Pollard's Kangaroo Method

November 26, 2016

The Kangaroo method (referred to by some as λ method) for computing discrete logarithms was introduced by J.M.Pollard in 1978 along with his ρ method [1]. The method is probabilistic, and its key component closely resembles Kruskal's "card trick" [2]. It is also easy to parallelize, and is particularly effective if, given the standard DLP setup $g^x \equiv \beta \pmod{p}$ the the exponent x is known to lie in a certain range $a \leq x \leq b$.

The general idea of the method (and the reason why it is named after the animal) is to let two Kangaroos – a wild one and a tame one – travel on their own until their paths cross and they fall into the same trap.

Converting the analogy into mathematical terms, this is how the method works:

1. Let G be the set of unique elements $G_i \equiv g^i \pmod{p}$.
2. Let S be a set of integers $S = \{S_1, S_2, \ldots, S_k\}$.

   This is a set of jumps that either of the Kangaroos may do on their path. Pollard performs some analysis in [1] and [3] to make a few recommendations regarding the set:

   - The size of the set, $|S|$, should be significantly smaller than $\sqrt{|G|}$
   - The set's mean, m, should be approximately $\sqrt{|G|}$
   - The set may (and, more often than not, should) contain duplicates as long as its elements are in a non-decreasing order
   - The set's elements should be powers of 2 (which happen to be easy to generate).
   - The set need not be stored in memory – each of the values can be calculated dynamically as long as they don't change.
3. Choose a hash function H. This function will provide a pseudorandom map $H : G \to S$. The goal of using this hash function is to pick "random" jumps depending on the Kangaroo's position on the path. Even though not mentioned in Pollard's paper, it is easy to reason that the efficiency of H is more important than its other qualities (such as "randomness"), because in practice it affects the runtime of the entire algorithm linearly (see below why).

4. Release the Tame kangaroo. Pick a number N (which may be some multiple of m, as suggested by Pollard in [1]) and compute the sequence $\{x_0, x_1, \ldots, x_N\}$ using a recursive rule:

   - $x_0 \equiv g^b \pmod{p}$ (where b is the upper bound of the range in which we expect the exponent to lie)
   - $x_i \equiv x_{i-1} \times g^{h(x_{i-1})} \pmod{p}$

   Observe:

   $$x_N \equiv g^b \times g^{h(x_0)} \times g^{h(x_1)} \times \ldots \times g^{h(x_{N-1})} \equiv g^{b+h(x_0)+h(x_1)+\ldots+h(x_{N-1})}$$

   Putting it into context, x_N resembles the trap which we want the Wild kangaroo to fall into.

5. We now calculate d, which will stand for the distance travelled by the Tame kangaroo:

   $$d = h(x_0) + h(x_1) + \ldots + h(x_{N-1}) = \sum_{i=0}^{N-1} h(x_i)$$

   Observe that $x_N = g^{b+d}$ (in other words, the position of the Tame kangaroo is equivalent to its starting point b plus the distance travelled d)

6. Release the Wild kangaroo. Begin computing the sequence $\{y_0, y_1, \ldots\}$ using a recursive rule similar to the one in step 3:

   - $y_0 \equiv \beta \equiv a^x \pmod{p}$
   - $y_i \equiv y_{i-1} \times a^{h(y_i-1)} \pmod{p}$

   Keep track of the Wild kangaroo's travel distance after each jump:

   $$d'_n = h(y_0) + h(y_1) + \ldots + h(y_n) = \sum_{i=0}^{n} h(y_i)$$

7. Stop once the Wild kangaroo either falls into or passes the trap which the Tame kangaroo is in:

   - If $y_i = x_N$, then the trap worked and the Wild kangaroo was caught. The exponent we are looking for can now be obtained using the formula: $x = d'_i - d + b$
   - If $d'_i > b - a + d$, then the Wild kangaroo passed the trap (i.e. the algorithm failed)