Brief description of the algorithms

Vaibhav Sinha

May 7, 2020

1 Discrete Log Problem

In this problem we are given g, y and n and we have to find an x such that $g^x \equiv y \pmod{n}$. We use this $\equiv$ and $=$ alterchangebaly in the report and the meaning is clear from the context.

First notice a few things:

1. If $g = 0$ or 1 then all powers x would only produce 0 and 1 respectively. We can always output 1 in this case. Also observe that without loss of generality, $g < n$. So in the rest of the report we will only consider $g \in \{2 \dots n-1\}$.

2. When n is prime, then $y \neq 0$, as $y = 0$ can not have any solution. So in such cases, $y \in [n-1]$ and $x \in [n-1]$. Notice that one can equivalently take the $x \in \{0, \dots n-2\}$ as $g^0 \equiv g^{n-1} \equiv 1 \pmod{g}$.

We now describe a few algorithms to solve the discrete log problem. Observe that none of these algorithms are truly polynomial time. No efficient algorithm are known for solving Discrete Log Problem in general setting. We implement these algorithms (and some of these only work) modulo a prime as Diffie-Hellman uses a prime modulus.

1.1 Brute Force Algorithm

In this algorithm we simply try all possible x's. This takes $O(n)$ time.

1.2 Baby-Step Giant-Step Algorithm

This algorithm trades off space to improve the running time over brute force. The idea is to write $x = i\lceil n\rceil + j$ for $0 \leq i, j \leq \lceil n\rceil$. For simplicity let $m = \lceil n\rceil$. First compute g^j for all $0 \leq j \leq m$. Then we compute $y \cdot g^{-im}$ for each $0 \leq i \leq m$. Notice that both can be done in $O(\sqrt{n})$ time. If we could find a collision $y \cdot g^{-im} = g^j$, then $x = im + j$. To find this collision efficiently we store all g^j (for all $0 \leq j \leq m$) by hash-sets and when we compute $y \cdot g^{-im}$ we simply check if that value is already present in the set. This finds the collision in $O(\sqrt{n})$ time (as we need to check if the value is present in the set only $O(\sqrt{n})$ times). Overall this algorithm uses $O(\sqrt{n})$ space and time.

1.3 Pollard's ρ algorithm

This is a randomized algorithm that works in $O(\sqrt{n})$ expected time and with $O(1)$ space requirement. We descirbe the algorithm for prime modulus. All the operations described in this section work mod p.

The idea is to divide $\{1, \ldots n-1\}$, which are the possible xs into three sets, say, S_0, S_1 and S_2 of roughly equal size. In practice we do it by taking $x \pmod 3$ ie. $x \in S_i$, where $i = x \pmod 3$. We do a random walk starting at $x_0 = g^{a_0} y^{b_0}$ where $a_0 = b_0 = 0$. To generate x_{i+1} from x_i we follow the following rule:

$$x_{i+1} = \begin{cases} x_i^2 \text{ if } x_i \in S_0 \\ yx_i \text{ if } x_i \in S_1 \\ gx_i \text{ if } x_i \in S_2 \end{cases}$$

Corresponding to these x_i's we also maintain the a_i, b_i such that $x_i = g^{a_i} y^{b_i}$. Now suppose we find i and j, $i \neq j$, such that $x_i = x_j$, or $g^{a_i} y^{b_i} = g^{a_j} y^{b_j}$. Then $g^{a_i - a_j} = y^{b_j - b_i} \pmod n$. Substituing $y = g^x$, we get the equation $a_i - a_j = x(b_j - b_i)$. We then solve this using a linear congrence solver to obtain x.

So all we need to so now is to find i and j, $i \neq j$, such that $x_i = x_j$. This is done using the classic Floyd's Hare and Tortoise algorithm (by which the algorithm gets the ρ in its name). Essentially there are two pointers, one that jumps two steps and another that jumps only one. If there are i and j, such that $x_i = x_j$, then these pointers would meet at the same point, giving us i and j.

1.4 Pohlig Hellman algorithm

Let $g^x = y \pmod n$. Given g, y and n where g is the generator $\mathbb{Z}_n^*$ we need to find x. We are also given the factorization of $n-1$ as,

$$n - 1 = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$$

where p's are distinct primes. Notice that $1 \leq y \leq n-1$ and $0 \leq x \leq n-2$ (as $g^0 = g^{n-1} = 1$).

The idea of Pohlig Hellman is to first compute $x \pmod{p_i^{e_i}}$ for each $i \in [k]$. As all $p_i^{e_i}$ are mututally coprime we can recover $x \pmod{n-1}$ by Chinese Remainder Theorem. We show how to compute $x \pmod{p_i^{e_i}}$. We now drop the indices from p_i and e_i for simplicity (and refer to them as p and e).

Consider the expansion of x,

$$x = \sum_{0 \leq i \leq e-1} x_i p^i + sp^e$$

Observe,

$$y^{\frac{n-1}{p}} \equiv g^{\left(x_0 \frac{n-1}{p} + k(n-1)\right)} \equiv g^{x_0 \frac{n-1}{p}} \pmod n$$

where k is some integer. This follows because of g is a generator and so $g^{n-1} \pmod n = 1$. So now we can recover x_0 by using Baby-Step Giant-Step algorithm (as $y^{\frac{n-1}{p}}$ is the x_0th power of $g^{\frac{n-1}{p}} \pmod n$).

This idea extends to find x_i. Assume that we have computed $x_0 \dots x_{i-1}$. Now we compute,

$$y \cdot g^{-(\sum_{0\leq j\leq i-1} x_j p^j)} = g^{(\sum_{i\leq j\leq e-1} x_j p^j)+sp^e}$$

Call this y'. Then notice that (using similar simplification as above),

$$(y')^{\frac{n-1}{p^{(i+1)}}} \equiv g^{x_i \frac{n-1}{p}} \pmod n$$

Again we use Baby-Step Giant-Step algorithm to compute x_j. We continue this to extract each x_i until x_{e-1} using which we create $x \pmod{p^e}$. This algorithm is exactly implemented in the code.

2 Man in the middle attack on the Diffie-Hellman Protocol

Assume that there is an attacker Eve who wants to evesdrop on Alice and Bob's conversation or worse even to modify their messages. Here's how Eve attacks using the 'Man In The Middle' attack if Alice and Bob use Diffie-Hellman Protocol.

Eve obtains the public g and p, the generator and modulus, that Alice and Bob have decided to use. Now according to protocol Alice chooses a and sends g^a and Bob chooses b and sends g^b. Eve herself chooses an e and generates g^e. She sends g^e to Alice and Bob claiming that she is Bob and Alice respectively. She receives both Alice and Bob's g^a and g^b which they had sent.

Now Alice receives g^e and believes that this message is from Bob. So she creates the key g^{ae}. Similarly, Bob receives g^e and believes that this message is from Alice. So he creates the key g^{be}. Eve computes both g^{ae} and g^{be}. Notice now that Eve has both the keys and has Alice and Bob convinced that they are talking to each other.

Now when Alice sends a message encrypted by the key g^{ae}, Eve decrypts it, reads (and possibly modifies) it, reencrypts the message using g^{be} and sends it to Bob who successfully decrypts it. The message recieved is what Eve sent him and not Alice but Bob believes that the message is from Alice. Eve similarly intercepts messages from Bob to Alice. Thus Alice and Bob think that they are communicating with each other without realizing that there is a 'man' in the middle, Eve, who is evesdropping and possibly modifing their messages.

This attack happens because there is no way for Alice and Bob to authenticate that they are talking to the persons they think they are. This vulenrability is fixed by using digital signatures (a mechanism for authentication) and other such protocols.