Homework 4

Due Thursday, September 30

Implementation Part

1. Implement the naïve algorithm to solve the discrete log problem. In particular, define a function `naiveDLP(g,h,p)` satisfying the following:

Input	Output
A prime p	$\log_g(h)$ if it exists
An element $g \in \mathbb{F}_p^*$	
An element $h \in \mathbb{F}_p^*$	

   It should work by computing $g \mod p, g^2 \mod p, g^3 \mod p, \cdots$ in a loop until it finds h, then returning the relevant power.

2. Implement the baby steps-giant steps algorithm to solve the DLP for $\mathbb{F}_p^*$. In particular, define a function `babyGiant(g,h,p,N=p-1)` satisfying the following:

Input	Output
A prime p	$\log_g(h)$ if it exists
An element $g \in \mathbb{F}_p^*$	
An element $h \in \mathbb{F}_p^*$	
The order N of g.	

   If no order N is given your algorithm should set $N = p - 1$.
   Hint

   - Recall that the hardest part of this algorithm wasn't making the lists of baby steps and giant steps, but of searching for and finding an element that is in both lists. If you do this by just comparing each element of the list one by one, then this will take $\mathcal{O}(\sqrt{N}^2) = \mathcal{O}(N)$ steps, and you won't have saved any time at all over the brute force attack of the discrete log. Instead, we will implement a *hash table*. In python this type of data structure is a *set*. You could initiate this like `babysteps = set()`, and then add an element x using `babysteps.add(x)`. The great thing about sets is you can do something like `x in babysteps` which will return true if x is in the set and false otherwise, and since x is paired with an index it can do this in $\mathcal{O}(1)$ time! The downside is hash tables aren't ordered, so you need some other way remember the discrete logs of your elements. One way is to also have a babysteps list alongside the set, and each time you generate a giant step, see if it is in the set (using `x in babystep` which is $\mathcal{O}(1)$), if it is then run through the babystep list to see which position the match is. This is better because you are only running through the babysteps list once (rather than $\sqrt{N}$ times with the naive list comparison).

3. Let's do some testing!

   (a) Let $p = 113$. Use the `naiveDLP` and `babyGiant` to compute $\log_3 19$ modulo p.

(b) Let $p = 1073741827$. Use `naiveDLP` and `babyGiant` to compute $\log_2 54382$. (Note: I got both of them to run, but one was instant and the other took nearly four minutes! This should demonstrate that although `babyGiant` is exponential, it is still a significant improvement over the naïve algorithm)).

(c) Let $p = 30235367134636331149$. Try using `babyGiant` to compute the discrete log $\log_6 3295$ modulo p. Did it run?

Written Part

4. In defining `babyGiant` didn't always know the exact order of g, and couldn't necessarily check it directly. Therefore the exact proofs of correctness of these algorithms in class need to be modified to make sure our algorithms work. Let's do this here.

   (a) If `babyGiant` doesn't receive an order as an input it defaults to $|g| = p-1$. Explain why this is just assuming that g is a primitive root. Suppose g is not a primitive root but `babyGiant` still assumes $N = p-1$. Prove that `babyGiant` returns the correct logarithm.

   (b) Even if the order given as input is not entirely correct (but say a multiple of the actual order), the algorithms work. Still, there are drawbacks to not having the correct order. What are they? (e.g., if using $p-1$ in `babyGiant` always works, why don't we just always set $N = p - 1$?).

   (c) Baby steps-giant steps as we introduced in class took $\mathcal{O}(\sqrt{N}\log N)$ because of how long it took to find an element in both the baby steps list and the giant steps list. In question 2 we implemented a hash table instead of a list, and checking if an element is in a hash table takes $\mathcal{O}(1)$ steps (which doesn't depend on the size of the table!). Use this fact to prove that your implementation runs in $\mathcal{O}(\sqrt{N})$ steps instead.

In class we showed that an Elgamal oracle can solve the Diffie-Hellman problem. Let's show the other direction, and conclude that Elgamal and Diffie-Hellman are equally difficult.

5. Suppose you have access to an oracle who can solve the Diffie-Hellman problem. That is, for any prime p, given g^a and $g^b \mod p$, the oracle can tell you $g^{ab} \mod p$. Show that you can use this oracle to crack the Elgamal Public Key Cryptosystem. Precisely, suppose Alice publishes a prime p, an element $g \in \mathbb{F}_p^*$, and a public key A, and Bob sends the cipher text (c_1, c_2). Consult with the oracle to find the message Bob sent.

Problem 2.10 in [HPS] gives an example of a cryptosystem where Alice and Bob need to send two rounds of messages back and forth to communicate. We reproduce it here. It might be fun to follow along on cocalc!

6. Alice and Bob decide on a prime $p = 32611$. The rest is secret. But any information crossing the middle channel (via the arrows) should be assumed to be intercepted by Eve.

	Alice	Eve	Bob
1.	Alice has message $m = 11111$		
2.	Alice chooses random $a = 3589$		
3.	Alice computes $u = m^a \mod p = 15950$	$\longrightarrow$	Bob recieves u
4.			Bob chooses random $b = 4037$.
5.	Alice recieves v	$\longleftarrow$	Bob Computes $v = u^b \mod p = 15422$
6.	Alice knows $a' = 15619$		
7.	Alice computes $w = v^{a'} \mod p = 27257$	$\longrightarrow$	Bob recieves w.
8.			Bb knows $b' = 31883$
9.			Bob computes $w^{b'} \mod p = 11111$. That's m!

(a) Notice how Alice knows a second exponent $a' = 15619$ in step 6. Where does this number come from and how does this relate to $a = 3589$ from step 2? Similarly how do Bob's exponents $b = 4037$ and $b' = 31883$ relate? Use this information to explain how the algorithm works.

(b) Formulate a general version of this algorithm using variables and show that it works in general.

(c) Can a solution to the DLP break this cryptosystem? Justify your answer.

(d) Can a solution to the DHP break this cryptosystem? Justify your answer.

7. In Homework 2 we studied square roots mod p. Let's use this to study square roots modulo p^e for some positive exponent e. Let p be a prime not equal to 2, and let b be an integer not divisible by p. Suppose further that b has a square root modulo p, i.e., the congruence:

$$x^2 \equiv b \mod p,$$

has a solution.

(a) The following fact may be handy. Show that $(a+b)^p = a^p + b^p + pab(\text{stuff})$. Conclude that $(a+b)^p = a^p + b^p \mod p$. (This is often called the *freshman's dream* by cynical math professors who have seen a similar formula on too many calculus exams).

(b) Show that for every exponent $e \geq 1$, b has a square root module p^e. That is, the congruence

$$x^2 \equiv b \mod p^e$$

has a solution. (**Hint:** Use induction on e, finding a solution modulo p^{e+1} by modifying the solution modulo p^e.)

(c) Let $x = \alpha$ be a square root of b modulo p. Prove that in part (a) we can find a square root root β of b mod p^e such that $\alpha \equiv \beta \mod p$.

(d) Suppose β, β' are two square roots of b mod p^e, and further that they are both equivalent to $\alpha \mod p$ as in part (b). Show that $\beta \equiv \beta' \mod p^e$.

(e) Conclude that the congruence $x^2 \equiv b \mod p^e$ has either 2 solutions or 0 solutions. (Use HW2 Probem 8).

8. Let G and H be groups, and denote their multiplication rules by $*_G$ and $*_H$ respectively. A function $\varphi : G \to H$ is called a *homomorphism* if for all $a, b \in G$:

$$\varphi(a *_G b) = \varphi(a) *_H \varphi(g).$$

(a) Recall that the identiy of a group $e \in G$ is an element such that for every $g \in G$,$e * g = g * e = g$. Show that the identity element of a group G is unique.

(b) Let e_G be the identity of G and e_H the identity of H. Show that if φ is a homomorphism then $\varphi(e_G) = e_H$

(c) Let $g \in G$. Recall that an inverse to g is an element g^{-1} such that $g * g^{-1} = g^{-1} * g = 1$. Show that g^{-1} is unique.

(d) Show that $\varphi(g^{-1}) = \varphi(g)^{-1}$ for all $g \in G$.

(e) Suppose $\varphi : G \to H$ is a homomorphism and is bijective. Show that the inverse $\varphi^{-1} : H \to G$ is a homomorphism as well. Such a map is called an *isomorphism*. We then call G and H *isomorphic*. This is a notion of being the *same* group in some sense.

(f) Prove that $\mathbb{F}_p^*$ and $\mathbb{Z}/(p-1)\mathbb{Z}$ are isomorphic. (You are welcome to cite work already done in past homeworks).