Cryptology Fall 2017

Chloe Martindale
TU/e

October 5, 2017

These notes are based on notes by Tanja Lange. In the last few lectures we constructed some cryptosystems based on the hardness of the discrete logarithm problem and looked at the best known generic attacks. We will now see the best known attack for the discrete logarithm problem applied to a group which is a subgroup of $\mathbb{F}_q^*$, called the *index calculus method*.

1 Index calculus method

The index calculus method is a way of solving the discrete logarithm problem for finite fields. Warning: this method does not apply to general groups!

As with all our algorithms to attack DLPs, our aim is, given $h \in \langle g \rangle \subseteq \mathbb{F}_q^*$, to find $a := log_g(h)$. To make things simpler, for now we will consider only finite fields with $q = p$ prime. The algorithm for $q = p$ is prime is as follows:

1. Lift h to an integer $\overline{h}$ (i.e. choose $\overline{h} \in \mathbb{Z}$ such that $h \equiv \overline{h} \bmod p$).
2. Factor $\overline{h}$ into prime powers $\overline{h} = \prod_{i=1}^{n} p_i^{e_i}$.
3. Compute $\log_g(p_i)$ for $i = 1, \ldots, n$.
4. Compute
$$\log_g(h) = \log_g(\prod_{i=1}^{n} p_i^{e_i}) = \sum_{i=1}^{n} e_i \log_g p_i.$$

Observations.

- At first glance, this algorithm may not look very efficient: in the context of RSA we said that (2) is hard, and in (3) we have to compute discrete logarithm problems. But: some numbers are easy to factor! For example 2^n.
- We call a number with only small prime factors a *smooth number*. For smooth numbers, both (2) and (3) will not be too hard.
- Even if $\overline{h}$ is not smooth, chances are that there is some small j for which $\overline{g^j h}$ *is* smooth. So, we try a few j until we find a smooth number $\overline{g^j h}$, compute $\log_g(g^j h)$ with the algorithm above, and then compute
$$\log_g(h) = \log_g(g^j h) - j.$$

- For step (3), we want to compute $\log_g(p_i)$ for lots of small primes p_i, called the *factor base* $\mathcal{F}$.
- If $|\mathcal{F}|$ is large, it is more likely that we find a small j such that $\overline{g^j h}$ factors of $\mathcal{F}$ (i.e. all the prime factors of $\overline{g^j h}$ are in $\mathcal{F}$).
- If $|\mathcal{F}|$ is large, we have to compute more discrete logarithm problems, which gives a subtle trade-off. So choosing a good $\mathcal{F}$ can only be done with careful study of the available space and computing time.
- If all the prime factors of $\overline{h}$ are in $\mathcal{F}$, we say that $\overline{h}$ is $\mathcal{F}$-smooth.

Let's apply this algorithm to an example:

Example. Set $p = 107$, $g = 17$, and $h = 91$. That is, we want to compute a such that $17^a \equiv 91 \bmod 107$. Observe that $\langle 17 \rangle = \mathbb{F}_{107}^*$, so for every $x \in \langle 17 \rangle$ we have that $x^{106} = 1$, so that the powers of elements in $\langle 17 \rangle$ can be computed mod 106.

We choose a factor base

$$\mathcal{F} = \{2, 3, 5\},$$

so we want to compute $\log_{17}(2)$, $\log_{17}(3)$, and $\log_{17}(5)$. To do this, we first compute 17^j for $j = 0, 1, \ldots$ until we find $3 = |\mathcal{F}|$ values of j such that 17^j is $\mathcal{F}$-smooth:

$$17^2 \equiv 3^1 \cdot 5^2 \bmod 107,$$

hence

$$2 = \log_{17}(3) + 2\log_{17}(5). \tag{1}$$

Then $17^3, \ldots, 17^8$ all have primes that are not in $\mathcal{F}$ in their factorisation, but

$$17^9 \equiv 2^2 \cdot 5^1 \bmod 107,$$

hence

$$9 = 2\log_{17}(2) + \log_{17}(5), \tag{2}$$

and

$$17^{11} \equiv 2 \bmod 107,$$

hence

$$\log_{17}(2) = 11. \tag{3}$$

Now, observe that (1), (2), and (3) are just 3 simultaneous equations with variables $\log_{17}(2)$, $\log_{17}(3)$, and $\log_{17}(5)$, so we can solve them to get

$$\log_{17}(2) \equiv 11 \bmod 106, \quad \log_{17}(3) \equiv 28 \bmod 106, \text{ and } \quad \log_{17}(5) \equiv 93 \bmod 106.$$

Now that we have the discrete logarithms of the primes in our factor base $\mathcal{F}$, we search for a j such that $\overline{g^j h}$ is $\mathcal{F}$-smooth, that is, such that all the prime factors of $\overline{g^j h}$ are in $\mathcal{F}$. This does not hold for $j = 0, 1, 2, 3, 4$, but

$$91 \cdot 17^5 \equiv 2^2 \cdot 5^2 \bmod 107.$$

Therefore,

$$\begin{aligned}\log_{17}(91) &= \log_{17}(17^{-5} \cdot 17^5 \cdot 91) \\ &= -5 + \log_{17}(17^5 \cdot 91) \\ &\equiv -5 + \log_{17}(2^2 \cdot 5^2) \bmod 106 \\ &\equiv -5 + 2\log_{17}(2) + 2\log_{17}(5) \bmod 106 \\ &\equiv -5 + 2 \cdot 11 + 2 \cdot 93 \bmod 106 \\ &\equiv 97 \bmod 106.\end{aligned}$$

From the size of the example, perhaps we already see that the index calculus method is much faster that Pollard rho or baby step giant step, but how do we measure this? Define

$$L_N(\alpha, c) = e^{c(\log N)^\alpha (\log\log N)^{1-\alpha}}.$$

This looks very complicated, but notice that

$$L_N(1, c) = e^{c \log N} = N^c$$

describes the complexity of a function that is *exponential* in N, and

$$L_N(0, c) = e^{c \log\log N} = (\log N)^c$$

describes the complexity of a function that is *polynomial* in N. We use $L_N(\alpha, c)$ to measure how close a function is to being polynomial or exponential; for $0 < \alpha < 1$ we say that the function is *subexponential* or *superpolynomial*.

The complexity of the index calculus algorithm that we described above is $L_q(1/2, c)$, and c depends on whether q is prime, a power of 2, or otherwise. With optimisations, the index calculus algorithm has complexity $L_q(1/3, c)$, so it is indeed much better than Pollard rho or baby-step-giant-step. Note that the complexity is independent of the order of g (which we called ℓ before) - it depends only on the size of the finite field.

Due to index calculus attacks, q has to be > 3000 bits to achieve 128-bit security with for example a Diffie-Hellman key exchange using $\mathbb{F}_q^*$, whereas if we can construct a different group with which we can perform Diffie-Hellman that is not susceptible to index calculus attacks, then q of 256 bits will suffice to protect against Pollard rho and baby-step-giant-step.

2 Elliptic curves

We will construct a group that is not just a subgroup of $\mathbb{F}_q^*$ by studying 'elliptic curves'. For suitably chosen 'elliptic curves' no subexponential attacks are known, so that (the optimised version) of Pollard rho is the best known attack! This means that smaller finite fields can be used than if we do crypto with $\mathbb{F}_q^*$. The advantage of this is that the arithmetic is faster and the protocols use less memory, which is good for small devices.

Before we get to elliptic curves, we'll try to construct a group on a more familiar curve: a circle. This is not an elliptic curve! Warning: elliptic curve and ellipse mean different things!

The equation of a circle over $\mathbb{R}$ is given by

$$x^2 + y^2 = 1,$$

so that the set of points on the circle is given by

$$G = \{x, y \in \mathbb{R} : x^2 + y^2 = 1\}.$$

Some examples of points:

$$(0,1),\ (0,-1),\ (1,0),\ (-1,0),$$

$$(\sqrt{3}/2, 1/2),\ (-\sqrt{3}/2, 1/2),\ (\sqrt{3}/2, -1/2),\ (-\sqrt{3}/2, -1/2),$$

$$(1/2, \sqrt{3}/2)\,(1/2, -\sqrt{3}/2),\ (-1/2, \sqrt{3}/2),\ (-1/2, \sqrt{3}/2).$$

We want our set G to be a *group*. Remember that a group has a *group operation* such as $+$ or $\times$. Let's call our group operation $\oplus$. So, we need a way of 'adding' points in G so that the sum of 2 points is also in G. That is, we want to use 2 points on the circle to get another point on the circle.

Recall that $x^2+y^2 = 1$ is paramatrised by $x = \sin(\alpha)$ and $y = \cos(\alpha)$, where α is the angle between the y-axis and the line passing through $(0,0)$ and (x,y). Let's try to define $\oplus$ by adding the angles, that is

$$(x_1, y_1) \oplus (x_2, y_2) = (\sin(\alpha_1, \cos(\alpha_1) \oplus (\sin(\alpha_2), \cos(\alpha_2)) = (\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)).$$

We can check that this makes $(G, \oplus)$ into a group by checking the group axioms:

(G1) $(x_1, y_1) \oplus (x_2, y_2) \in G$ by construction.

(G2) $(x_1, y_1) \oplus ((x_2, y_2) \oplus (x_3, y_3)) = ((x_1, y_1) \oplus (x_2, y_2)) \oplus (x_3, y_3)$ by construction.

(G3) The neutral element is $(0, 1)$, as this corresponds to the zero angle.

(G4) The inverse of (x, y) is $(-x, y)$, as the sum of the angles gives the zero angle.

All our axioms are satisfied, so $(G, \oplus)$ is a group. Using the addition formulae from trigonometry, we can even write down a 'group law':

$$\begin{aligned}(x_1, y_1) \oplus (x_2, y_2) &= (\sin(\alpha_1), \cos(\alpha_1)) \oplus (\sin(\alpha_2), \cos(\alpha_2)) \\ &= (\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) \\ &= (\sin\alpha_1 \cos\alpha_2 + \sin\alpha_2 \cos\alpha_1, \cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2) \\ &= (x_1 y_2 + x_2 y_1, y_1 y_2 - x_1 x_2).\end{aligned}$$

This means that we can add points without computing the angles corresponding to them using the group law, for example

- '9:00' $\oplus$ '6:00' $= (-1,0) \oplus (0,-1) = (-1 \cdot -1 + 0 \cdot 0, 0 \cdot -1 - (-1) \cdot 0) = (1,0)$ = '3:00'.
- $2 \cdot (3/5, 4/5) = (3/5, 4/5) \oplus (3/5, 4/5) = (3/5 \cdot 4/5 + 4/5 \cdot 3/5, (4/5)^2 - (3/5)^2) = (24/25, 7/25)$.

Having succeeded in constructed a group on a circle, let's see if we can use this do construct a group on a 'circle over a finite field'. Recall that our group, the set of points on the circle, was given by

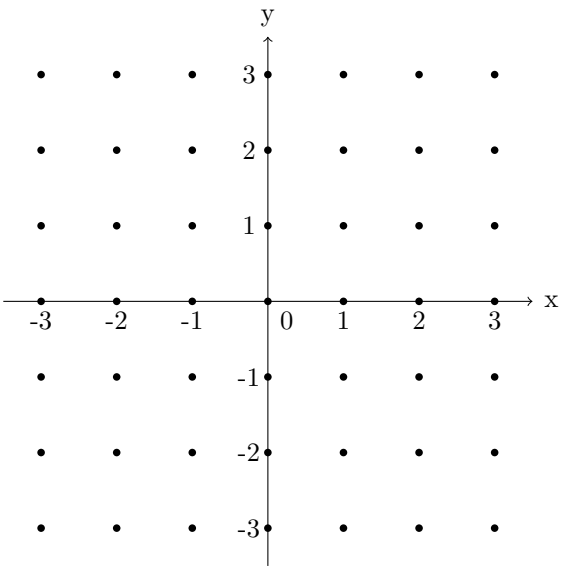
$$\{(x,y) \in \mathbb{R} \times \mathbb{R} : x^2 + y^2 = 1\}.$$

So we define set the points on a circle over a finite field $\mathbb{F}_q$ to be

$$G_q = \{(x,y) \in \mathbb{F}_q \times \mathbb{F}_q : x^2 + y^2 = 1\}.$$

Can we make a group from G_q like we did with the circle over $\mathbb{R}$? Let's first look at a small example.

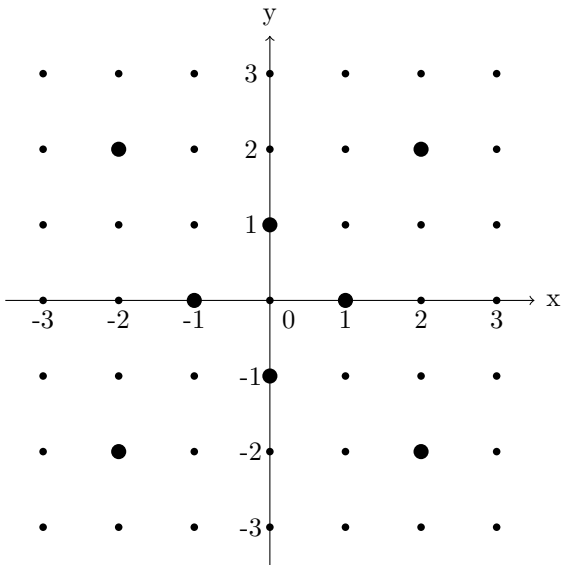
Example. Represent $\mathbb{F}_7$ as $\mathbb{F}_7 = \{-3, -2, -1, 0, 1, 2, 3\}$. Then we can draw $\mathbb{F}_7 \times \mathbb{F}_7$ as



and we can draw the 'circle'

$$G_7 = \{(x,y) \in \mathbb{F}_7 \times \mathbb{F}_7 : x^2 + y^2 = 1\}$$

as



Now, we can use the group law that we found for the circle over $\mathbb{R}$ and check that it defines a group law for G_7, by defining

$$(x_1, y_1) \oplus (x_2, y_2) = (x_1y_2 + x_2y_1 \bmod 7, y_1y_2 - x_1x_2 \bmod 7)$$

for (x_1, y_1) and $(x_2, y_2) \in G_7$. Checking that this satisfies the group axioms (G1)-(G4) is left as an exercise.

Now, we have defined a group $(G_7, \oplus)$, and in a similar way we can define a group $(G_p, \oplus)$ for any prime p. Now that we have a group, we can do a Diffie-Hellman key exchange with this group:

Setup:

Alice and Bob publicly agree on a finite field $\mathbb{F}_p$ and a point (x, y) such that $x^2 + y^2 \equiv 1 \bmod p$.

Key exchange:

1. Alice choose a random private key $a \in \mathbb{Z}$ and Bob choose a random private key $b \in \mathbb{Z}$.

2. Alice computes her public key $a \cdot (x, y) := (x, y) \oplus (x, y) \oplus \cdots \oplus (x, y)$ (a times), and Bob computes his public key $b \cdot (x, y)$.

3. Alice sends Bob $a \cdot (x, y)$ and Bob sends Alice $b \cdot (x, y)$ (over an insecure channel).

4. Alice computes their shared secret $a \cdot (b \cdot (x, y)) = (ab) \cdot (x, y)$, as does Bob via $b \cdot (a \cdot (x, y)) = (ab) \cdot (x, y)$.

This key exchange works, but is still susceptible to index calculus-like attacks due to the simplicity of the group law, so we need to do something slightly

cleverer to make a more secure protocol. Next time we will see how to alter this example to get a group law on an *Edwards curve*, which is an example of an elliptic curve.