Written problems:

1. Every day, Alice and Bob perform Diffie-Hellman key exchange, using public parameters p and g. Unfortunately, Alice and Bob do not randomize their secret numbers well.

   On Monday, Alice sends Bob the number A, Bob sends Alice the number B, and they establish a shared secret S. On Tuesday, Alice sends A', Bob sends B', and they establish a shared secret S'. Eve examines the numbers A, B, A', B', and discovers the following facts (resulting from poor random number generation).

   $$\begin{aligned} A' &\equiv Ag \pmod{p} \\ B' &= B^2 \pmod{p} \end{aligned}$$

   Show that if Eve manages to learn Monday's shared secret S, then she can quickly determine Tuesday's shared secret S' as well.

   More precisely, describe a procedure Eve could follow to efficiently compute the number S'. You may assume that Eve knows p, g, A, B, A', B', and S. Do not assume that Eve knows (or can learn) Alice and Bob's secret numbers a or b. You do not need to write your solution as a program, but be clear about any algorithms Eve will require in her computation, and explain why your method will work.

2. Textbook exercise 2.10, parts (a), (b), and (c). (On a three-transmission cryptosystem)

3. Write a function that reduces the problem of breaking the cryptosystem in the previous problem to the Diffie-Hellman problem. That is, assumed that you have an efficient function `dhOracle(p,g,A,B)` that extracts the shared secret from the public parameters and transmitted values in Diffie-Hellman, and use it to write a function `analyze210(p,u,v,w)` that would efficiently find the plaintext m in the system from exercise 2.10. It is fine to write the code by hand. (Obviously I cannot autograde it because I am unwilling to confirm or deny that I have a Diffie-Hellman oracle at this time.)

   Hint. The reduction is a little tricky to find; think about all the different ways you could match up the information you know with the g, A, B from Diffie-Hellman.

4. Let $p \geq 3$ be a prime number, and let g be a primitive root modulo p. For any $h \in (\mathbb{Z}/p\mathbb{Z})^\times$, denote by $\log_g h$ the solution $x \in \{0, 1, \cdots, p-2\}$ to the congruence $g^x \equiv h \pmod{p}$.

   (a) Prove that this notation is well-defined, i.e. prove that for each choice of h there is a *unique* $x \in \{0, 1, \cdots, p-2\}$ solving this discrete logarithm problem.

   (b) Prove that $\log_g h$ is even if and only if h has a squre root modulo p (a "square root modulo p" is a solution to the congruence $x^2 \equiv h \pmod{p}$).

   (c) Prove that for all $h_1, h_2 \in (\mathbb{Z}/p\mathbb{Z})^\times$,

   $$\log_g(h_1 h_2) \equiv \log_g h_1 + \log_g h_2 \pmod{p-1}.$$

5. Use the babystep-giantstep algorithm to solve each of the following discrete logarithm problems. Show your calculations, e.g. in the form of the table on page 83 of the textbook.

   (a) $10^x \equiv 13 \pmod{17}$

   (b) $15^x \equiv 16 \pmod{37}$

   (c) $5^x \equiv 72 \pmod{97}$

Programming problems:

1. Suppose that you are given two integers m_1, m_2 with $\gcd(m_1, m_2) = 1$, and two integers a_1, a_2. Write a function `crt(a1,m1,a2,m2)` that efficiently determines the unique integer x such that

$$\begin{aligned} x &\equiv a_1 \pmod{m_1}, \\ x &\equiv a_2 \pmod{m_2}, \end{aligned}$$

   and $0 \leq x < m_1 m_2$. (There are several efficient ways to do this; one is to follow the strategy of exercise 1.24(a), i.e. PSet 3 problem 4.)

2. Implement the Babystep-Giantstep algorithm, efficiently enough to solve the discrete logarithm problem for primes up to 40 bits in length. That is, write a function `bsgs(g,h,p)` that finds an integer x such that $g^x \equiv h \pmod{p}$. You may assume that p is prime, $1 \leq g, h < p$, and that a solution x exists.

   The test bank on Gradescope will be identical to the test bank for `disclog` on Problem Set 2, but now your code must solve all 40 cases for full credit.

   (It is fine to implement any algorithm you can devise to solve the discrete logarithm problem, but BSGS is probably the easiest to implement based on what we've discussed in class.)

   > **Note** If implemented a certain way, your solution to this problem can be almost exactly identical to your solution to the following problem, and indeed can consist of a one-line call to the function you write for the following problem.

3. Write function `bsgsBoundedOrder(g,h,p,N)` to solve the discrete logarithm function $g^x \equiv h \pmod{p}$ under the assumption that the order of g is at most N. You may assume that p is a prime number, but unlike the previous problem it will be quite large (256 bits). The integer N 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.