wesley Publishing Company Advanced Book Program, Reading, MA.

Odlyzko, A. M.(1984). Discrete logarithms in finite fields and their cryptographic significance, In Advances in cryptology (Paris, 1984), volume 209 of Lecture Notes in Comput. Sci., pages 224–314. Springer, Berlin.

Odlyzko, A. M.(2013).Discrete logarithms over finite fields. In *Handbook of Finite Fields*, G. Mullen and D. Panario, eds., CRC Press, pp. 393-401.

Oorschot, P.C. and Wiener, M.J (1999). Parallel collision search with cryptanalytic applications, *J. Cryptology*, 12(1):1–28.

Pohlig, S. C. and Hellman, M. E.(1978). An improved algorithm for computing logarithms over GF(p) and its cryptographic significance, *IEEE Trans. Information Theory, IT-*24(1):106–110.

Pollard, J. M. (1978). Monte Carlo methods for index computation (mod p), Math. Comp., 32(143):918–924.

Shi, X. (2018).Cryptography and Number Theory http://math.uchicago.edu/~may/reu2018/reupapers/shi,xinyu.pdf

Schirokauer, O. (2010). The number field sieve for integers of low weight, Math. Comp., 79(269):583–602.

Stallings, W. (2011).Cryptography and Network Security, Principles and practice, fifth edition,Prentice Hall.

Kleinjung, T., Aoki, K., Franke, J., Lenstra, A. K., Thome, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P. L., Osvik, D.A.(2010). Herman te Riele, Timofeev, A., and Zimmermann, P. Factorization of a 768-bit RSA modulus, In Advances in cryptology—CRYPTO 2010, volume 6223 of Lecture Notes in Comput. Sci., pages 333–350. Springer, Berlin.

Wang, P. and Zhang, F. (2012). An Efficient Collision Detection Method for Computing Discrete Logarithms with Pollard's Rho, *Journal of Applied Mathematics,* Volume 2012,

Wells Jr, A. (1984). A polynomial form for logarithms modulo a prime (corresp.), *Information Theory, IEEE Transactions on,* 30(6):845–846.

Zhe-Xian W. (2008). A shorter proof for an explicit formula for discrete logarithms in finite fields, *Discrete Math.*, 308(21):4914–4915.