Written problems

1. Evaluate the discrete logarithm $\log_{40} 33$ in $\mathbb{Z}/73\mathbb{Z}$ using the Pohlig-Hellman algorithm, according to the following steps (see the statement of Theorem 2.31 in the textbook for details on the notation). You may use, without proof, the fact that 40 is a primitive root modulo 73.

   (a) Let N be the order of 40 (mod 73). Factor N into prime powers as $N = q_1^{e_1} \cdots q_t^{e_t}$.

   (b) Determine the numbers g_i and h_i for each i from 1 to t inclusive. For each i, what is the order of g_i modulo 73?

   (c) For each i, evaluate the discrete logarithm $y_i = \log_{g_i} h_i$ in $\mathbb{Z}/73\mathbb{Z}$, using a method of your choice.

   (d) Solve the system of congruences $x \equiv y_i \pmod{q_i^{e_i}}$ to obtain the discrete logarithm $x = \log_{40} 33$.

   Note: The remaining three written problems concern material that we won't cover until Friday 3/8.

2. Textbook exercise 3.7 (3.6 in first edition) (RSA example)

3. Textbook exercise 3.8 (3.7 in first edition) (Cracking RSA by factoring)

4. Textbook exercise 3.11 (3.10 in first edition) (a proposed, but ultimately insecure, alternative to RSA)

Programming problems

The first three problems are building blocks towards an implementation of the Pohlig-Hellman algorithm. The last problem assembles these various pieces together; you should paste the functions you write in the first three problems into your code for the fourth and use them in your solution.

1. Write a function `ppFactor(N)` which accepts an integer $N \geq 2$, and returns a list of the prime powers (all powers of different primes) factoring N, in any order. For example, if $N = 12$ the function should return either `[4,3]` or `[3,4]`. The integer N may be quite large (up to 1024 bits), but you may assume that all of the prime-power factors are 16 bits or smaller.

   Suggested approach: There are many ways to do this, and certainly many more efficient than what I'm about to describe, but here is one relatively quick-to-implement approach. Write a `for` loop to iterate through all numbers p from 2 to 2^{16}. For each number, check whether it divides N. If so, divide N by p repeatedly until it is no longer divisible by p (and replace N by the new value), then add the appropirate power of p to the list you will eventually return. As long as you shrink N as you go, you will never find that $p \mid N$ unless p is in fact prime, since any smaller factor would have already been found to divide N.

2. Write function `bsgsBoundedOrder(g,h,p,q)` to solve the discrete logarithm function $g^x \equiv h \pmod{p}$ under the assumption that the order of g is at most q. You may assume that p is a prime number, but unlike the previous set it will be quite large (256 bits). The integer q will be various sizes, up to 40 bits. Any correct solution x will be marked correct, even if it is not the smallest possible solution.

Note: The time limit is set to 1 second on Gradescope as usual, which may be slightly tight on the largest cases. Let me know if you are running into the time limit so that I can suggest improvements. I may extend the time limit if it seems that generally efficient implementations are not quite meeting it.

3. Write a function `crtList(ls)` that takes a list `ls` of pairs (a_i, m_i) of integers, with any two of the values m_i relatively prime, and returns a pair (a, m) such that the system of congruences $x \equiv a_i \pmod{m_i}$ is equivalent the single congruence $x \equiv a \pmod{m}$, and $0 \leq a < m$ (i.e. a is reduced modulo m).

   For example, `crtList( [(2,3), (3,5), (0,2)] )` should return $(8, 30)$, since the system of three congruences $x \equiv 2 \pmod{3}, x \equiv 3 \pmod{5}, x \equiv 0 \pmod{2}$ is equivalent to the single congruence $x \equiv 8 \pmod{30}$.

   The integer a should be reduced modulo m, i.e. $0 \leq a < m$. The moduli m_i will be integers up to 256 bits in length, and the list will contain up to 128 entries.

4. Implement the Pohlig-Hellman algorithm. That is, write a funciton `ph(g,h,p)` that solves the discrete logarithm problem $g^x \equiv h \pmod{p}$ under the assumption that p is a "weak" prime, in the sense that $p - 1$ factors into small prime factors. More specifically: you may assume that $p - 1$ factors into prime powers, all 16 bits or smaller, but p will be 64 bits in length.