This article will appear in *Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography* (Mathematical Sciences Research Institute Publications), J. P. Buhler and P. Stevenhagen (eds.). It is only for the personal use of the participants of the MAA MathFest Short Course (Summer 2002), and should not be reproduced.

ELEMENTARY THOUGHTS ON DISCRETE LOGARITHMS

Carl Pomerance

Given a cyclic group G with generator g, and given an element t in G, the discrete logarithm problem is that of computing an integer l with $g^l = t$. The problem of computing discrete logarithms is fundamental in computational algebra, and of great importance in cryptography. In this lecture we shall examine how sometimes the problem may be reduced to the computation of discrete logarithms in smaller groups (though this reduction may not always lead to an easier problem). We give an example of how the reduction may be used profitably in taking "square roots" in cyclic groups of even order. We shall look at several exponential-time algorithms that work in a quite general setting, and we shall discuss the index calculus algorithm for taking discrete logarithms in the multiplicative group of integers modulo a prime.

"The" cyclic group of order n

I should begin by saying that the discrete logarithm (dl) problem is not always hard. Obviously it is easy if the target element t is the group identity, or in general, some small power of g. Less obviously, there are entire families of cyclic groups for which the dl problem is easy. Take, for example, the additive group $G = \mathbf{Z}/n\mathbf{Z}$. If we use the generator $g = 1$, the problem of computing discrete logarithms is absolutely trivial. (Here, and in the sequel, we identify elements of $\mathbf{Z}/n\mathbf{Z}$ with their least non-negative residue. As we shall see at the end of the lecture in connection with the index calculus algorithm, the fact that in some groups we may naturally represent group elements as integers can be quite useful.) If we change to another generator, it is still trivial. In fact, if g is a generator, then it is coprime to n. Finding the multiplicative inverse of $g \pmod n$, via Euclid's algorithm, suffices for finding the discrete logarithm of 1, and so we quickly get everything else.

Let us take $n = 100$, $g = 11$, $t = 17$ for example. The multiplicative inverse of 11 modulo 100 is 91, so the discrete logarithm of 17 is 91×17, that is, 47.

Good. Now let us take another cyclic group of order 100, namely $(\mathbf{Z}/101\mathbf{Z})^*$, the multiplicative group of reduced residues modulo 101. Coincidentally, 11 is still a generator. Quick, what is an integer l with $11^l \equiv 17 \pmod{101}$? Well, maybe this is not so easy anymore, though of course this small example is doable.

Note, if you ever hear someone talk of "*the* cyclic group of order n," beware. They are likely not talking about anything computational. The way the cyclic group is presented to you makes all the difference.

Reductions

We now embark on a tour of some fairly clear ways to reduce a dl problem in a cyclic group to dl problems in various subgroups. Before embarkation though, I should discuss some ground rules. It is assumed that with the group G, we know how to perform the group operation, including the taking of inverses. Sometimes it may be difficult to see if two elements in a group are equal. This may happen if the group is presented as a quotient structure, or perhaps as a group of binary quadratic forms. We shall assume that the cost of telling if two elements are equal is of the same magnitude as performing a group operation. Finally, we shall assume that it is possible to assign symbols to the group elements so that they may be sorted.

For our first reduction, assume the order n of the cyclic group G may be nontrivially factored as $n = uv$, where $\gcd(u, v) = 1$. Then, we may reduce the problem of solving for a discrete logarithm in G to solving for discrete logarithms in the subgroups of G of order u and v. In particular, if $G = \langle g \rangle$, then $\langle g^u \rangle$ generates the subgroup of u-th powers in G, which has order v, and similarly $\langle g^v \rangle$ generates the subgroup of v-th powers, which has order u. Say we solve the for the discrete logs l_u, l_v where

$$(g^u)^{l_u} = t^u, \quad (g^v)^{l_v} = t^v.$$

(The powers g^u, g^v, t^u, t^v are easy to find via repeated squaring.[1]) Say we also find integers a, b with $au + bv = 1$, using Euclid of course. Then

$$t = t^{au+bv} = (t^u)^a (t^v)^b = g^{ul_u a} g^{vl_v b} = g^{aul_u + bvl_v},$$

so that the discrete logarithm of t is $aul_u + bvl_v$, and we are through.

The next reduction considers the case when the order of G is a prime power, say p^a, where $a > 1$. (The argument does not use the primality of p, but it may as well be assumed because of the first reduction.) We will see that a dl problem in this group can be reduced to a dl problems in the cyclic subgroup of G of order p. Say, as usual, we are trying to find l such that $g^l = t$. If l is the least nonnegative value that works, and we write l in the base p, we have

$$l = l_0 + l_1 p + \ldots + l_{a-1} p^{a-1},$$

with each l_j an integer in $[0, p-1]$. We shall sequentially find $l_0, l_1, \ldots$ as follows. First note that

$$t^{p^{a-1}} = (g^l)^{p^{a-1}} = g^{lp^{a-1}} = g^{l_0 p^{a-1}} = (g^{p^{a-1}})^{l_0},$$

that is, l_0 is the solution of a dl problem in the cyclic subgroup of p^{a-1}-powers generated by $g^{p^{a-1}}$. Suppose that $l_0, \ldots, l_{j-1}$ have been computed. Consider $t_j = tg^{-l_0 - l_1 p - \ldots - l_{j-1} p^{j-1}}$.

[1] See the lecture of S. Wagon.

We have that t_j is a p^j-power, so that $t_j^{p^{a-j-1}}$ is in the subgroup of p^{a-1}-powers. Solve the dl problem

$$t_j^{p^{a-j-1}} = (g^{p^{a-1}})^{l_j}$$

for l_j. This is the next base p digit of l that we are searching for.

As an illustration of these reductions, let's return to the example of $g = 11, t = 17$ in $(\mathbf{Z}/101\mathbf{Z})^*$. Since the order of the group is 100, it suffices to solve two dl problems each in groups of order 2 and 5. First, $17^{25} \equiv -1 \pmod{101}$, so in our subgroup of order 4, 17^{25} must have discrete logarithm 2, that is, $l_{25} = 2$. (We have solved now both dl problems in the group of order 2: the first is 0, the second 1, so $l_{25} = 0 + 1 \cdot 2 = 2$.) Next, we must find l_4 where $11^{l_4} \equiv 17^4 \equiv 95 \pmod{101}$, which is solving a dl problem in a cyclic group of order 25. This is reduced to two dl computations in a group of order 5. We begin by computing $17^{20} \equiv 95^5 \equiv 1 \pmod{101}$, so that l_4 is a multiple of 5. Also, $11^{20} \equiv 87 \pmod{101}$. We have to find a power of 87 that is congruent to 95 modulo 101, and we know the answer is $0, 1, 2, 3$, or 4. Evidently it is not 0 or 1, and checking 2, we find that it works. Thus $l_4 = 0 + 2 \cdot 5 = 10$. Now $(-6) \cdot 4 + 1 \cdot 25 = 1$, so the discrete logarithm of 17 is $(-6) \cdot 4 \cdot 10 + 1 \cdot 25 \cdot 2 = -190$. The least nonnegative discrete logarithm is 10.

So, if we have the group order in our possession (which is not always the case), and since solving interesting dl problems is usually harder in practice than factoring, we might first run a factorization algorithm on the order, and reduce the problem to simpler cases as above.

Well, at least they are smaller cases. As we shall see, working in a subgroup is not always simpler.

An application

But first, I will describe a nice application to the second dl reduction, namely reducing a dl problem in a cyclic group of order p^a to a dl problems in a cyclic group of order p. The application is to taking square roots in a cyclic group. Say G is a cyclic group of order n. If n is odd, then every element is a square, and square roots are simple: the square root of an element h is $h^{(n+1)/2}$. If n is even, exactly half of the elements of G are squares. There is a very simple test for squareness: h is a square if and only if $h^{n/2} = 1$, where I am writing the group identity as 1. Suppose $n/2$ is odd. Then again, it is very easy to find a square root. If h is a square, then $h^{(n/2+1)/2}$ is a square root. How might one find the other square root? This is easy if you can find a group element that is not a square. If g is such a group element, then $x = g^{n/2}$ has order 2 and is not 1. Thus, if y is a square root of h, then xy is the other square root.

This last idea works in general, even when n is divisible by a high power of 2. If g is a nonsquare in G, then $g^{n/2}$ is an element of order 2 and can be used as x in the above.

But how would one find even one square root of a square h if n is divisible by a power of 2 higher than the first power? We again will make use of a nonsquare g. Say $n = 2^u v$, where v is odd. Then the element g^v has order 2^u. The element h^v is in $\langle g^v \rangle$. Solve for the discrete log. As we have seen, this is very simple, since the order of the group is 2^u. Say $h^v = (g^v)^l$. Of necessity, since h is a square and g is a nonsquare, l must be even. Then a square root of h, as is easily checked, is $h^{(v+1)/2}(g^v)^{-l/2}$, and we are done.

This polynomial time algorithm has one small flaw, did you spot it? It is the production of a nonsquare g. Of course, if you are given a cyclic generator of G, then you may use this generator as a nonsquare. But what if you are not given this? For example, say $G = (\mathbf{Z}/p\mathbf{Z})^*$, where p is a large prime. It may be hard to find a generator (a primitive root), especially if we don't know the prime factorization of $p-1$. But surely, finding a nonsquare shouldn't be hard, since half of all elements in the group are nonsquares, and the test for one is simple. So, we have a random algorithm that will work very nicely. Choose elements from G at random, and test for nonsquareness. The expected number of trials is 2. This method begs the question of how one is supposed to choose elements from a group at random. This is not so hard for $(\mathbf{Z}/p\mathbf{Z})^*$, but is conceivably a problem in general. So, modulo this problem of finding *some* nonsquare, taking square roots is easy.

As you might notice, this idea generalizes to taking p-th roots for all primes p. Further, if p is small, the various dl problems that arise may all be handled quickly.

Baby steps, giant steps

The title of this section has nothing to do with Neil Armstrong, the first person to step on the moon. Rather, it has to do with an algorithm of Dan Shanks that works very well at taking discrete logarithms.

As before, take a group $G = \langle g \rangle$ of order n, with $t \in G$. Our task is to find an integer l with $0 \le l \le n-1$ and $g^l = t$. Suppose $m = \lceil \sqrt{n} \rceil$ and we write our discrete logarithm l in the base m. Then $l = l_0 + l_1 m$ where $0 \le l_0, l_1 \le m-1$. It is evident that to find l, it is sufficient to find l_0, l_1.

Prepare two lists: $1, g, \ldots, g^{m-1}$ and $t, tg^{-m}, \ldots, tg^{-(m-1)m}$. Since $g^{l_0} = tg^{-l_1 m}$, there must be a common element in the two lists. Moreover, any common element immediately allows us to find the discrete logarithm of t. We're done.

You might wonder how to find the element in common in the two lists. At first blush, you might think that there are m^2 possible comparisons to make with an element from the first list and an element from the second list. That is *one* way to do it, but there is a better way. We assume that we can label the group elements in such a way that they can be sorted. Sort the elements in the first list. Then sequentially run through the second list to check for membership in the first list. The sorting can be done in $O(m \log m)$ comparisons, and each membership check, via a binary search, can be done in $O(\log m)$ comparisons. (A binary search involves identifying the midpoint of the sorted list, deciding if the searched-for element is in the first half or the second half, and then iterating in the appropriate half.) So in total we do about $O(m \log m) = O(\sqrt{n} \log n)$ comparisons after the lists are computed.

It is clear the first list, $1, g, \ldots, g^{m-1}$, can be computed in m group operations. After these baby steps, the giant step g^{-m} can be computed in two more operations, and then we can sequentially get the terms of the giant step sequence $t, tg^{-m}, tg^{-2m}, \ldots$ with one group operation per step. Note that after each giant step is taken, the look-up can be done in the baby step list, and we may stop as soon as the match is found. Thus, on average, we might expect to traverse only about half of the giant steps before completion. Note too that if we wish to find the discrete logarithm of another group element t', the same baby step sequence may be reused.

It might seem that one needs the group order n to use baby steps, giant steps. However, all that is needed in the above is that $m \geq \sqrt{n}$. So start with a small choice for m, try the algorithm out, and if it fails try again with $2m$, etc. Eventually it will work, and when this happy event occurs, the total time spent is of the shape $O(\sqrt{n}\log n)$, even though we may still not know what n is.

Actually, Shanks invented the baby steps, giant steps method as a means of computing the order of an abelian group G, not necessarily cyclic. He was interested in particular in the class group of an imaginary quadratic number field. By other means he is able to get a rough estimate of the order of the group: Say the group order is in the interval $[x, x+y]$, where $y < x$. (In fact, he is able, using the Extended Riemann Hypothesis, to get $y < x^{1-c}$ for some positive c.) He then chooses a random element h_1 in the group, and via baby steps, giant steps, he finds an integer $n_1 \in [x, x+y]$ such that $h_1^{n_1} = 1$. By factoring n_1 into primes, it is then possible to compute the actual order m_1 of h_1, and if it is n_1, we are done; this must be the order of the group. If $m_1 < n_1$ there is more work to be done. Choose another random element h_2, use baby steps, giant steps to compute the order m_2 of the subgroup $\langle h_1, h_2\rangle$, and so on. When finally a subgroup order m_k is found in $[x, x+y]$, we have $n = m_k$ and we are done. In the case at hand of class groups, it is also possible to use the ERH to find a fairly small set of group elements known to generate the group, so that randomness is not needed at all.

ρ

The baby steps, giant steps method, while a rigorous method of "square-root complexity," suffers from a high memory load, also about the square root of the group order. In contrast, John Pollard's ρ method, which also runs in about the square root of the group order, has negligible space requirements. The down sides are that the ρ method requires the group order, and it does not (yet) have a rigorous analysis. As with many other algorithms which produce a readily checkable answer, the fact that the method of achieving the answer is heuristic is not a practical concern, only a mathematical one.

The ρ method is based heuristically on the birthday paradox. If you throw balls randomly into n urns, where each urn is equally likely to receive a ball even if it already has one, how many balls should you expect to throw before some urn has two balls in it? The answer is surprisingly small, it is of magnitude $\sqrt{n}$. In particular, if $c\sqrt{n}$ balls are thrown, the probability that some urn has at least two balls is about $1 - e^{-c^2/2}$. So, in a room of 23 people, it is better than even odds that two of them have the same birthday.

Suppose $G = \langle g\rangle$ has order n. If $x_1, x_2, \ldots, x_k$ is a random sequence of elements of G, we would expect to see some $x_i = x_j$ when k is of order $\sqrt{n}$. However, we do not wish to use a truly random sequence, even if we had the means of generating it. We specifically wish to use a pseudorandom sequence, in fact one where the next term x_{i+1} depends in a specific manner on the current term x_i. As you will see, this conscious choice of avoiding randomness is important. It is also what keeps us at present from rigorously analyzing the algorithm.

So, we would like to define a function $f : G \to G$ which is both easy to compute, and seemingly random. Say we have some straightforward method of labelling the elements of G with the integers $1, 2, \ldots, n$. [2] Thus, it makes sense to talk of elements in the first third of G, the second third, and the third third. Call these type I, type II, and type III elements, respectively.

Suppose we are trying to find the discrete logarithm of a group element t. For $x \in G$, let

$$f(x) = \begin{cases} tx, & \text{if } x \text{ is type I} \\ x^2, & \text{if } x \text{ is type II} \\ gx, & \text{if } x \text{ is type III.} \end{cases}$$

We shall take for our pseudorandom sequence, $g, f(g), f(f(g)), \ldots$. Let x_i be the i-th term of this sequence, $i = 0, 1, 2, \ldots$. We also wish to keep track of the different types of elements we see as we traverse the sequence. Consider the sequence (a_i, b_i) of pairs of residues modulo n with initial term $(0, 1)$ and the rule

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i), & \text{if } x_i \text{ is type I} \\ (2a_i, 2b_i), & \text{if } x_i \text{ is type II} \\ (a_i, b_i + 1), & \text{if } x_i \text{ is type III.} \end{cases}$$

Then, as is easily checked, $x_i = t^{a_i} g^{b_i}$. And if it is discovered that $x_i = x_j$, then $g^{b_i - b_j} = t^{a_j - a_i}$. If $a_j - a_i$ is coprime to n, the discrete logarithm of t is the inverse of $a_j - a_i$ modulo n multiplied by $b_i - b_j$.

The condition that $a_j - a_i$ is coprime to n is not a strenuous one. As we saw above, it is possible to reduce the dl problem to the case of prime group orders. So, it might be assumed that n is prime. And so the only way for the equation $x_i = x_j$ not to lead to the discrete logarithm of t is if the pair (a_i, b_i) is identical to the pair (a_j, b_j). In practice this event does not frequently occur. If it did occur, one could try for a new sequence where now one is searching for the discrete logarithm of gt, for example. Or, one could let the initial seed be g^r for some random choice of r.

What remains to be discussed is an efficient way of searching for a pair i, j with $x_i = x_j$. We certainly don't want to write down all of the terms and exhaustively check all pairs. Not only would this kill the square root running time, it would consume too much space. There is a very neat method for finding a repeat in the sequence, known as the Floyd cycle-finding algorithm. The idea is to compute the sequences $x_i, (a_i, b_i)$ twice, once at single speed, and once at double speed. That is, if you have $x_i, (a_i, b_i)$ and $x_{2i}, (a_{2i}, b_{2i})$, use the rules to compute $x_{i+1}, (a_{i+1}, b_{i+1})$, $x_{2i+1}, (a_{2i+1}, b_{2i+1})$, $x_{2i+2}, (a_{2i+2}, b_{2i+2})$, so that each stage you have at hand x_i, x_{2i}. Check only these pairs for equality.

At first glance you might say that a great deal of generality is lost if we insist that $j = 2i$ for our equation $x_i = x_j$. But here is where we use that our sequence is *not* random, but rather an f-orbit. Note that if $x_{i_0} = x_{j_0}$, where $i_0 < j_0$ is the first occurence of equality, then so too do we have $x_{i_0+1} = x_{j_0+1}, x_{i_0+2} = x_{j_0+2}$, etc. Thus, if $\mu = j_0 - i_0$, then $x_u = x_v$ whenever $u, v \geq i_0$ and $u \equiv v \pmod{\mu}$. That is, the sequence becomes purely periodic starting at the i_0-th term. Well then, if we take u as the first multiple of μ that is at least i_0, namely, $u = \mu\lceil i_0/\mu \rceil$, then $x_u = x_{2u}$. And notice too that u is not very

[2] For a very simple example, suppose $G = (\mathbf{Z}/p\mathbf{Z})^*$. As we have been doing, we may label the elements of G with the integers $1, 2, \ldots, p-1$.

much bigger than i_0. That is, it is hardly a restriction at all to search for an equality of the form $x_u = x_{2u}$. We have so transformed a potentially quadratic search into a linear one. And, there are negligible memory requirements, since one only needs the current candidate x_i, x_{2i} to find the next one.

The ρ method can be used as a factoring algorithm, also an idea of Pollard. To factor n, use the function $f(x) = x^2 + a \bmod n$, where a is not $0, -2$. Instead of checking for an equality $x_i = x_{2i}$, check for a nontrivial value of $\gcd(x_i - x_{2i}, n)$.

The ρ method gets its name from the suggestive shape of the letter ρ, which might be the diagram for a sequence with a non-periodic beginning that eventually becomes periodic.

λ

Pollard also suggested a version of the ρ method that lends itself fairly easily to many computers sharing the job of computing one discrete logarithm. The key part of the shape of the letter ρ where the actual success is found is the point where the round part intersects the straight part. Focussing then on the convergence of two streams, the key Greek letter is λ.

Suppose we have k computers each following its own random sequence in the group G. If the order of G is n, then when the length of the sequences is about $\sqrt{n}/k$, we will begin to expect that some term in one computer's sequence will have a match with some term in another computer's sequence. Of course, we will not want to make every possible comparison. So we introduce the idea of a "distinguished point" and we use a pseudorandom iteration that has the property that once there is a match between two streams, they stay identical from then on.

To be specific, suppose we use the same iteration as in the ρ method, but we have computer m initialize its pseudorandom sequence at g^{r_m}, where r_m is a random number. We also make use of a hash of group elements, an easily computable mapping of group elements to integers, which is 1:1. (This might be the same labelling as in the ρ method.) We say a group element is distinguished if its hash is divisible by 2^{20}, say. Then each computer goes merrily along down its sequence, but whenever it arrives at a distinguished point, which occurs about every millionth iterate, it reports the event to a central computer. The central computer then sorts the incoming hashes of distinguished points, looking for a match. When one occurs, the data involved, if actually representing some $x_i = x_j$, can then be used to compute the desired discrete logarithm. So on average, we only have to wait about a half-million iterations after the first coincidence to find our match. Note that it is possible for the match to occur between two reports of the same computer, namely for some reason, that computer had extraordinary luck with the ρ method. That's fine, the λ method will take it.

One can take other pseudorandom functions f. One way of choosing f is to have a small set of integers S, and pre-compute g^s for $s \in S$. Then, the function f would send x to some $g^s x$, which one depending on some property of the hash of x. An initial seed is t^r for some random exponent r. This version is sometimes referred to as the "kangaroo method" where the various values of g^s are considered as "hops."

There are numerous ideas for fine-tuning and speeding up both the ρ and λ methods.

For this I refer you to two of the most sophisticated practitioners, Robert Harley and Edlyn Teske. As far as I know, the record for these methods is a dl computation, by Harley in April, 2000, in a subgroup of an elliptic curve group, with the subgroup having prime order approximately equal to 2^{108}.

The index calculus and the search for smoothness

The methods we have described so far have an exponential run time. For some cyclic groups there are subexponential algorithms that involve smooth numbers.[3] At first thought you might wonder how smooth numbers can possibly be involved in group theory. The idea is to recognize the group as the unit group in a homomorphic image of the ring of integers $\mathbf{Z}$ or of another ring in which it makes sense to speak of smooth elements.

The prime example is the cyclic group $G = (\mathbf{Z}/p\mathbf{Z})^*$, pun intended. For p prime, G is cyclic of order $p-1$, and there is an obvious way to realize G as the group of units of a homomorphic image of $\mathbf{Z}$. Whenever we represent elements of G by integers, we are tacitly thinking in terms of this homomorphism. In particular, a multiplicative relation among integers leads to a multiplicative relation among group elements.

The idea with the index calculus method is to look at powers of a generator g of G. Again, staying with our example, if g^r is represented by an integer in the range $[1, p-1]$ and we happen to have the prime factorization of this integer, say as $p_1^{a_1} \cdots p_k^{a_k}$, then we have the index, or discrete logarithm, relation

$$r \equiv \log_g g^r \equiv a_1 \log_g p_1 + \ldots + a_k \log_g p_k \pmod{p-1}. \tag{1}$$

However, even though we know r and $a_1, \ldots, a_k$, we don't know the discrete logarithms $\log_g p_1, \ldots, \log_g p_k$. Thus, we may view (1) as sort of an equation in k unknowns.

Say we choose a smoothness bound B, and suppose that $p_1, \ldots, p_k$ are the primes up to B. We continually take random values of r, find the least positive integer representing g^r, and see if it is B-smooth. If it is, we get a relation as in (1). Continuing, suppose we have more than k of these relations. Then presumably, linear algebra will allow us to solve for the unknowns, namely the discrete logarithms $\log_g p_1, \ldots, \log_g p_k$.

So, this would allow us to find the discrete logarithms of the small primes. However, suppose you are interested in the discrete logarithm of t, and the integer representing t is not B-smooth. Then continue to choose random exponents r until one is found with the integer representing $g^r t$ being B-smooth, say it is $p_1^{b_1} \cdots p_k^{b_k}$. Then

$$r + \log_g t \equiv b_1 \log_g p_1 + \ldots + b_k \log_g p_k \pmod{p-1}.$$

But now, the only thing unknown in our relation is $\log_g t$, and this is then found instantly.

Choosing an optimal value of B and using the elliptic curve factoring method of H. W. Lenstra, Jr. to recognize B-smooth integers, the expected complexity of the outlined method is $\exp((\sqrt{2} + o(1))\sqrt{\log p \log\log p})$. Moreover, once the initial work is done

[3] See the lecture of A. Granville and also my lecture "Smooth numbers and the quadratic sieve."

to find the discrete logarithms of the small primes, the additional time to find the discrete logarithms of a given group element is much smaller, only about $\exp((1/\sqrt{2} + o(1))\sqrt{\log p \log\log p})$. (If a de-optimized larger value of B is used, the precomputation takes longer, but once it is done, the individual dl computations are even speedier.)

Even though the elliptic curve factoring method has not been rigorously analyzed *in toto*, it can be shown that it recognizes sufficiently many smooth numbers in subexponential time, that it may be used as a subroutine in some rigorously analyzed algorithms. This somewhat paradoxical, but happy occurence pertains for the index calculus method in $(\mathbf{Z}/p\mathbf{Z})^*$.

The key aspect of the index calculus algorithm is the use of smooth numbers. Can this always be done? That is an important question, and we really do not have a complete answer for various groups of interest. But for some groups, we can. For example, say we look at the multiplicative group of a finite field, $\mathbf{F}_q^*$, where $q = p^a$ is a prime power. We've just looked at the case $a = 1$. Let us look at the other extreme, $p = 2$. Does it make sense to speak of elements of $\mathbf{F}_{2^a}^*$ being smooth?

To answer this question, we think how the finite field $\mathbf{F}_{2^a}$ is "constructed." One way is to view it as $\mathbf{F}_2[x]/(f(x))$, where $f(x)$ is an irreducible polynomial over $\mathbf{F}_2$ of degree a. So now we may view our group as the group of units in a homomorphic image of the polynomial ring $\mathbf{F}_2[x]$. This ring is a unique factorization domain, where the prime elements are irreducible polynomials. And the degree of a polynomial gives us a measure of "size." That is, we can say a polynomial is B-smooth if all of its irreducible factors have degrees at most B. There is a completely analogous development of the study of the distribution of smooth polynomials as with the study of smooth integers, and yes we can obtain a rigorous, subexponential discrete logarithm algorithm for $\mathbf{F}_{2^a}^*$. In fact this works more generally for $\mathbf{F}_{p^a}^*$, and is subexponential in p^a as long as $a \to \infty$ arbitrarily slowly. (This is a result of Renet Lovorn Bender and me.)

What about the case $a > 1$, a fixed? Then we can represent $\mathbf{F}_{p^a}$ as the quotient ring of p in the ring of integers of an algebraic number field of degree a in which p is inert. And we may define smoothness in a number ring: an element is smooth if its norm to the rational integers is smooth. This heuristically gives a subexponential dl algorithm for all the cases of a fixed or slowly growing, and it does so rigorously in the case $a = 2$, a result of Lovorn Bender.

What makes elliptic curve groups of prime order so attractive for cryptography at present, is that we know no way of introducing smooth numbers to solve dl's in them. We seem to be condemned to use the earlier methods of this lecture.

Some people have also suggested using large subgroups of very large cases of $\mathbf{F}_q^*$. What about index calculus? Yes, it can be used, but only in the parent group, which is very large. So, as a function of the size of the subgroup, the complexity is prohibitive, even though it is a subexponential function of q. So, another unsolved problem is to find a way of introducing smooth numbers directly into the subgroup without going up to the parent group.

The basic ideas of the index calculus can be taken much further, with tremendous gains in efficiency. In particular, the number field sieve for factoring integers may be adapted to the dl problem for the multiplicative group of a finite field. This follows work of Don Coppersmith, Dan Gordon, Oliver Schirokauer, Len Adleman, and others.[4]

For further reading, connections to cryptography, and references to original papers and other surveys, see the following:

R. Crandall and C. Pomerance, *Prime numbers: a computational perspective*, Springer-Verlag, New York, 2001.
A. M. Odlyzko, *Discrete logarithms: The past and the future*, Designs, Codes, and Cryptography **19** (2000), 129–145.
O. Schirokauer, D. Weber, and T. Denny, *Discrete logarithms: the effectiveness of the index calculus method*, in "Algorithmic Number Theory: Proc. ANTS II, Talence, France," (H. Cohen, ed.) LNCS **1122**, Springer-Verlag, 1996, pp. 337–361.

[4] See the lecture of O. Schirokauer.