CCS Discrete Math I — Professor: Padraic Bartlett

Lecture 10: Elliptic Curve Cryptography

Week 10 — *UCSB 2014*

We don't know where the constants came from in the first place. We only know that whoever came up with them could have the key to this backdoor. And we know there's no way for NIST – or anyone else – to prove otherwise.

This is scary stuff indeed.

Bruce Schneier, cryptographer

1 Elliptic Curve Cryptography

1.1 Diffie-Hellman for elliptic curves.

When we discussed modern cryptographical systems in week 5, the main method that we studied was Diffie-Hellman. We implemented this cryptosystem as follows:

Algorithm. Suppose that we have two people, Alice and Bob, that want to communicate over a public network.

Ahead of time, Alice and Bob agree on a public base prime p (in practice, a 300+ digit prime) to work in, as well as a public seed g (usually 2, 3 or 5, depending on some small silly things that are tricky to go into.) All calculations from here on out are now done in $\mathbb{Z}/p\mathbb{Z}$.

From there, Alice and Bob each pick a "secret key" a, b from $\mathbb{Z}$. They then do the following:

1. Alice sends the number g^a publicly to Bob.
2. Bob then takes this received number g^a and uses his key to raise it to the b-th power, which gives him g^{ab}.
3. Bob then sends the number g^b publicly to Alice.
4. Alice then takes this received number g^b and uses her key to raise it to the a-th power, which gives her g^{ab}.
5. Bob and Alice are now both in possession of the same secret key g^{ab}. Any eavesdroppers will only have heard g^a and g^b, and thus have no obvious way to figure out g^{ab}.
6. To communicate a message $m \in \mathbb{Z}/p\mathbb{Z}$, then, either Alice or Bob can send to the other party $m \cdot g^{ab}$. No other parties know the secret key, so they cannot decode this message.

7. Moreover, because both Alice and Bob know the secret key g^{ab}, along with the values g^a, g^b, they can calculate g^{-ab} via
$$(g^a)^{(p-1)-b} = g^{a(p-1)-ab} = g^{a(p-1)} \cdot g^{-ab} = \left(g^{p-1}\right)^a \cdot g^{-ab} = g^{-ab},$$
because by Fermat's little theorem, we know that
$$g^{p-1} \equiv 1 \mod p.$$
Alice, using a, p, and g^b, can do the same trick as well; this allows both people to read their messages!

Fun fact: this works with any abelian group! In fact, we can make this work for the group associated to any elliptic curve E as follows:

Algorithm. This time, Alice and Bob agree on a public curve E and some point $P \in E$; as well, Alice and Bob each still pick a "secret key" a, b from $\mathbb{Z}$. They then do the following:

1. Alice calculates the point $aP = \overbrace{P + P + \ldots + P}^{a \text{ times}}$, and sends this publicly to Bob.
2. Bob then takes aP and adds it to itself b times to get abP.
3. Bob calculates bP and sends this point publicly to Alice.
4. Alice then takes bP and adds it to itself a times to get abP.
5. Bob and Alice are now both in possession of the same secret key abP. Any eavesdroppers will only have heard aP and bP, and thus have no obvious way to figure out abP.
6. To communicate a message m, determine some way of encoding a message m as a point $M \in E$, and have our our messaging party send $M + abP$ to the other
7. Because the inverse of any point abP is found by simply flipping the y-coördinate, it is easy for both Alice and Bob to calculate $-abP$; so they can both decode these messages! (In particular, this saves us the Fermat's last theorem exponentiation and work of the earlier method.)

The reason that both of these methods "work," in a sense, is because the **discrete logarithm problem**, described earlier, is a "hard" problem to solve:

Problem. The **discrete logarithm problem** is the following task: Suppose you're given a pair of numbers a, b in $\mathbb{Z} \mod p\mathbb{Z}$. The discrete logarithm problem asks the user for a value k such that
$$a^k \equiv b \mod p,$$
if some such value exists.

More generally: suppose we have some group G with an element $g \in G$. Suppose we are given g^a for some $a \in \mathbb{Z}$; the discrete logarithm problem is the task of finding a with only the knowledge of g and g^a.

One naive algorithm that you might be tempted to try is simply raising a to higher and higher powers, and then reducing mod p each time until you got something that is equal to b. However, this takes a hideously long time to run: if we just take values of k starting at 1 and going up, we're effectively looking through the elements of $\mathbb{Z}/p\mathbb{Z}$ one at a time, in the order $a, a^2, a^3 \ldots$. If we assume that b is chosen more or less at random, then we'd expect to find b after going through about half of $\mathbb{Z}/p\mathbb{Z}$ (and in the worst-case after going through all of $\mathbb{Z}/p\mathbb{Z}$.) In either case, we have runtime that is linear in the size of the group! Therefore, if our group has say 300 digits (like we have for the size of prime requested in the algorithm above,) our algorithm has runtime that's absolutely awful (i.e. exponential in the number of digits in the size of the group.)

There, however, are better algorithms! This, in fact, is one of the reasons that we are widely moving to elliptic curve cryptography over methods like Diffie-Hellman over finite fields or RSA or other methods; while we do not have a polynomial-time algorithm to solve the discrete logarithm problem for $\mathbb{Z} \mod p\mathbb{Z}$, we do have methods that are much faster than you might suspect! To illustrate one way to solve the discrete logarithm problem for $\mathbb{Z}/p\mathbb{Z}$, called the **index calculus**, we make the following definition:

Definition. Take any prime p and any $g \in \mathbb{Z}/p\mathbb{Z}$. Suppose that there are values $a \in \mathbb{Z}/p\mathbb{Z}, h \in \mathbb{Z}/p\mathbb{Z}$ such that $g^a = h$. We then **define** the discrete logarithm L over $\mathbb{Z}/p\mathbb{Z}$ with respect to g by $L(h) = a$.

Notice that just like how normal logs transformed multiplication[1] into addition, the discrete log does this as well: for any two h_1, h_2 such that $g^{a_1} = h_1, g^{a_2} = h_2$, we have

$$L(h_1 h_2) = L(g^{a_1 + a_2}) = a_1 + a_2 = L(g^{a_1}) + L(g^{a_2}) = L(h_1) + L(h_2).$$

Algorithm. Eve is eavesdropping on two people, Alice and Bob, who are communicating using Diffie-Hellman over a finite field. Thus far through Alice and Bob's communication, Eve has seen g^a by wiretapping Alice's sent communications, and knows $g, \mathbb{Z}/p\mathbb{Z}$ because these two pieces of information are publicly known.

Eve now wants the ability to "impersonate" Alice: that is, given g^a, a, and p, she wants the ability to find a. She can do this via the **index calculus**, as defined here:

1. At first, Eve will pick out some list of small primes $p_1 = 2, p_2 = 3, \ldots p_r$, ranging from 2 up to the r-th prime number. It is remarkably difficult to describe what the optimal choice of r is here (see this link and this link for some discussion that demonstrates how hard, and in some senses still open, this question is!).

   Call this collection of small primes the **factor base** for our calculations.

2. Now, repeatedly take random powers of g. For each such random power g^m, factor the resulting number into primes. If you ever get a result like

   $$g^m = p_{i_1}^{k_1} \ldots p_{i_j}^{k_j},$$

   for some collection of prime $p_{i_1}, \ldots p_{i_j}$ all in our factor base, write this equation down. Notice that if we take discrete logarithms of both sides, we get

   $$m = L(g^j) = L(p_{i_1}^{k_1} \ldots p_{i_j}^{k_j}) = k_1 L(p_{i_1}) + k_2 L(p_{i_2}) + \ldots + k_j L(p_{i_j}).$$

[1] To be explicit, $\log(ab) = \log(a) + \log(b)$ is the property we're referring to here.

In other words, we have a linear equation involving j of our factor base primes!

Keep doing this process until you get enough linear equations to solve for the discrete logarithms of our factor base primes. Again, it is hard / in some senses open to determine about how many times you will need to do this process, but you can do it!

3. Take your collection of linear equations for the $L(p_j)$'s, and solve for all of them! We now know $L(p_j)$ for every p_j in our factor base.

4. Finally: take g^a, which it is given that we know. Repeatedly pick out random values $m \in \mathbb{Z}/p\mathbb{Z}$, calculate $g^m \cdot g^a$, and factor the result into primes. Suppose that at some step we eventually get that $g^m \cdot g^a$ factors into some product of the "small" primes in our factor base: then we have

$$\begin{aligned} g^m \cdot g^a &= p_{i_1}^{k_1} \dots p_{i_j}^{k_j} \\ \Rightarrow L(g^m \cdot g^a) &= L(p_{i_1}^{k_1} \dots p_{i_j}^{k_j}) \\ \Rightarrow m + a &= k_1 L(p_{i_1}) + k_2 L(p_{i_2}) + \dots + k_j L(p_{i_j}) \\ \Rightarrow a &= k_1 L(p_{i_1}) + k_2 L(p_{i_2}) + \dots + k_j L(p_{i_j}) - m. \end{aligned}$$

We know all of the $L(p_i)$'s from step 3; therefore we've solved for a!

This is overly abstract as described above; so let's calculate an example!

Example. Suppose that two parties Alice, Bob are communicating using finite-field Diffie-Hellman where $p = 101$ and $g = 2$. Suppose that we've intercepted $2^a = 83$. Can we find a?

To solve this, we use the index calculus method. We start by fixing a collection of small primes, say $\{2, 3, 5, 7\}$. From here, we calculate small values of 2^m for various values of m, highlighting only those results that are products of elements in our factor base.

$$\begin{array}{rrr} 2^1 = 2, & 2^2 = 4, & 2^7 = 3^3, \\ 2^8 = 2 \cdot 3^3, & 2^9 = 7, & 2^{13} = 11, \\ 2^{17} = 3 \cdot 5^2, & 2^{18} = 7^2, & 2^{21} = 89, \\ 2^{22} = 7 \cdot 11, & 2^{23} = 53, & 2^{24} = 5. \end{array}$$

We now use these equations to solve for $L(2), L(3), L(5), L(7)$!

One important wrinkle to note here is that all of the arithmetic done to solve these linear equations is not done in $\mathbb{Z}/101\mathbb{Z}$, as you might expect, but rather $\mathbb{Z}/100\mathbb{Z}$! To see why this is true, notice that because 101 is prime, we have that $(\mathbb{Z}/101\mathbb{Z})^\times$ is a multiplicative group with 100 elements (namely, all of the numbers in $\mathbb{Z}/101\mathbb{Z}$ except for 0.) We know that for any group of order n and any element g in that group, $g^n = id$ by Fermat's little theorem! Therefore, $2^{100} = 1 = 2^0$, and therefore for any x we have $2^{100+x} = 2^{100}2^x = 2^x$.

So, if we're working with the subgroup of $\mathbb{Z}/100\mathbb{Z}$ generated by all of the elements of the form 2^x for some x, we have that $2^x = 2^y$ whenever $x \equiv y \mod 100$! For our logarithms in particular, this means that if we have an equation like

$$7 = L(2^7) = L(27) = L(3^3) = 3L(3),$$

we're really talking about

$$2^7 = 2^{3L(3)},$$

and therefore want to handle our multiplication/division/etc mod 100! In particular, this tells us here that $3L(3) \equiv 7 \mod 100$. We can solve this in this case; because 3 and 100 are relatively prime, we know that there is a multiplicative inverse of 3 in $\mathbb{Z}/100\mathbb{Z}$, namely 67, because $3 \cdot 67 = 101 \equiv 1 \mod 100$. Therefore, we know that

$$67 \cdot 3L(3) \equiv L(3) \equiv 67 \cdot 7 \equiv 69 \mod 100,$$

and therefore that $L(3)$ should be 69. Checking via calculator verifies that this works!

From our highlighted equations, we can deduce that

$$L(2) = 1, L(3) = 69, L(7) = 9, L(5) = 24.$$

From here, our lives are not too hard. We simply calculate values of $2^m \cdot 83$ until we get something that is a product of primes in our factor basis:

$$2^2 \cdot 83 = 29, \qquad 2^3 \cdot 83 = 2 \cdot 29, \qquad 2^4 \cdot 83 = 3 \cdot 5.$$

Taking logs here gives us that

$$\begin{aligned} L(2^4 \cdot 83) &= L(3 \cdot 5) \\ \Rightarrow 4 + L(83) &= L(3) + L(5) \\ \Rightarrow L(83) &= 69 + 24 - 4 \equiv 89 \mod 100, \end{aligned}$$

and therefore that $2^{89} = 83$. Success!

This algorithm, with appropriate choices of factor base, runs in time $O(e^{\sqrt{2\ln(p)\ln(\ln(p))}})$. We can't prove this here, because we can't even quite say what the best choice of factor base is without a lot more machinery; but the relevant thing to take away here is that this growth rate is slower than exponential in the number of digits in p (that is, $\ln(p)$!) It's still growing faster than any polynomial (check this if you don't believe it!), but it is much smaller than say $p = e^{\ln(p)}$, the runtime of our "brute-force" algorithm from before. In practice, this forces us to want to use primes p that are about 300 digits long or so for finite-field Diffie-Hellman to get an acceptable level of security. (If our best-known algorithm was the brute-force algorithm, then we could get away with just 60-digit primes, as $e^{60} \sim O(e^{2\sqrt{300\ln(300)}})$.)

Also, it's worth noting that when we run the algorithm above, the first three steps are "pre-computable:" that is, we can do them without knowing g^a! So, if someone uses the above algorithm to the discrete log problem for a given p, g^a, they've actually done almost all of the work for **any** g^a! This is perhaps worrisome; not only can we solve this problem in not-completely-awful amounts of time, once it's solved we've solved it for pretty much **all** keys in the field, instead of just for one!

These observations are perhaps the best motivation for elliptic curve cryptography that exist. Unlike finite fields, there is no known algorithm for solving the discrete logarithm problem for arbitrary elliptic curves that runs in sub-exponential time like the above! In fact, the best that most algorithms can do for an elliptic curve over $\mathbb{F}_p$ is (roughly) just $\sqrt{p} = e^{\ln(p)/2}$, which is far smaller than the index calculus's runtime; to get equivalent security to a 300-digit prime there, we just need to use a 120-digit prime!

We illustrate one method for solving the discrete logarithm problem, called the "baby-step, giant-step" algorithm, here:

Algorithm. Suppose that we have **any** arbitrary finite group G of order n, any base element $g \in G$, and an element $h \in G$ that we know is of the form g^a for some $a \in \mathbb{Z}$. This algorithm illustrates one process for finding[2] a given this information.

1. Let $m = \lceil \sqrt{n} \rceil$. Calculate $1, g, g^2, g^3, \ldots g^{m-1}$, and record all of these values in a table that we can quickly look up elements in.

2. Now, for each $j \in \{0, 1, \ldots m-1\}$, calculate the value
$$h \cdot g^{-mj},$$
and check to see if it shows up in our table of precomputed values.

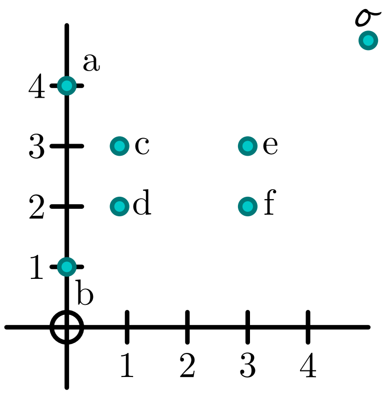
3. If this happens, then there is some i, j such that
$$g^i = h \cdot g^{-mj} \quad \Rightarrow \quad h = g^{i+mj},$$
and therefore that we've solved our discrete logarithm problem!

The only point to wonder about with the above algorithm is **why** it must work. This is fairly simple: because the group G is of order n, notice that we can assume that $a \in \{0, 1, \ldots n-1\}$ without losing any generality, as Fermat's little theorem tells us that $g^n = id$ whenever n is the order of the group. From here, notice that we can write any number a from 0 to $m^2 - 1 \geq n-1$ in the form $i + j \cdot m$, by just looking at its quotient and remainder when divided by m. The process above goes through all pairs $i + j \cdot m$; consequently it must eventually find our number!

[2]Strictly speaking, this finds a modulo the order of the element g. This is because if the order of g is m, then $g^m = g^0$, and thus we cannot tell g^{x+m} from g^x.

Again, to give a feel for this algorithm, we calculate an example:

Example. Take the elliptic curve E consisting of the collection of points in $\mathbb{F}_5^2 = (\mathbb{Z}/5\mathbb{Z})^2$ such that $y^2 = x^3 + 2x + 1$, that we studied last week:



Suppose that we take c as our generator, and we want to solve the discrete logarithm problem for $n \cdot c = e$. What is n? (Note that we are using multiplication instead of exponentiation; this is because our group operation here is now $+$ instead of $\cdot$, and so repeated applications of c to itself look like $\overbrace{c + c + \ldots + c}^{n \text{ times}} = nc$.)

To answer this, we first recall the group table we calculated last week:

$+$	$\mathcal{O}$	a	$-a$	c	$-c$	e	$-e$
$\mathcal{O}$	$\mathcal{O}$	a	$-a$	c	$-c$	e	$-e$
a	a	$-c$	$\mathcal{O}$	$-a$	$-e$	c	e
$-a$	$-a$	$\mathcal{O}$	c	e	c	$-e$	$-c$
c	c	$-a$	e	$-e$	$\mathcal{O}$	$-c$	a
$-c$	$-c$	$-e$	c	$\mathcal{O}$	e	a	$-a$
e	e	c	$-e$	$-c$	a	$-a$	$\mathcal{O}$
$-e$	$-e$	e	$-c$	a	$-a$	$\mathcal{O}$	c

To perform the little-step big-step algorithm here, we proceed as follows:

1. (Little-step.) Our group is size 7, so we set $m = \lceil\sqrt{7}\rceil = 3$ and precalculate $0c = \mathcal{O}, 1c = c$, and $2c = -e$.

2. (Big-step.) Now, we repeatedly calculate $e + (jm)c$ for values of $j \in \{0, 1, 2\}$, until we get an entry in our list:

$$e - (0 \cdot 3)c = e$$

$$e - (1 \cdot 3)c = e - a = -e$$

3. (Conclusion.) Therefore, we have $2c = e - 3c$, which implies $e = 5c$, and thus that $n = 5$. Success!

Cryptography! As you probably expect, there is a **lot** more depth you could go into when studying these problems. Write me for book references if you want to go further here!