Week 5 - Programming Assignment [optional: extra credit]

Help

Warning: The hard deadline has passed. You can attempt it, but **you will not get credit for it**. You are welcome to try it as a learning exercise.

☐ **In accordance with the Coursera Honor Code, I (Antonio Diaz Arroyo) certify that the answers here are my own work.**

Question 1

Your goal this week is to write a program to compute discrete log modulo a prime p. Let g be some element in $\mathbb{Z}_p^*$ and suppose you are given h in $\mathbb{Z}_p^*$ such that $h = g^x$ where $1 \le x \le 2^{40}$. Your goal is to find x. More precisely, the input to your program is p, g, h and the output is x.

The trivial algorithm for this problem is to try all 2^{40} possible values of x until the correct one is found, that is until we find an x satisfying $h = g^x$ in $\mathbb{Z}_p$. This requires 2^{40} multiplications. In this project you will implement an algorithm that runs in time roughly $\sqrt{2^{40}} = 2^{20}$ using a meet in the middle attack.

Let $B = 2^{20}$. Since x is less than B^2 we can write the unknown x base B as $x = x_0 B + x_1$ where x_0, x_1 are in the range $[0, B-1]$. Then

$$h = g^x = g^{x_0 B + x_1} = (g^B)^{x_0} \cdot g^{x_1} \quad \text{in } \mathbb{Z}_p.$$

By moving the term g^{x_1} to the other side we obtain

$$\boxed{h/g^{x_1} = (g^B)^{x_0}} \quad \text{in } \mathbb{Z}_p.$$

The variables in this equation are x_0, x_1 and everything else is known: you are given g, h and $B = 2^{20}$. Since the variables x_0 and x_1 are now on different sides of the equation we can find a solution using meet in the middle (Lecture 3.3):

- First build a hash table of all possible values of the left hand side h/g^{x_1} for $x_1 = 0, 1, \ldots, 2^{20}$.

- Then for each value $x_0 = 0, 1, 2, \ldots, 2^{20}$ check if the right hand side $(g^B)^{x_0}$ is in this hash table. If so, then you have found a solution (x_0, x_1) from which you can compute the required x as $x = x_0 B + x_1$.

The overall work is about 2^{20} multiplications to build the table and another 2^{20} lookups in this table.

Now that we have an algorithm, here is the problem to solve:

$$\begin{aligned}
p = \quad & 13407807929942597099574024998205846127479365820592393 \backslash \\
& 377723561443721764030073546976801874298166903427690031 \backslash \\
& 858186486050853753882811946569946433649006084171 \\
g = \quad & 11717829880366207009516117596335367088558084999998952205 \backslash \\
& 59997945906392949973658374667057217647146031292859482967 \backslash \\
& 5428279466566527115212748467589894601965568 \\
h = \quad & 323947510405045044356526437872806578864909752095244 \backslash \\
& 952783479245297198197614329255807385693795855318053 \backslash \\
& 2878928001494706097394108577585732452307673444020333
\end{aligned}$$

Each of these three numbers is about 153 digits. Find x such that $h = g^x$ in $\mathbb{Z}_p$.

To solve this assignment it is best to use an environment that supports multi-precision and modular arithmetic. In Python you could use the gmpy2 or numbthy modules. Both can be used for modular inversion and exponentiation. In C you can use GMP. In Java use a BigInteger class which can perform mod, modPow and modInverse operations.

In accordance with the Coursera Honor Code, I (Antonio Diaz Arroyo) certify that the answers here are my own work.

Submit Answers Save Answers

You cannot submit your work until you agree to the Honor Code. Thanks!