The discrete logarithm problem and its application to cryptography

Abhijit Das
Department of Computer Science and Automation
Indian Institute of Science, Bangalore 560 012

Abstract

In this article we describe the discrete logarithm problem (DLP) over finite fields. No algorithms are known to solve the DLP in time bounded by a polynomial in the size of the underlying finite field. The best algorithms known till date are based on the index calculus method and take time 'subexponential' in the size of the field. We describe some variants of the index calculus method for solving the DLP. The computational intractability of the DLP is exploited in designing various cryptosystems, for example, the Diffe-Hellman key agreement scheme, ElGamal data encryption and signature schemes etc. The integer factorization problem (IFP) and the elliptic curve discrete logarithm problem (ECDLP) together with their relations with the DLP are also briefly mentioned.

Contents

1 Background material and notations

In this section, we discuss some common definitions and results from number theory and algebra, that will be useful for the rest of the article. If the reader is already familiar with these concepts, (s)he can skip this section and refer to this section later, if any ambiguity regarding notations is encountered.

In what follows we denote by $\log x$ the logarithm of the positive real number x to some unspecified base (for example, 10). The notations $\lg x$ and $\ln x$ are reserved for denoting logarithm of x respectively to the bases 2 and $e = \sum_{n=0}^{\infty} \frac{1}{n!}$. For a real number x, $\lfloor x \rfloor$ denotes the largest integer less than or equal to x (called the *floor* of x), whereas $\lceil x \rceil$ denotes the smallest integer larger than or equal to x (called the *ceiling* of x).

1.1 Elementary number theory

We denote by $\mathbb{Z}$ the set of whole numbers (integers), namely, $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$. The set of natural numbers is the set $\mathbb{N} = \{1, 2, 3, \ldots\}$. The set of real numbers is denoted by $\mathbb{R}$.

For any integers a and b with $a \neq 0$, there exist (unique) integers q and r satisfying: $b = aq + r$ and $0 \leqslant r < |a|$. The integers q and r are respectively called the *quotient* and the *remainder* of (Euclidean) division of b by a. We denote $r = b \operatorname{rem} a$. If $r = 0$, we say that a divides b and denote this by the notation $a|b$.[1]

For two integers a and b, not both zero, the largest (positive) integer c that divides both a and b is called the *greatest common divisor* of a and b and is denoted as $\gcd(a, b)$. If $d = \gcd(a, b)$, then there are integers u, v such that $d = ua + vb$. If $d = 1$, we say that a and b are *relatively prime*. Given two integers a and b, not both 0, $d = \gcd(a, b)$ and the integers u and v as defined above can be computed *efficiently* using the (extended) Euclidean gcd algorithm (or the (extended) binary gcd algorithm).

The integers 1 and -1 are called *units*. An integer $a \neq 0, 1, -1$ is called a *prime*, if any factorization $a = bc$ implies that either b or c is a unit. In this note an arbitrary prime is usually denoted by p. The sequence of (positive) primes, on the other hand, is denoted by $q_1, q_2, q_3, \ldots$, where $q_1 = 2$, $q_2 = 3$, $q_3 = 5$ and so on. If $a \neq 0, 1, -1$ is not a prime, then a is called *composite*.

Any non-zero integer a can be expressed as a product $a = up_1p_2 \ldots p_k$, where u is a unit and p_i are (positive) primes (not necessarily all distinct). This factorization is *unique* upto permutation of the primes p_i. This factorization can also be expressed as $a = u\mathfrak{p}_1^{\alpha_1}\mathfrak{p}_2^{\alpha_2} \ldots \mathfrak{p}_l^{\alpha_l}$, where $\mathfrak{p}_i$ are *distinct* primes and α_i are non-negative integers.

An integer $x > 0$ is said to be y*-smooth* (for an integer $y > 0$), if all prime factors of x are $\leqslant y$. When y is understood from the context, we simply say that x is smooth. The probability that an integer chosen randomly between 1 and x is y-smooth is asymptotically equal to u^{-u} where $u = \log x / \log y$ (and $x \geqslant y$).[2] This probability is a sensitive function of u and decreases rapidly as we increase u. This means that for a given y, smaller integers have higher chance of being y-smooth.

Given an integer $m > 0$, two integers a and b are said to be *congruent* modulo m, if $m|(a - b)$. This is denoted by $a \equiv b \pmod{m}$. If $a \equiv b \pmod{m}$ and $f(x)$ is a polynomial with integer coefficients, then $f(a) \equiv f(b) \pmod{m}$. A congruence $ax \equiv b \pmod{m}$ is called a linear congruence (in one variable x) and is solvable (for x) if and only if $\gcd(a, m)|b$. In particular, if a and m are coprime, then the congruence $ax \equiv b \pmod{m}$ is solvable for all values of b. If $b = 1$, then x is called the (multiplicative) inverse of a modulo m and is denoted by $x \equiv a^{-1} \pmod{m}$. Note that since $\gcd(a, m) = 1$, we have integers x and y

[1]More correctly, $a|b$ if $b = ac$ for some $c \in \mathbb{Z}$. By this definition $0|0$. Also note that any integer a divides 0, whereas 0 divides no integer other than 0.

[2]Under some conditions on x and y

such that $ax + my = 1$ which means that $ax \equiv 1 \pmod{m}$. Thus modular multiplicative inverses can be computed by the extended gcd algorithm.

1.2 Groups

A *group* G is a set (also denoted by G) together with a binary operation $\diamond$ such that

1. [Closure] For all $a, b \in G$, $a \diamond b \in G$.
2. [Associativity] For all $a, b, c \in G$, $a \diamond (b \diamond c) = (a \diamond b) \diamond c$. Note that the associativity property allows us to write unambiguously $a \diamond b \diamond c$, or more generally, $a_1 \diamond a_2 \diamond \ldots \diamond a_n$ for any non-negative integer n, where $a_i \in G$. The implication is that all possible ways of computing the element $a_1 \diamond a_2 \diamond \ldots \diamond a_n$ (by inserting pairs of parentheses) lead to the same element of G.
3. [Identity] There exists a (unique) element $e \in G$ such that for all $a \in G$, $a \diamond e = e \diamond a = a$.
4. [Inverse] For each $a \in G$, there exists (a unique) $b \in G$ such that $a \diamond b = b \diamond a = e$.

A group G is called *Abelian* or *commutative* if

5. [Commutativity] for all $a, b \in G$, $a \diamond b = b \diamond a$.

The groups that are used for cryptographic purposes are usually Abelian.

For a group G, $a \diamond a \diamond \ldots \diamond a$ (n times) is denoted by na or a^n depending on whether $\diamond$ is identified respectively as an addition operation or a multiplication operation. A group G (not necessarily finite) is called *cyclic*, if there is an element $g \in G$ such that every element $a \in G$ can be written as $a = g \diamond g \diamond \ldots \diamond g$ (n times) for some $n \in \{0, 1, 2, \ldots\}$. The case $n = 0$ gives the identity element e. In this case, we say that G is *generated* by g or g is a *generator* of G.

A group G is called *finite*, if the set G consists of finitely many elements. The cardinality of G is denoted by $|G|$. Let G be a finite *multiplicative* group. For an element $a \in G$, the smallest *positive* integer h such that $a^h = e$ in G is called the (multiplicative) *order* of a. It can be shown that each $a \in G$ has a multiplicative order which divides $|G|$. (Additive orders of elements in a finite additive group can be defined analogously.)

1.3 Fields

A *field* K is a set (also denoted by K) together with two binary operations $+$ and $\cdot$ such that:

1. [Additive closure] For all $a, b \in K$, $a + b \in K$.
2. [Commutativity of $+$] For all $a, b \in K$, $a + b = b + a$.
3. [Associativity of $+$] For all $a, b, c \in K$, $a + (b + c) = (a + b) + c$.
4. [Additive identity] There exists a (unique) element $0 \in K$ such that for all $a \in K$, $a + 0 = 0 + a = a$.
5. [Additive inverse] For each $a \in K$, there exists (a unique) $b \in K$ such that $a + b = b + a = 0$. b is denoted by $-a$.
6. [Multiplicative closure] For all $a, b \in K$, $a \cdot b \in K$.
7. [Commutativity of $\cdot$] For all $a, b \in K$, $a \cdot b = b \cdot a$.
8. [Associativity of $\cdot$] For all $a, b, c \in K$, $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
9. [Multiplicative identity] There exists a (unique) element $1 \in K$ such that for all $a \in K$, $a \cdot 1 = 1 \cdot a = a$.
10. [Multiplicative inverse] For each $a \in K$, $a \neq 0$, there exists (a unique) $b \in K$ such that $a \cdot b = b \cdot a = 1$. b is denoted by a^{-1}.
11. [Distributivity of $\cdot$ over $+$] For all $a, b, c \in K$, $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = a \cdot c + b \cdot c$.

In a field subtraction of b from a, denoted $a - b$, is defined to be the addition of a with the additive inverse of b, that is, $a - b = a + (-b)$. Similarly the division of a by b is defined as $a/b = a \cdot b^{-1}$, if $b \neq 0$. When no ambiguities are likely, we drop the multiplication sign $\cdot$ and represent $a \cdot b$ simply as ab.

Loosely speaking, a field is an algebraic structure where one can add, subtract and multiply any two elements and divide any element by any non-zero element. Note that for a field K, K is an Abelian group under addition $+$; this is called the *additive group* of the field K. Similarly the set of non-zero elements of K, denoted $K^* = K \setminus \{0\}$, is an Abelian group under $\cdot$ (multiplication); this group is called the *multiplicative group* of K.

1.4 Polynomials over fields

A non-zero polynomial $f(x)$ over a field K is a (formal) expression of the form

$$f(x) = a_d x^d + a_{d-1} x^{d-1} + \ldots + a_1 x + a_0,$$

where $a_i \in K, d \in \{0, 1, 2, \ldots\}, a_d \neq 0$. The integer d is called the *degree* of $f(x)$ and is denoted by $\deg f(x)$. a_d is called the *leading coefficient* of $f(x)$. If $a_d = 1$, then $f(x)$ is called *monic*. The *zero* polynomial is denoted by 0 and has degree $-\infty$. A polynomial of degree $\leqslant 0$ is called a *constant* polynomial. The set of all polynomials over K is denoted by $K[x]$.

Arithmetic operations on polynomials ($+$, $-$, $\cdot$ and division with remainder) are performed as demonstrated in high school. The only point to highlight here is that in this case the arithmetic on the coefficients of polynomials is that of the field K. Polynomials over a field K possess properties very similar to those of integers.

For polynomials $f(x) \neq 0$ and $g(x)$, there exist unique polynomials $q(x)$ and $r(x)$ such that $g(x) = q(x)f(x) + r(x)$ where $\deg r(x) < \deg f(x)$. (Note that this includes the case $r(x) = 0$.) We call $q(x)$ and $r(x)$ respectively the quotient and the remainder of (Euclidean) division of $g(x)$ by $f(x)$. We denote $r(x) = g(x) \text{ rem } f(x)$. If $r(x) = 0$, we say that $f(x)$ divides $g(x)$ and denote this as $f(x)|g(x)$.[3]

The *greatest common divisor* of two polynomials $f(x)$ and $g(x)$, not both zero, is the *monic* polynomial $h(x)$ of largest degree such that $h(x)|f(x)$ and $h(x)|g(x)$. We denote $h(x) = \gcd(f(x), g(x))$. In that case, there exist polynomials $u(x)$ and $v(x)$ such that $h(x) = u(x)f(x) + v(x)g(x)$. The gcd $h(x)$ and the polynomials $u(x)$ and $v(x)$ can be computed using an algorithm which is very similar to the extended Euclidean gcd algorithm for integers.

A non-constant polynomial $f(x)$ is called *irreducible* over K if for any factorization $f(x) = g(x)h(x)$ with $g(x), h(x) \in K[x]$, either $g(x)$ or $h(x)$ is a constant polynomial. Otherwise, $f(x)$ is said to be *reducible*.

Any non-zero polynomial $f(x) \in K[x]$ can be expressed as the product $f(x) = uh_1(x)h_2(x)\ldots h_k(x)$, where $u \in K$, and $h_i(x)$ are monic irreducible polynomials in $K[x]$ (not necessarily all distinct). This factorization is *unique* upto permutation of $h_i(x)$. One can also express this factorization as $f(x) = u\mathfrak{h}_1(x)^{\alpha_1}$ $\mathfrak{h}_2(x)^{\alpha_2} \ldots \mathfrak{h}_l(x)^{\alpha_l}$, where $\mathfrak{h}_i(x)$ are *distinct* monic irreducible polynomials and α_i are non-negative integers.

The notion of congruence can be extended to the case of polynomials. We won't go into the details. Instead let us point out that if the gcd of two polynomials $f(x)$ and $g(x)$ is 1, then we can calculate $u(x)$ and $v(x)$ satisfying $u(x)f(x) + v(x)g(x) = 1$ using the extended gcd algorithm. Here $u(x)$ (resp. $v(x)$) is the multiplicative inverse of $f(x)$ (resp. $g(x)$) modulo $g(x)$ (resp. $f(x)$). Thus $u(x) \equiv f(x)^{-1} \pmod{g(x)}$ and $v(x) \equiv g(x)^{-1} \pmod{f(x)}$.

[3] As in the case of integers, the more correct definition of $f(x)|g(x)$ demands the existence of a polynomial $h(x)$ such that $g(x) = f(x)h(x)$. Thus we have $0|0$.

1.5 Finite fields

A *finite field* K is a field such that the cardinality of the set K is finite. Let us denote the cardinality by $q = |K|$. It can be shown that $q = p^n$, where p is a prime and $n \in \mathbb{N}$. Conversely, given any prime p and $n \in \mathbb{N}$, there exists a unique[4] field whose cardinality is $q = p^n$. This field is denoted by $\mathbb{F}_q$. The prime p is called the *characteristic* of the field $\mathbb{F}_q$. Finite fields are also referred as *Galois fields* (and denoted also as $GF(q)$) after the name of Evariste Galois whose seminal work gave birth to the theory of finite fields (and many other branches in modern algebra).

The multiplicative group $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$ is cyclic. Any generator g of $\mathbb{F}_q^*$ is called a *primitive element* of $\mathbb{F}_q^*$. Note that the multiplicative order of a primitive element is $q-1$. An arbitrary element $a \in \mathbb{F}_q^*$ has a multiplicative order which divides $q-1$.

If $q = p$, a prime, $\mathbb{F}_p$ is called a *prime field*. Elements of $\mathbb{F}_p$ are represented as $\{0, 1, 2, \dots, p-1\}$. Arithmetic operations in $\mathbb{F}_p$ are carried out as integer operations modulo the prime p. For example, in order to calculate the sum of two elements a and b in $\mathbb{F}_p$, we compute the integer sum of a and b and take the remainder of division of this sum by p, that is, $a + b = (a + b) \text{ rem } p$. Similarly, $-a = p - a$ if $a \neq 0$, $-0 = 0$, $a - b = (a - b) \text{ rem } p$, $a \cdot b = (ab) \text{ rem } p$. For $a \in \mathbb{F}_p$, $a \neq 0$, we have $\gcd(a, p) = 1$ and hence we compute an integer x satisfying $ax \equiv 1 \pmod p$ by the extended gcd algorithm and take $a^{-1} = x \text{ rem } p$. Finally $a/b = (ab^{-1}) \text{ rem } p$.

If $q = p^n$ for $n > 1$, we represent elements of $\mathbb{F}_q$ as polynomials in $\mathbb{F}_p[x]$ of degree less than n. In order to do arithmetic on these elements, we choose a monic irreducible polynomial $f(x) \in \mathbb{F}_p[x]$ of degree n. It is a well-known result that for any p and n such an irreducible polynomial exists.[5] This polynomial $f(x)$ is called the *defining polynomial* of $\mathbb{F}_q$ for this representation. Arithmetic in $\mathbb{F}_q$ is arithmetic of polynomials over $\mathbb{F}_p$ modulo the defining polynomial $f(x)$. For example, if $g(x)$ and $h(x)$ are two elements of $\mathbb{F}_q$, then their product $g(x) \cdot h(x)$ in $\mathbb{F}_q$ is obtained by computing the product $g(x)h(x)$ and taking the remainder of division of this product by $f(x)$, that is, $g(x) \cdot h(x) = (g(x)h(x)) \text{ rem } f(x)$. Note that the product $g(x)h(x)$ on the right side and the remainder is computed in $\mathbb{F}_p[x]$, that is, coefficient operations are those over the prime field $\mathbb{F}_p$. Other operations are similar. In order to compute the inverse of $g(x) \in \mathbb{F}_q^*$ we note that $\gcd(g(x), f(x)) = 1$, since $f(x)$ is an irreducible polynomial. We then compute using the extended polynomial gcd algorithm a polynomial $u(x)$ such that $u(x)g(x) \equiv 1 \pmod{f(x)}$ and take $g(x)^{-1} = u(x) \bmod f(x)$.

Note that other representations of elements of $\mathbb{F}_q$ are possible. The representations we have described above are the ones most commonly used in practice. For cryptographic applications, the fields $\mathbb{F}_q$ with $q = p$ and $q = 2^n$ are usually employed. Arithmetic over these classes of fields using the methods described above can be implemented relatively efficiently. For the rest of this article, we won't address issues related to the fields $\mathbb{F}_q$ with $q = p^n$ for $n > 1$ and $p \neq 2$.

1.6 Complexity issues

Let us say that we have two positive real-valued functions F and G defined on the set $\mathbb{N}$ of natural numbers. Then we say

$$
\begin{aligned}
F = O(G),&\ \text{if there exists } s_0 \in \mathbb{N} \text{ and } c \in \mathbb{R} \text{ with } c > 0 \text{ such that } F(s) \leqslant cG(s) \text{ for all } s \geqslant s_0,\\
F = \Omega(G),&\ \text{if there exists } s_0 \in \mathbb{N} \text{ and } c \in \mathbb{R} \text{ with } c > 0 \text{ such that } F(s) \geqslant cG(s) \text{ for all } s \geqslant s_0,\\
F = \Theta(G),&\ \text{if } F = O(G) \text{ and } F = \Omega(G),\\
F = o(G),&\ \text{if } F(s)/G(s) \to 0 \text{ as } s \to \infty, \text{ and}\\
F = O\tilde{}(G),&\ \text{if } F = O(G(\log G)^{\delta}) \text{ for some real } \delta \geqslant 0.
\end{aligned}
$$

[4] Here uniqueness is upto *isomorphism*, a term which we choose *not* to describe in this note.

[5] Actually something more general can be said: For any n and any finite field $\mathbb{F}_q$ (q is not necessarily a prime), there is a (monic) irreducible polynomial of degree n in $\mathbb{F}_q[x]$.

For computational problems over a finite field $\mathbb{F}_q$ the input size s is taken to be the value $\lceil \lg q \rceil = \lceil \log_2 q \rceil$, since this is the (maximum) number of bits needed to represent an element in the field. Let the running time of a certain algorithm A for $\mathbb{F}_q$ be $F(s)$. If $F = O(G)$ for some polynomial G in s, then we say that A is a *polynomial-time* algorithm for $\mathbb{F}_q$. If $F = \Omega(a^s)$ for some real $a > 1$, then the algorithm is said to be *exponential*.

---

2 The discrete logarithm problem

Let G be a multiplicatively written cyclic Abelian group of cardinality $|G| = m$ and with identity element e. Let g be an element of G with (multiplicative) order $r|m$. Then the elements $g^0 (= e), g^1, g^2, \ldots, g^{r-1}$ are all distinct. (Note that $g^r = e$, so that $g^{r+t} = g^t$ for all $t \in \mathbb{Z}$.) Given an integer x, $0 \leqslant x < r$, let us denote the power g^x by y. The computation of y from the knowledge of G, g and x is called the *discrete exponentiation problem* (DEP). The integer x (unique modulo r) is called the *discrete logarithm* or *index* of y to the base g and is denoted by $\operatorname{ind}_g y$. The computation of x from the knowledge of G, g and y is called the *discrete logarithm problem* (DLP). Note that if y is an arbitrary element of G and $r \neq m$, then we may not have an integer x for which $y = g^x$. In that case, an algorithm for DLP should return an error message to the effect that the discrete logarithm of y to the base g does not exist. If, however, g is a generator of G, that is, if $r = m$, then $\operatorname{ind}_g y$ exists for all $y \in G$. The DEP and DLP as defined above are the inverse problems of each other (given a fixed G and g).

Now let us take $G = \mathbb{F}_q^*$, the multiplicative group of the finite field $\mathbb{F}_q$. The discrete exponentiation and discrete logarithm problems can be applied to this group. The following observations make these problems useful for cryptographic applications:

- It is *easy* to solve the DEP in $\mathbb{F}_q^*$. Here 'easy' means that there exist algorithms for the DEP, that run in time bounded by a polynomial in the size $s = \lceil \lg q \rceil$ of the field $\mathbb{F}_q$. One such algorithm is the well-known repeated-squaring-and-multiply algorithm. Carefully implemented versions of this algorithm take time $O(s^3)$.

- The DLP, on the other hand, is a very *difficult* computational problem. No algorithms are known to solve this problem in time which is a polynomial function of the size s. Older methods, called square-root methods, take time $O(\sqrt{q})$ in the worst case (This expression is *fully* exponential in s.). Recently (in the last two decades) a family of algorithms based on the *index calculus method* has been proposed. These are currently the fastest algorithms known to solve the DLP and take time *subexponential* in the size s. (We will define the term 'subexponential' later.)

The implication of this is that one can use the DEP as a *one-way function* suitable for cryptography. Various cryptographic schemes have been proposed to exploit this property of DEP. The Diffe-Hellman key agreement scheme and the ElGamal encryption and signature schemes are some examples that we will discuss at length in a later section. It is expedient to note here that it is not *proved* that the DLP is really a computationally difficult problem. Indeed lot of human effort has been devoted to find a satisfactory algorithmic solution of the DLP for quite a long period of time and the best we now have at our disposal are the (slow) methods based on the index calculus attack as mentioned earlier.

For the rest of this article, we describe the methods to solve the DLP. We briefly mention the square root methods. The Pohlig-Hellman method deserves specific mention in this regard, because its running time depends on the largest prime factor of $q - 1$. This fact implies that one must use those $\mathbb{F}_q$ for which $q - 1$ has a suitably large prime divisor. The index calculus method is described next. We start with a generic description of this method. Subsequently, we describe some concrete variants of the method for the fields $\mathbb{F}_p$ and $\mathbb{F}_{2^n}$. Henceforth we will assume that g is a primitive element of $\mathbb{F}_q$, that is, the order of g in $\mathbb{F}_q^*$ is $q - 1$. The description for a non-primitive g is similar, but omitted here in order to avoid unnecessary complications. Note that it is, in general, very difficult (as difficult as the DLP) to find out primitive elements in $\mathbb{F}_q$ or even to check if a given element is primitive. For cryptographic applications however, g need indeed not be primitive. What is sufficient is that the multiplicative order of g be sufficiently large (say, an integer of length at least 160 bits). Some well-known cryptographic protocols which derive their security from the intractability of the DLP are described subsequently. We end this note by suggesting a list of references for further study.

2.1 The Diffe-Hellman problem

The Diffe-Hellman problem (DHP) over finite fields is yet another difficult computational problem that is widely believed to be equivalent to the DLP. This can be generically described as follows. Let again G be a multiplicative (cyclic Abelian) group of cardinality $|G| = m$ and let $g \in G$ have the multiplicative order $r|m$. Finally let a and b be arbitrary integers between 1 and $r - 1$. The computation of g^{ab} from the knowledge of G, g, g^a and g^b is termed the Diffe-Hellman problem. (Note that the knowledge of a and b is assumed to be absent.)

As for DLP, the DHP can be applied to $\mathbb{F}_q^*$. If one can compute discrete logarithms in $\mathbb{F}_q^*$, one can solve the DHP as follows. To start with one computes the index a of g^a and then computes the discrete exponentiation $(g^b)^a = g^{ab}$ in $\mathbb{F}_q^*$. Since discrete exponentiations can be computed rather easily, this implies that if the DLP is solved efficiently, so also is the DHP. However, the reverse implication is not clear. That is what would be the status of the DLP, given an efficient solution of the DHP? Partial results in this direction are only known.

3 Square root methods for solving the DLP

The square root methods are non-practical methods for solving the DLP over $\mathbb{F}_q$. Each of these takes a worst-case running time on the order of $\sqrt{q}$. However they have the advantage (over index-calculus methods) that they can be applied to any arbitrary (cyclic) groups G. Let $|G| = m = \prod_{i=1}^{t} p_i^{\lambda_i}$ be the prime factorization of the cardinality of the (finite) group G. The following three square root methods are well-known. For the sake of brevity, we only mention the running times of these methods and refrain from an algorithmic description of the same.

- *Shank's Baby-Step Giant-Step method* requires $O(\sqrt{m})$ space and runs in time $O(\sqrt{m}\log m)$.
- *Pollard's rho heuristic* runs in expected time $O(\sqrt{m})$.
- The *Pohlig-Hellman method* takes time $O(\sum_{i=1}^{t} \lambda_i(\log m + \sqrt{p_i}\log p_i))$. Thus if $\mathfrak{p}$ is the largest prime factor of m, then the Pohlig-Hellman method runs in time $O(\log m\sqrt{\mathfrak{p}}\log \mathfrak{p})$. In general, $\mathfrak{p}$ is $O(m)$ and hence the running time is $O(\sqrt{m}\log^2 m)$ which is exponential in $\log m$. If, however, all the prime factors (in particular, $\mathfrak{p}$) of m are small, then the Pohlig-Hellman method solves the DLP quite efficiently. This implies that for building a cryptographic application over a finite field $\mathbb{F}_q$ one must choose q such that $q - 1$ has a suitably large prime divisor. When $q = p$, one can construct such values of q (the so-called strong primes and safe primes) rather easily. If $q = 2^n$, then we should know the factorization of the Mersenne numbers $M_n = 2^n - 1$. In general, it is rather difficult to factorize M_n for practical ranges of values of n (512–2048). However, extensive tables of (complete or partial) factorizations of M_n are available. We can pick up a suitable n by investigating the tables.

4 The index calculus method

As we have mentioned earlier, the index calculus method may not apply to arbitrary cyclic groups. But whenever it applies, it is usually the best method for computing discrete logarithms. In particular, several variants of the index calculus method are applicable for the fields $\mathbb{F}_q$ for both $q = p$ and $q = 2^n$. We start with a generic description of the index calculus method and then concentrate on the specific variants. We assume that g is a generator of $\mathbb{F}_q^*$.

4.1 Generic description

Suppose that we want to compute $\operatorname{ind}_g(a)$ in $\mathbb{F}_q^*$. In the index calculus method, we start by choosing a *factor base* B which is a *suitable* subset of $\mathbb{F}_q^*$ of *small* cardinality. Let us denote $B = \{b_1, b_2, \ldots, b_t\}$. We then search for *relations* of the form

$$g^\alpha a^\beta \prod_{i=1}^{t} b_i^{\gamma_i} = \prod_{i=1}^{t} b_i^{\delta_i} \quad (\text{in } \mathbb{F}_q)$$

This gives us a linear congruence

$$\alpha + \beta \operatorname{ind}_g a + \sum_{i=1}^{t} \gamma_i \operatorname{ind}_g b_i \equiv \sum_{i=1}^{t} \delta_i \operatorname{ind}_g b_i \pmod{q-1}$$

The index calculus method proceeds in two stages. In the first stage, we search for relations with $\beta = 0$. When sufficiently many relations are available, the resulting system of linear congruences is solved mod $q-1$ for the unknowns $\operatorname{ind}_g b_i$. In the second stage, a single relation with $\gcd(\beta, q-1) = 1$ is found. Substituting the *precomputed* values of $\operatorname{ind}_g b_i$ yields $\operatorname{ind}_g a$. Note that in the second stage discrete logarithms of any number of elements of $\mathbb{F}_q^*$ can be computed. The values of $\operatorname{ind}_g b_i$ need be precomputed (and stored) only *once* for computing all these individual logarithms.

The running time of the index-calculus method is of the form

$$L\langle q, \omega, c\rangle = \exp\left((c + o(1))(\ln q)^\omega (\ln\ln q)^{1-\omega}\right)$$

for some positive constant c and for some real number $0 < \omega < 1$. By an abuse of notation, we denote by $L(q, c)$ any quantity that satisfies

$$L(q, c) = \exp\left((c + o(1))\sqrt{\ln q \ln\ln q}\right)$$

This corresponds to $\omega = \frac{1}{2}$ in $L\langle q, \omega, c\rangle$. When q is understood from the context, we write $L[c]$ for $L(q, c)$.

Note that $L\langle q, 0, c\rangle$ is a polynomial in the input size $\lg q$, whereas $L\langle q, 1, c\rangle$ is an exponential expression of $\lg q$. For $0 < \omega < 1$, the expression $L\langle q, \omega, c\rangle$ is in between a polynomial and an exponential function of $\lg q$ and is therefore called *subexponential*. The smaller the value of ω is, the closer the expression is to a polynomial expression. For the basic variant of the index calculus method and the linear and cubic sieve methods, $\omega = 1/2$. The number-field sieve (NFS) method (for $\mathbb{F}_p$) and the Coppersmith method (for $\mathbb{F}_{2^n}$) have running time $L\langle q, 1/3, c\rangle$ and are, therefore, asymptotically faster. Experimentation with both the NFS method and the Coppersmith variant has revealed that in practical situations these methods indeed perform better than the methods with $\omega = 1/2$, when q is sufficiently large.

4.2 Variants of the index calculus method for $\mathbb{F}_p$

We describe three variants of the index calculus method for prime fields $\mathbb{F}_p$. These are the basic method, the linear sieve method and the cubic sieve method. Each of these has a running time on the order of $L[c]$. For the basic method c is quite large (between 1.5 and 2), so that we cannot use this method unless p is small (say, less than 100 bits in length). The two sieve methods just mentioned are *practical* methods for primes p of size $\leqslant 250$ bits. For the linear sieve method $c = 1$, whereas for the cubic sieve method $c = \sqrt{2\alpha} < 1$ for some $1/3 \leqslant \alpha < 1/2$. The cubic sieve method is thus asymptotically faster than the linear sieve method. Other popular variants of the index-calculus method for $\mathbb{F}_p$, that we do not discuss in this note include the Gaussian integer method (running time $L[1]$), the residue list sieve method (running time $L[1]$) and the generalized number field sieve method (running time $L\langle p, 1/3, c\rangle$).

4.2.1 The basic method

For this method, the factor base is $B = \{q_1, q_2, \ldots, q_t\}$, where q_i is the ith prime ($q_1 = 2$, $q_2 = 3$ and so on). With a harmless abuse of convention, we call an integer n to be B-smooth, if all the prime factors of n are in B, that is, if n factorizes completely over the factor base B. The first stage of the method computes the discrete logarithms of the elements of B (with respect to g). In order to do that, one raises g to a random power j, $1 \leqslant j \leqslant p-2$ and checks if g^j factorizes completely over the factor base B as an integer. Thus, if $g^j = \prod_{i=1}^{t} q_i^{\alpha_i}$, then

$$j \equiv \sum_{i=1}^{t} \alpha_i d_i \pmod{p-1}$$

where $d_i = \operatorname{ind}_g q_i$. This gives us a linear congruence in the unknowns d_i. After t such linearly independent relations are found, the resulting system is solved modulo $p-1$. Every search step for a relation involves the following two computations:

1. Computation of the discrete exponentiation g^j.
2. A check if g^j factorizes completely over B. This smoothness test is carried out by trial divisions by the elements of B.

The second stage of the basic method proceeds analogously. Once again we choose a random j, $1 \leqslant j \leqslant p-2$ and compute ag^j and try to factorize ag^j as an integer over the factor base B. Once we get a factorization $ag^j = \prod_{i=1}^{t} q_i^{\beta_i}$, we compute

$$\operatorname{ind}_g a \equiv -j + \sum_{i=1}^{t} \beta_i d_i \pmod{p-1}$$

from the values of d_i computed in the first stage.

4.2.2 The linear sieve method

Let $H = \lfloor \sqrt{p} \rfloor + 1$ and $J = H^2 - p$. Then $J \leqslant 2\sqrt{p}$. Let us consider the congruence

$$(H + c_1)(H + c_2) \equiv J + (c_1 + c_2)H + c_1 c_2 \pmod{p}$$

For *small* integers c_1, c_2, the right side of the above congruence, henceforth denoted as

$$T(c_1, c_2) = J + (c_1 + c_2)H + c_1 c_2$$

is on the order of $\sqrt{p}$. If the integer $T(c_1, c_2)$ is smooth with respect to the first t primes $q_1, q_2, \ldots, q_t$, that is, if we have a factorization like $J + (c_1 + c_2)H + c_1c_2 = \prod_{i=1}^{t} q_i^{\alpha_i}$, then we have a relation

$$\operatorname{ind}_g(H + c_1) + \operatorname{ind}_g(H + c_2) = \sum_{i=1}^{t} \alpha_i \operatorname{ind}_g(q_i) \tag{1}$$

For the linear sieve method, the factor base comprises the primes $q_1, q_2, \ldots, q_t$ and the integers $H + c$ for $-M \leqslant c \leqslant M$, where the parameters t and M are suitably chosen. We check the factorization of $T(c_1, c_2)$ for all values of c_1 and c_2 satisfying $-M \leqslant c_1 \leqslant c_2 \leqslant M$, and collect all the relations available. We assume that the generator g is a small prime that itself is in the factor base, say $g = q_k$. Then we have the relation $\operatorname{ind}_g q_k \equiv 1 \pmod{p-1}$. The resulting system of linear congruences is then solved modulo $p - 1$.

Note that for the basic method the integers (namely g^j) that we check for smoothness are on an average as large as $p/2$, whereas for the linear sieve method, the integers (namely $T(c_1, c_2)$) checked for smoothness are $O(\sqrt{p})$. Thus we expect to have a larger density of smooth candidates among $T(c_1, c_2)$ than among g^j. This is why the linear sieve runs faster than the basic method. For the cubic sieve method we are going to describe next, we check the smoothness of integers $R(A, B, C)$ (defined later) which are $O(p^\alpha)$ for some $1/3 \leqslant \alpha < 1/2$. Thus the integers $R(A, B, C)$ are on an average smaller than the integers $T(c_1, c_2)$. This means that for the cubic sieve method we expect to have even higher a density of smooth integers. Once again this phenomenon is reflected in the (faster) running time of the cubic sieve method.

In addition to the fact that the sieve methods check smaller integers for smoothness (compared to the basic method), they considerably speed up the smoothness tests by a process called *sieving*. We now make a somewhat detailed description of the sieving process for the linear sieve method. To start with one fixes a c_1, $-M \leqslant c_1 \leqslant M$, and initializes to zero an array $\mathfrak{A}$ indexed $c_1, \ldots, M$. One then computes for each prime power q_k^h (q_k is a small prime in the factor base and h is a small positive exponent), a solution for c_2 of the congruence $(H + c_1)c_2 + (J + c_1H) \equiv 0 \pmod{q_k^h}$. To avoid unnecessary details let us assume that $\gcd(H + c_1, q_k) = 1$, i.e. $H + c_1$ is not a multiple of q_k. Then the solution is given by $d \equiv -(J + c_1H)(H + c_1)^{-1} \pmod{q_k^h}$. The inverse in the last equation can be calculated by running the extended gcd algorithm on $H + c_1$ and q_k^h. Then for each value of c_2 ($c_1 \leqslant c_2 \leqslant M$) that is congruent to $d \bmod q_k^h$, $\lg q_k$ is added[6] to the array location $\mathfrak{A}_{c_2}$.

Once the above procedure is carried out for each small prime q_k in the factor base and for each small exponent h,[7] we check for which values of c_2, the entry of $\mathfrak{A}$ at index c_2 is *sufficiently close* (see below) to the value $\lg(T(c_1, c_2))$. These are precisely the values of c_2 such that for the given c_1, the integer $T(c_1, c_2)$ factorizes smoothly over the small primes in the factor base.

The criterion for 'sufficient closeness' of the array element $\mathfrak{A}_{c_2}$ and $\lg(T(c_1, c_2))$ goes like this. If $T(c_1, c_2)$ factorizes smoothly over the small primes in the factor base, then it should differ from $\mathfrak{A}_{c_2}$ by a small positive or negative value. On the other hand, if the former is not smooth, it would have a factor at least as small as q_{t+1}, and hence the difference between $\lg(T(c_1, c_2))$ and $\mathfrak{A}_{c_2}$ would not be too less than $\lg q_{t+1}$. In other words, this means that the values of the difference $\lg(T(c_1, c_2)) - \mathfrak{A}_{c_2}$ for smooth values of $T(c_1, c_2)$ are well-separated from those for non-smooth values and one might choose for the criterion a check whether the absolute value of the above difference is less than 1.

It is interesting to have an intuitive feeling why sieving speeds up the smoothness tests. Recall that for the basic method, the smoothness test is carried out by trial divisions of g^j by q_k. This is a costly process. For the linear sieve method, on the other hand, one replaces (multi-precision) division operations by (single-precision) additions (of $\lg q_k$). Of course, one has to compute $-(J + c_1H)(H + c_1)^{-1}$ modulo q_k^h (this is a costly operation), but that is done for each c_1 only (and not for each (c_1, c_2) pair). This is precisely the reason why sieving methods are very popular for solving the DLP (and many other similar problems).

[6] More precisely, some approximate value of $\lg q_k$, say, for example, the integer $\lfloor 1000 \lg q_k \rfloor$.

[7] The exponent h can be chosen in the sequence $1, 2, 3, \ldots$ until one finds an h for which none of the integers between c_1 and M is congruent to d.

This completes the description of the equation collecting phase of the first stage of the linear sieve method. This is followed by the solution of the linear system modulo $p-1$. The description of the second stage is left out here.

4.2.3 The cubic sieve method

Let us assume that we know a solution of the Diophantine equation

$$\begin{aligned} X^3 &\equiv Y^2 Z \pmod{p} \\ X^3 &\neq Y^2 Z \end{aligned} \tag{2}$$

with X, Y, Z of the order of p^α for some $\frac{1}{3} \leqslant \alpha < \frac{1}{2}$. Then we have the congruence

$$(X+AY)(X+BY)(X+CY) \equiv Y^2 \Big[Z + (AB+AC+BC)X + (ABC)Y \Big] \pmod{p}$$

for all triples (A, B, C) with $A+B+C=0$. If the bracketed expression on the right side of the above congruence, namely,

$$R(A,B,C) = \Big[Z + (AB+AC+BC)X + (ABC)Y \Big]$$

is smooth with respect to the first t primes $q_1, q_2, \ldots, q_t$, that is, if we have a factorization $R(A,B,C) = \prod_{i=1}^{t} q_i^{\beta_i}$, then we have a relation like

$$\mathrm{ind}_g(X+AY) + \mathrm{ind}_g(X+BY) + \mathrm{ind}_g(X+CY) \equiv \mathrm{ind}_g(Y^2) + \sum_{i=1}^{t} \beta_i \, \mathrm{ind}_g(q_i) \pmod{p-1} \tag{3}$$

If A, B, C are *small* integers, then $R(A,B,C)$ is of the order of p^α, since each of X, Y and Z is of the same order. This means that we are now checking integers smaller than $O(p^{\frac{1}{2}})$ for smoothness over first t primes. As a result, we expect to get relations like (3) more *easily* than relations like (1) as in the linear sieve method.

This observation leads to the formulation of the cubic sieve method as follows. The factor base comprises primes $q_1, q_2, \ldots, q_t$, the integer Y^2 and the integers $X+AY$ for $0 \leqslant |A| \leqslant M$, for some suitably chosen values of t and M. In order to check the smoothness of $R(A,B,C)$ sieving techniques are employed. One fixes C and varies B. Since $A+B+C=0$, the value of A is fixed, once those for B and C are fixed. The sieving process is somewhat more involved than that in case of the linear sieve method. We leave out the details here.

Attractive as it looks, the cubic sieve method has several drawbacks which impair its usability in practical situations.

1. It is currently not known how to solve the congruence (2) for a general p. And even when it is solvable, how large can α be? For practical purposes α should be as close to $1/3$ as possible.

2. The expression $R(A,B,C)$ involves quadratic and cubic expressions in A, B and C. For practical sizes of p this makes $R(A,B,C)$ close to $p^{1/2}$, even when α is close to $1/3$. Recall that the expression for $T(c_1, c_2)$ involves only the *linear* factor $(c_1 + c_2)$.

3. The second stage of the cubic sieve method is asymptotically as slow as the equation collection stage. For the linear sieve method, on the other hand, individual logarithms can be computed much faster than the equation collecting phase.

For some special class of primes a solution of the congruence (2) is naturally available. For example, when p is slightly smaller than a whole cube, we can apply the cubic sieve method with $X = \lceil \sqrt[3]{p} \rceil$, $Y = 1$ and $Z = X^3 - p$.

4.3 Variants of the index calculus method for $\mathbb{F}_{2^n}$

For the fields of even characteristic, many variants of the index calculus methods are in use. Here we describe the basic method and the Coppersmith method. The running time of the basic method is $L\langle 2^n, 1/2, c\rangle$, whereas that for the Coppersmith method is $L\langle 2^n, 1/3, c'\rangle$. The second method is, therefore, asymptotically faster than the other.

Recall that elements of $\mathbb{F}_{2^n}$ are represented as polynomials over $\mathbb{F}_2$ and of degree $< n$. Arithmetic in $\mathbb{F}_{2^n}$ is carried out as polynomial arithmetic over $\mathbb{F}_2$ modulo a predetermined irreducible polynomial $f(x)$ of degree n. The term 'polynomial' without reference to a field will imply, at least in this section, a polynomial over $\mathbb{F}_2$. When no ambiguities are likely, we will denote a polynomial $a(x)$ by a. We assume that g is a primitive element of $\mathbb{F}_{2^n}^*$ and are supposed to compute the discrete logarithm of $a \in \mathbb{F}_{2^n}^*$ with respect to g, that is, $\operatorname{ind}_g a$. For brevity we refer 2^n as q.

Before we proceed further, let us mention the following result. A polynomial $f(x) \in \mathbb{F}_2[x]$ of degree n has all of its irreducible factors with degree $\leqslant m (< n)$ with asymptotic probability $p(n, m) = \exp\left((1 + o(1))\frac{n}{m} \ln \frac{n}{m}\right)$. This relation holds for $n \to \infty$, $m \to \infty$ and $n^{1/100} \leqslant m \leqslant n^{99/100}$ and provides a good approximation for *finite* values of n and m. Note that the probability expression implies that the smaller the degree of $f(x)$ (i.e. n) is, the higher is the chance that all the irreducible factors of $f(x)$ have degree $\leqslant m$.

4.3.1 The basic method

The basic index calculus method for $\mathbb{F}_{2^n}$ is very similar to that for $\mathbb{F}_p$. We choose an $m < n$ and let the factor base B consist of all irreducible polynomials (over $\mathbb{F}_2$) of degree $\leqslant m$. Let $B = \{h_1, h_2, \dots, h_t\}$. In the first stage, one chooses a random integer j, $1 \leqslant j \leqslant q - 2$ and checks if g^j factorizes completely over the factor base B as a polynomial. Thus, if $g^j = \prod_{i=1}^t h_i^{\alpha_i}$, then we get a linear congruence in the unknowns $d_i = \operatorname{ind}_g h_i$ as follows:

$$j \equiv \sum_{i=1}^{t} \alpha_i d_i \pmod{q-1}$$

After t linearly independent congruences are available, the resulting system is solved modulo $q - 1$. Each step in this search process involves the following two computations:

1. Computation of the discrete exponentiation g^j in $\mathbb{F}_{2^n}$.
2. A check if all irreducible factors of g^j have degree $\leqslant m$. This test is performed by trial divisions of g^j by the irreducible polynomials h_i in the factor base.

In the second stage, we choose random integers j, $1 \leqslant j \leqslant q - 2$ and try to factorize ag^j as $ag^j = \prod_{i=1}^t h_i^{\beta_i}$. Once we get one such factorization, the index $\operatorname{ind}_g a$ is computed as

$$\operatorname{ind}_g a \equiv -j + \sum_{i=1}^{t} \beta_i d_i \pmod{q-1}$$

from the values of d_i available from the first stage.

A suitable choice of m gives a running time of $L[c]$ with $c = 1.178$ for the basic index calculus method over $\mathbb{F}_{2^n}$. Numerous modifications of this basic variant have been proposed. All these modifications are based on the fact that the lower the degree of a polynomial is, the larger is the chance that the polynomial has irreducible factors each of degree $\leqslant m$. In the basic method we check the factorization of polynomials g^j the degrees of which are close to n in general. A modification of the basic method due to Blake et. al. checks the factorization of two polynomials g_1 and g_2 of degrees close to $n/2$ for a given polynomial $g^j \equiv g_1/g_2 \pmod{f}$. This leads to a faster variant of the basic method, but still leads to a running time of $L[1.178]$. Two other variants, similar to the linear and cubic sieve methods for $\mathbb{F}_p$, test (per iteration of the search process) the factorization of a *single* polynomial of degree $\approx n/2$ or $n/3$ and reduce the running time to $L[0.833]$ and $L[0.680]$ respectively. We do not describe these variants in this note. Instead we describe another variant proposed by Coppersmith, that reduces the search for a relation to the question of complete factorizability (over B) of two polynomials each of degree $\tilde{O}(n^{2/3})$. This trick reduces the exponent ω from $1/2$ to $1/3$ in the running time $L\langle q, \omega, c\rangle$.

4.3.2 The Coppersmith method

Let us assume that the defining polynomial $f(x)$ is of the special form $f(x) = x^n + f_1(x)$ with the degree of $f_1(x)$ of the order of $\lg n$. The factor base B consists of irreducible polynomials of degree $\leqslant m$, where $m = O\left(n^{1/3}(\ln n)^{2/3}\right)$. It is also assumed that $g \in B$. We choose a $k \in \mathbb{N}$ such that $2^k = O\left((n/\ln n)^{1/3}\right)$ and define $h = \lfloor n/2^k \rfloor + 1$. Now we choose two polynomials $u_1(x)$ and $u_2(x)$ of degrees $\leqslant D$, for some $D = O\left(n^{1/3}(\ln n)^{2/3}\right)$. We should have $\gcd(u_1(x), u_2(x)) = 1$. Let

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

Then we have the congruence $\operatorname{ind}_g w_2 \equiv 2^k \operatorname{ind}_g w_1 \pmod{q-1}$. Now we try to factorize the polynomials $w_1(x)$ and $w_2(x)$ over the factor base $B = \{h_1, h_2, \ldots, h_t\}$. If $w_1(x) = \prod_{i=1}^t h_i^{\alpha_i}$ and $w_2(x) = \prod_{i=1}^t h_i^{\beta_i}$, then we get a relation

$$
\sum_{i=1}^{t} \alpha_i d_i \equiv 2^k \left(\sum_{i=1}^{t} \beta_i d_i \right) \pmod{q-1}, \tag{4}
$$

where $d_i = \operatorname{ind}_g h_i$. We see that $\deg w_1 \leqslant \max(\deg u_1 + h, \deg u_2) = \tilde{O}(n^{2/3})$ by choice of h and D. On the other hand, $w_2(x) \equiv u_1(x^{2^k})x^{h2^k} + u_2(x^{2^k}) \equiv u_1(x^{2^k})x^{h2^k - n} f_1(x) + u_2(x^{2^k}) \pmod{f(x)}$, so that $\deg w_2 \leqslant \max\left(2^k \deg u_1 + h2^k - n + \deg f_1, 2^k \deg u_2\right) = \tilde{O}(n^{2/3})$ due to the choices of f_1, k, h and D. Thus finding a relation like (4) is equivalent to finding two polynomials (w_1 and w_2) each factorizable over B and of degree $\tilde{O}(n^{2/3})$. By varying the polynomials u_1 and u_2, sufficient number of relations are generated, and the resulting system (which includes the congruence $\operatorname{ind}_g g \equiv 1 \pmod{q-1}$) is solved to determine $d_i = \operatorname{ind}_g h_i$. As in the other variants of the index calculus method, the second stage computes $\operatorname{ind}_g a$ for any $a \in \mathbb{F}_{2^n}$. We do not elaborate this stage in this note.

The running time of the Coppersmith method for a suitable choice of k and D is $L\langle q, 1/3, 1.099\rangle$. Note that the conventional sieving procedure as we described in connection with the linear and cubic sieve methods for $\mathbb{F}_p$, can not be directly applied to the Coppersmith method. Gordon and McCurley introduced the concept of a *polynomial sieving* procedure that can be applied to the Coppersmith method. They also report an implementation of the Coppersmith method together with their sieving procedure, which enabled them to compute discrete logarithms in fields as large as $\mathbb{F}_{2^{503}}$.

4.4 Solving large linear systems of congruences

The first stage of the index calculus method generates a large number of linear congruences modulo $q-1$ ($q = p$ or $q = 2^n$) and solves the system modulo $q-1$. In a typical application, the number of unknowns (say, t) in the system may be as high as 10^6. Solving such a large system using the conventional Gaussian elimination procedure demands a huge amount of computing resources, namely $O(t^2)$ space and $O(t^3)$ time. It is not possible to solve such huge systems using the Gaussian elimination method in *feasible* time.

Luckily enough all the congruences generated by the first stage of the index calculus method are necessarily *sparse*. That is, only a very few coefficients in each equation are non-zero. This sparsity of the linear system can be effectively exploited and the system can be solved in time $O\tilde{}(t^2)$ using $O\tilde{}(t)$ memory space. Some special-purpose algorithms must be employed for this purpose. The algorithms that are commonly used are:

1. structured Gaussian elimination method
2. Wiedemann's algorithm
3. conjugate gradient method
4. Lanczos algorithm

Another difficulty with the system of congruences modulo $q-1$ is that the modulus $q-1$ is usually not a prime. If the complete factorization of $q-1$ is known, say $q-1 = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_r^{\alpha_r}$, then for each i, the system is solved modulo p_i, the solution is then *lifted* to a solution modulo $p_i^{\alpha_i}$ and finally the solutions modulo $p_i^{\alpha_i}$ for $i = 1, 2, \ldots, r$ are combined to find a solution modulo $q-1$. If, on the other hand, the factorization of the integer $q-1$ is not known, one may proceed as if $q-1$ is a prime (so that the set of integers modulo $q-1$ behaves like a field) and use linear algebra techniques as applicable to a field. This method works quite well in practice, but may incur occasional difficulties.

4.5 Parallelization issues

The equation collecting phase in the first stage of the index calculus method can be massively parallelized. For example, in the basic method, different processors can generate different sets of g^j and obtain relations independently. Similarly for the linear (or cubic) sieve methods one can run the sieving procedures for different sets of values of c_1 (or C) on different processors. Finally for the Coppersmith variant, the task of generating relations from the pairs of polynomials (u_1, u_2) can be effectively distributed among several processors.

The procedures for solving systems of linear congruences, on the other hand, are essentially sequential and have to be run either on a single processor or on a closely coupled network of processors that can communicate with each other very fast. As a result this part of the index calculus method often turns out to be the bottleneck of the entire procedure.

Finally the second stage of the index calculus method (which is usually much faster than the first stage) can be parallelized effectively similar to the equation collecting phase of the first stage.

5 Applications to cryptography

As is evident from the discussion of the earlier sections, the procedures for solving the DLP over a finite field $\mathbb{F}_q$ are computationally very expensive. If q is sufficiently large (say $q \geqslant 2^{768}$), then even the best-known variants of the index calculus method can not compute discrete logarithms in $\mathbb{F}_q$ in feasible time. This basic observation leads to the development of several cryptographic schemes that derive their security from the intractability of the DLP. In this section, we describe the Diffe-Hellman key agreement scheme and the ElGamal encryption and signature schemes.

5.1 The Diffe-Hellman key agreement procedure

Let us suppose that two parties A (Alice) and B (Bob) want to share a secret, for example, the secret key of a symmetric-key cipher (like DES) for future use. However, all communication channels between A and B are insecure in that an eavesdropper C (Carol) may intercept messages between A and B. Let us also assume that A and B can not communicate with each other personally.

In such a case A and B execute what is known as a *key exchange protocol* or a *key agreement protocol*. The Diffe-Hellman key agreement scheme is an example of such a protocol. In this protocol, A and B choose a particular finite field $\mathbb{F}_q$ and a compute a generator g of $\mathbb{F}_q^*$.[8] A then generates a random integer a, $2 \leqslant a \leqslant q-2$, computes the field element g^a and sends g^a to B. In a similar fashion, B generates a random integer b, $2 \leqslant b \leqslant q-2$ and sends g^b to A. Now A and B can both compute the field element $g^{ab} = (g^b)^a = (g^a)^b$ from the knowledge they have. This element g^{ab} can be used by A and B as a shared secret. The eavesdropper C can know q, g, g^a and g^b (by listening to the conversation between A and B). However, if q is chosen to be sufficiently large, C cannot compute g^{ab} in feasible time, because the DHP is known to be intractable. Of course, if the DLP is efficiently solved, the DHP is also efficiently solved. That is why[9] we loosely say that the Diffe-Hellman scheme is based on the intractability of the DLP. Note that this comment is equally applicable to the ElGamal encryption scheme that we describe below.

5.2 ElGamal encryption scheme

In an encryption scheme, an entity A encrypts a message m using the *encryption key* e and sends the encrypted message to another entity B. B possesses the knowledge of the corresponding *decryption key* d with which he decrypts the encrypted message to recover the original message m. An eavesdropper C without knowledge of d can not perform the decryption operation in feasible time. In a *symmetric-key* system (like DES, IDEA, BlowFish etc.) the same key is used for encryption and decryption, that is, $e = d$. In an *asymmetric-key* (also called *public-key*) system, e is made public and is called the *public key*, whereas d is kept secret and is called the *private key*. Such public-key systems are based on the assumption that the computation of d from the knowledge of e is a computationally intractable problem. The DLP is one such problem and allows us to build cryptosystems with $e = g^d$ in some field $\mathbb{F}_q$. (The private key d is the discrete logarithm of the public-key e.) The ElGamal scheme is based on this observation and can be described as follows.

1. Suppose that A wants to send a (secret) message m to B.
2. B chooses a finite field $\mathbb{F}_q$ and a generator g of $\mathbb{F}_q^*$.[10] He then chooses a random private key d, $2 \leqslant d \leqslant q-2$, computes $e = g^d$ and makes e (also q, g and, if necessary, the representation of $\mathbb{F}_q$) public. d is kept secret.
3. A gets B's public key e, generates a random integer k, $2 \leqslant k \leqslant q-2$ and computes the following quantities: $r = g^k$ and $s = mg^{kd} = me^k$. She then sends the pair (r, s) to B.
4. B recovers m as $m = sg^{-kd} = s(g^k)^{-d} = sr^{-d} = sr^{q-1-d}$.

[8] It is not necessary that g be a generator of $\mathbb{F}_q^*$. In general, it is computationally expensive to find generators of $\mathbb{F}_q^*$. What is suggested instead is to choose a finite field $\mathbb{F}_q$ such that a large prime factor r of $q-1$ is *known a priori*. It is easy to find out a random element $g \in \mathbb{F}_q^*$ whose multiplicative order is r.

[9] Not exactly! We told earlier that the DLP is actually believed to be equivalent to the DHP. That is, if the DHP is efficiently solvable, so also is the DLP. However, a complete proof of this assertion is not settled yet.

[10] Once again, g need not be a primitive element of $\mathbb{F}_q^*$. Any element of large multiplicative order will do.

Note that C knows $r = g^k$ and $e = g^d$. She can decrypt the message, if she can compute g^{kd}. So her problem is once again the DHP (and not the DLP). The believed equivalence of the DHP and DLP lets us say that the ElGamal scheme derives its security from the intractability of the DLP.

5.3 The ElGamal signature scheme

In a digital signature scheme an entity A can create signed messages that anybody can verify. A uses her private-key d to generate a digitally signed message, whereas anybody with the knowledge of A's public key e can check if the message is actually signed by A. An adversary who tries to forge A's signature must have a knowledge of d. The signature scheme should be designed such that knowledge of e does not allow the adversary to compute d in feasible time. Once again we can exploit the intractability of the DLP to build a signature scheme (with $e = g^d$) as follows. For simplicity we assume that the underlying finite field is a prime field $\mathbb{F}_p$. The method can be easily generalized to arbitrary fields $\mathbb{F}_q$ (in fact, to arbitrary finite Abelian groups).

1. Suppose that A wants to sign a message m.
2. A generates a (suitably large) finite field $\mathbb{F}_p$ and a generator[11] g of $\mathbb{F}_p^*$. She then selects a random d, $2 \leqslant d \leqslant p-2$, computes $e = g^d$, makes p, g and e public and keeps d secret.
3. A then selects a random secret integer k, $2 \leqslant k \leqslant p-2$ with $\gcd(k, p-1) = 1$ and computes $r = g^k$ (in $\mathbb{F}_p$) and $s \equiv k^{-1}(H(m) - dr) \pmod{p-1}$, where H is a cryptographic hash function.[12] Her signature on the message m is the pair (r, s).
4. To verify A's signature B does the following. B gets A's public-key e, computes $v_1 = e^r r^s$ and $v_2 = g^{H(m)}$ in $\mathbb{F}_p$ and accepts the signature if $v_1 = v_2$.

Note that if C can compute discrete logarithms in $\mathbb{F}_p$ efficiently, she can compute d from e, generate a random k and produce the signature (r, s) on any message m. In this case, the security of the system is thus based directly on the DLP (not on the DHP).

6 Notes and references

This article is intended to give the reader an overview of the discrete logarithm problem and its application to cryptography. The main results have been highlighted often without any proof or sufficient motivation. In this section, we provide further references in this area, that interested readers and researchers would find worth investigating.

Background material

For elementary number theory one can consult the book by Montgomery, Niven and Zuckerman ('An introduction to the theory of numbers', John Wiley & Sons, 5th. edition, 1991). In order to use more sophisticated tools like the number field sieves, analytic estimates of various parameters, etc., some knowledge of analytic number theory and algebraic number theory would be useful. Apostol's book ('Introduction to analytic number theory', Undergraduate text in Mathematics, Springer Verlag, 1976) is a good starting point for learning analytic number theory. For algebraic number theory, one may start with the book by Ireland

[11] More precisely, an element of high multiplicative order

[12] See text books on cryptography to know how hash functions are defined and used in cryptography.

and Rosen ('A classical introduction to modern number theory', Graduate Text in Mathematics #84, 2nd Edition, Springer-Verlag, 1990). Herstein's book ('Topics in Algebra', 2nd. Edition, John Wiley & Sons, 1975) covers discrete structures (groups, rings, fields, polynomials and linear algebra) in good details. For linear algebra, one may additionally refer the book by Hoffman and Kunze ('Linear algebra', 2nd. Edition, Prentice-Hall, 1971). Complexity theory to the extent needed for cryptography can be found in any text book on data structures and algorithms; see, for example, Cormen, Lieserson and Rivest's book ('Introduction to algorithms', MIT Press & McGraw Hill, 1990). Some recent theoretical results in the area of cryptography requires knowledge of some sophisticated complexity classes like NC, BPP etc. One may read Koblitz's book ('Algebraic aspects of cryptography', Springer Verlag, 1998) or Goldreich's internet book ('Foundations of Cryptography', Fragments of a book, downloadable from http://theory.lcs.mit.edu/~oded/frag.html) to know more about these complexity classes.

Anybody interested in finite fields should read the *bible* by Lidl and Niederreiter ('Finite Fields', Encyclopedia of Mathematics and its Applications, 20, Cambridge University Press, 1984) which provides an encyclopedic treatment of both theoretical and computational aspects of finite fields, known till early eighties. Some books that came later (and are, therefore, more up-to-date) include those by Small ('Arithmetic of finite fields', Marcel Dekker Inc., 1991), Shparlinski ('Computational problems in finite fields', Kluwer Academic Publishers, 1992) and Menezes ('Applications of finite fields', Kluwer Academic Publishers, 1993). Also look at the survey article by Shparlinski ('On some problems in the theory of finite fields', Russian Mathematical Surveys, 46:1 (1991), 199–240).

Representation of finite fields in computers and related issues are covered in the book by Mignotte ('Mathematics for Computer Algebra', Springer Verlag, 1992), the book by Menezes mentioned above and in the paper by Win et. al. ('A fast software implementation for arithmetic operations in GF(2^n)', Advances in Cryptology – AsiaCrypt'96, LNCS #1163 (1996), Springer-Verlag, 65–76). For implementation of multiple-precision integer routines, one may consult the books by Knuth ('The Art of Computer Programming, Vol. 2: Seminumerical Algorithms', 3rd edition, Addison Wesley, 1997) and by Cohen ('A course in computational algebraic number theory', Graduate Text in Mathematics #138, Springer Verlag, 1993).

The discrete logarithm problem

The book by Menezes mentioned above is a good introduction to the discrete logarithm problem. Coppersmith, Odlyzko and Schroeppel's paper ('Discrete logarithms in $GF(p)$', Algorithmica, 1 (1986), 1–15) introduces the linear sieve, the cubic sieve and the Gaussian integer methods. A subsequent paper by LaMacchia and Odlyzko ('Computation of discrete logarithms in prime fields', Designs, Codes, and Cryptography, 1 (1991), 46–62) reports implementation of the linear sieve method and the Gaussian integer method. Also look at the survey article by McCurley ('The discrete logarithm problem', Cryptology and Computational Number Theory, Proceedings of the Symposium in Applied Mathematics, 42 (1990), 49–74) to know more about the discrete logarithm problem over prime fields $\mathbb{F}_p$. Odlyzko's survey article ('Discrete logarithms and their cryptographic significance', Advances in Cryptology: Proceedings of Eurocrypt '84, LNCS #209 (1985), Springer-Verlag, 224–314) describes the index calculus method as applicable to the fields $\mathbb{F}_{2^n}$ of characteristic 2. McCurley and Gordon implemented the Coppersmith method and computed discrete logarithms in $\mathbb{F}_{2^{401}}$ ('Massively Parallel Computation of Discrete Logarithms', Proc. Crypto '92, Lecture Notes in Computer Science, 740 (1992), Springer-Verlag, 312–323). They also completed the equation collection phase for the field $\mathbb{F}_{2^{503}}$.

The more recent advances in the area of DLP, namely the number field sieve method and its implementation, can be found in the papers by Schirokauer ('Discrete logarithms and local units', Philosophical transactions of the Royal Society of London, Series A, 345 (1993), 409–423), Schirokauer, Weber and Denny ('Discrete logarithms: the effectiveness of the index calculus method', Proc. ANTS II, LNCS #1122 (1996), Springer-Verlag), Weber ('Computing discrete logarithms with the general number field sieve', Proc. ANTS II, LNCS #1122 (1996), Springer-Verlag) and Weber and Denny ('The solution of McCurley's discrete log challenge', Advances in Cryptography – Crypto'98, LNCS #1462 (1998), Springer-Verlag, 458–471).

See the paper by LaMacchia and Odlyzko ('Solving large sparse linear systems over finite fields', Advances in Cryptology – CRYPTO '90, LNCS #537 (1991), Springer Verlag, 109–133) to know about the algorithms for solving large sparse system of congruences modulo $q - 1$ (where $\mathbb{F}_q$ is a finite field). Other interesting articles include the survey by Lenstra and Lenstra ('Algorithms in number theory', in Handbook of Theoretical Computer Science, ed. J. van Leeuwen, 1990, 675–715) and the collection of lecture notes edited by Lenstra and Lenstra ('The development of the number field sieve', Lecture notes in mathematics 1554, Springer Verlag, 1993). The authors' Ph.D. thesis ('Galois Field computations: Implementation of a library and a study of the discrete logarithm problem', Computer Science and Automation, Indian Institute of Science, Bangalore, 1999) describes many details left out in this article. Also look at the paper by the author and Veni Madhavan ('Performance Comparison of linear sieve and cubic sieve algorithms for discrete logarithms over prime fields', Proc. 10th International Symposium on Algorithms and Computation (ISAAC'99), in LNCS #1741 (1999), 295–306) for some results on the cubic sieve method.

The integer factorization problem

The integer factorization problem (IFP) is another computationally intractable problem which has been used extensively to build cryptosystems like the ones in the RSA family. Loosely speaking, the IFP problem can be stated like this: Given the product n of two prime numbers p and q, find out the primes. It is widely believed that the IFP is computationally equivalent to the DLP. However a complete proof for this alleged equivalence is not yet available. There are some partial results in favor of the equivalence; see, for example, Section 6.9 of Menezes' book ('Applications of finite fields') discussed earlier. Indeed, save a few exceptions, most algorithms we use nowadays for solving the DLP are direct adaptations of the algorithms for solving the IFP. Any new algorithm to solve one of these problems is expected to apply to the other problem as well. It is, therefore, important to be aware of the algorithms for solving the IFP, if one is interested in the DLP (and vice versa). Indeed the IFP has been studied at a greater depth than the DLP. Some methods to solve the IFP are the continued fraction method, the quadratic sieve method, the elliptic curve method and the number field sieve method. Numerous articles on the IFP have been published. Here is a short list (chosen according to the personal preference of the author): the book by Bressoud ('Factorization and Primality Testing', Undergraduate Text in Mathematics, Springer Verlag, 1989), Cohen's book ('A course in computational algebraic number theory' discussed earlier), the survey article by Lenstra and Lenstra in the Handbook of Theoretical Computer Science (also mentioned above) and the article by Odlyzko ('The future of integer factorization', CryptoBytes, RSA Laboratories, 1 (no. 2) (1995), pp. 5–12).

Cryptography

Cryptography (more broadly, cryptology) has a vast literature of which we mention here only a small part that interested readers and software developers would find handy to start with. The 'Handbook of Applied Cryptography' by Menezes, Oorschot and Vanstone (CRC Press, 1997) deals with a huge coverage on many basic topics in cryptography and provides a big list of references. Schneier's book ('Applied Cryptography', John Wiley & Sons, 1996) is another book of similar nature (and bulk). One may also look at the books by Koblitz ('A course in number theory and cryptography', 2nd edition, Springer Verlag, 1994, and 'Algebraic aspects of cryptography', Springer Verlag, 1998). Goldreich's book ('Foundations of Cryptography') mentioned earlier in this section talks about mathematical foundations regarding security of a cryptographic system.

Public-key cryptography was born from the seminal discovery of key exchange schemes by Diffe and Hellman ('New directions in cryptography', IEEE transactions on Information Theory, 22 (1976), 644–654) and from the discovery of the RSA algorithms by Rivest, Shamir and Adleman ('A method for obtaining digital signatures and public-key cryptosystems', Communications of the ACM, 21 (1978), 120–126). ElGamal ('A public-key cryptosystem and a signature scheme based on discrete logarithms', IEEE transactions on Information Theory, 31 (1985), 469–472) applied the intractability of the DLP to design encryption and signature schemes over finite fields.

'RSA laboratories' has released public-key cryptography standards (PKCS) that standardize the public-key encryption, signatures and key-exchange techniques. Another useful article on standards is the IEEE P1363 draft available from IEEE. NIST (National Institute of Standards and technology, U.S. Department of Commerce) and ITU-T (International Telecommunications Union, Telecommunications Standards Sector) published many documents to standardize various cryptographic schemes and protocols. Internet RFC's (Request for comment) and drafts are also good source of knowledge on many cryptographic algorithms.

Elliptic curves and cryptography

Recently elliptic curves have been employed for designing various public-key cryptographic schemes, because computation of discrete logarithms in elliptic curves over finite fields seems to be a very difficult problem (even harder than the finite field discrete logarithm problem). A direct adaptation of the index calculus method for computing elliptic curve discrete logarithms is expected to lead to a running time *worse* than that of brute-force search. Logarithms in a singular elliptic curve defined over $\mathbb{F}_q$ with a cusp can be computed in polynomial time. The discrete logarithm problem for a general elliptic curve over $\mathbb{F}_q$ can be reduced to the discrete logarithm problem in the field $\mathbb{F}_{q^k}$ for a suitable k. However, this k is quite large in general and the reduction takes time exponential in $\log q$. For supersingular elliptic curves, this reduction can be done in probabilistic polynomial time. For non-supersingular elliptic curves the best methods known till date are the square root methods which run in time exponential in the size of the group. Recently, Joseph H. Silverman has proposed a new method, called the *xedni calculus method* for computing discrete logarithms in elliptic curves. However, this method has been experimentally and heuristically shown to be impractical.

Miller ('Uses of elliptic curves in cryptography', Advances in cryptology, CRYPTO'85, LNCS #18 (1986), Springer Verlag, 417–426) and Koblitz ('Elliptic curve cryptosystems', Mathematics of Computation, 48 (1987), 203–209) first pointed out that the intractability of the elliptic curve discrete logarithm problem (ECDLP) can be effectively applied to build secure cryptographic systems. The book by Menezes ('Elliptic curve public key cryptosystems', Kluwer Academic Publishers, 1993) provides a nice survey on the ECDLP and its application to cryptography. Also see the book by Koblitz ('A course in number theory and cryptography', 2nd edition, Springer Verlag, 1994) and the Certicom white papers (downloadable from http://www.certicom.com/). For the Xedni calculus method look at Silverman's paper ('The Xedni Calculus and the Elliptic Curve Discrete Logarithm Problem', Design, Codes, and Cryptography, to appear) and the technical report by Jacobson et. al. ('Analysis of the xedni calculus attack', Technical report, CORR 99-06, University of Waterloo, 1999).