Discrete Log in the Symmetric Group

November 14, 2023

1 Introduction

The *Diffie-Hellman Key Exchange* (*DH*) and the *El Gamal Cryptosystem* are based on the *Discrete Logarithm Problem* (*DLP*): in the setting of an arbitrary group $G = \langle g \rangle$ of size n, given g^x, find x.

The Discrete Logarithm is conjectured to be a one-way function, i.e., easy to compute, but impossible to revert given the capacities of a reasonably resourced adversary. However, not all groups give us this infeasible-inversion property. The present challenge illustrates how breaking the DLP is trivial in the *Symmetric Group* S_n. S_n is defined over a set X and contains all permutations of X.

Example. S_3 is defined over $X := \{1, 2, 3\}$ and contains the elements $\{1, 2, 3\}, \{1, 3, 2\}, \{2, 1, 3\}, \{2, 3, 1\}, \{3, 1, 2\}, \{3, 2, 1\}$.

In the challenge, the symmetric group is used for a DH key exchange: Alice picks a random integer a, and Bob picks a random integer b. These integer values constitute the participants' secret keys. Subsequently, both take the base element p and compute their public keys: $A = p^a$ in the case of Alice, $B = p^b$ in the case of Bob. Alice sends A to Bob, and Bob sends B to Alice. Alice computes: $K = B^a = p^{ab}$, Bob computes $K = A^b = p^{ba} = p^{ab}$ (the symmetric group is commutative). Thus, they now share a symmetric key they can use for exchanging messages.

2 The Solution

How can we break the scheme given p, A, and B? Observe that if we were able to either determine a from $A = p^a$ or b from $B = p^b$, i.e., if we were able to solve the Discrete Logarithm problem to obtain one of the private keys from the respective public key, then we could compute K and get the flag. Luckily, in the symmetric group it is relatively easy to do so! Here are the steps:

1. Firstly, we need to obtain the disjoint cycle representations of p, A, B. Each permutation can be decomposed into disjoint cycles: e.g., regard

the permutation $f = \{3, 2, 1, 5, 4\}$. We interpret it as follows: in the first place, at the first index if you will, where we'd find the number 1 if no permuting had taken place, we find the number 3. But in the spot where we'd expect the number 3 if there had not been any permuting, we find the number 1! Thus, our first cycle is $(1, 3)$. Continuing, we find the number 2 at index 2, so it is just mapped to itself. This means that 2 is in its own cycle of length 1. Next, we have left the numbers 4 and 5, which have swapped places (4 is at index 5, and vice versa). In total, the disjoint cycle representation is $f = (1, 3)(2)(4, 5) = (1, 3)(4, 5)$ (trivial cycles of length 1 are commonly left out). In our case, the permutations are way bigger and start at index 0.

2. Secondly, we need to determine the shifts for each of the disjoint cycles. The shift per cycle is defined as follows: consider the respective cycle in its permutation occurring in the public key. Register the first and second element of the cycle. Next, regard the corresponding cycle in its permutation occurring in the base element. Count the steps between the first and second element of the cycle's permutation as found in the public key based on the cycle's permutation as found in the base element. For example: assume the base element is $g = (1, 7)(2, 6, 8)(3, 5, 4, 9, 10)$ and Alice's public key is $A = g^a = (1, 7)(2, 8, 6)(3, 4, 10, 5, 9)$. Then for the first cycle, the first and second elements based on A are 1 and 7. Looking at the respective cycle in g, we can see that there is one step between these elements. For the second cycle, the first and second elements as per A are 2 and 8. In g, 2 steps separate these elements. For the third and last cycle, the first and second elements as per A are 3 and 4, and in g's third cycle, there are two steps between 3 and 4.

3. Thirdly, based on the disjoint cycle representation and the shifts determined in step 2, we can compute an instance of the (generalized) Chinese Remainder Theorem, where the congruences are formed by pairs of a shift and the length of the corresponding cycle. Continuing the example from step 2, we obtain the three congruences

$$\begin{aligned} a &\equiv 1 \mod 2 \\ a &\equiv 2 \mod 3 \\ a &\equiv 2 \mod 5 \end{aligned}$$

   We may need the *generalized* version of the Chinese Remainder Theorem here because the cycle lengths are not necessarily co-prime; see the function `generalized_crt` in `crt.py` (implementation based on this blog post).

Unfortunately, I must have failed somewhere there - and currently cannot wrap my head around where. My efforts thus far are in `sol.py`, might continue at some point if I should feel inspired! A small proof of concept in which the whole thing seems to work can be found in `poc.ipynb`.