Refer to the second page of the Course Survey for instructions on submitting written work on Gradescope.

Written problems

1. Textbook exercise 3.13 (3.12 in the 1st edition) (Danger of repeating the same modulus with different encrypting exponents)

2. In this problem, you will empirically investigate the Prime Number Theorem, and some of its variations.

   (a) Using any method you wish, determine the number of primes exactly n bits long for each of the following values of n: $4, 8, 12, 16, 20$. (Note: probably the easiest way to do this is to use Wolfram Alpha; it can correctly answer questions of the form "number of primes between a and b". You can also use your Miller-Rabin code, or implement the Sieve of Eratosthenes, or use any other method you can think of to count primes).

   (b) The simplest of of the Prime Number Theorem says that the number of primes less than or equal to n is approximately $\frac{n}{\ln(n)}$. Use this approximation to give a formula estimating the number of primes in a closed interval $[a, b]$ (where $a, b \in \mathbb{Z}$), and determine the number of exactly n-bit primes this predicts for the five values of n in part (a).

   (c) Another version of the Prime Number Theorem says that the number of primes in a closed interval $[a, b]$ is approximated by the sum

   $$\sum_{m=a}^{b} \frac{1}{\ln(m)}.$$

   (Informally, you can pretend that "the probability that m is prime is $\frac{1}{\ln(m)}$." This is nonsense if taken literally, but it is a useful fiction: the sum above would then be the expected value of the number of primes between a and b inclusive, since it the sum, for each m in that interval, of the probability that m is prime.)

   Compute the number of exactly n-bit primes predicted by this estimate for the same five values of n. (You can compute this however you like; it can, for example, be done with a few lines of Python code, or using Wolfram Alpha. State in your write-up exactly how you computed it.)

   (d) Summarize the numbers you found in parts (a) through (c) in a table. Discuss the relative accuracy of the two different estiamtes in parts (b) and (c).

   Note: if you're interested (not part of the course), you can try to prove that the two estimates, despite looking differnet, are the same asymptotically (that is, the limit of their ratios converges to 1 as $b \to \infty$). This can be done by approximating the sum with an integral, applying integration by parts, and finding some bounds on the result.

   (e) Modify the approximation methods from parts (b) and (c) to instead approximate the number of primes $p \equiv 1 \pmod 5$ of each of the five bit lengths, and compute the exact number of such primes (Wolfram Alpha can do this as well; ask me if you're having trouble getting it to understand the question). Construct a table like in part (c) to compare the true value and the two estimates. Briefly discuss any observations you can make from this table.

3. (Solve the Miller-Rabin programming problem first, so that you have code that you can use to count Miller-Rabin witnesses) For each integer n between 1,000,000 and 1,000,009 inclusive, determine the proportion of the numbers from 1 to $n-1$ inclusive that are Miller-Rabin witnesses. Which of these numbers are prime? (The figures you obtain should convince you that the 75% figure from Rabin's theorem is rather conservative, and explains why most people are not worried about using only a few Miller-Rabin trials to test primality).

4. Suppose that p is a large prime (e.g. 1024 bits), g is a primitive root modulo p, and Alice has an Elgamal public key A corresponding to a private key a (that is, $A \equiv g^a \pmod{p}$, and Alice knows the number a). Bob does not believe that Alice actually knows the private key a corresponding to A, so she asks her to solve the following challenge to prove it. Bob will give Alice a positive integer d of his choosing. Alice must return (in a reasonable amount of time) *two* integers b, c such that

$$g^b \cdot b^c \equiv A^d \pmod{p}.$$

If she succeeds, Bob will be convinved that Alice really does know her private key.

(a) Describe a procedure that Alice can use to solve Bob's challenge efficiently.
Hint. Choose an integer e at random, and choose b to be $g^e \pmod{p}$. Then find a choice of c.

(b) Explain briefly why Bob should be convinced that Eve (or anyone else who doesn't know the private key) would not be able to carry out the procedure you describe in part (a).

Note. This exercise prefigures the basic idea behind Elgamal "digital signatures," which we will discuss soon. You can solve this problem without knowing anything about signatures, however.

Programming problems

1. In one of the written problems (3.13 in the 2nd edition, 3.12 in the 1st), you saw that it is unsafe to use the same modulus N in two different RSA public keys. In this problem, you will implement the algorithm that Eve could use to exploit that situation, in a more general context.

Suppose that you know a modulus N, two relatively prime integers e, f, and two powers $m^e \pmod{N}$ and $m^f \pmod{N}$ of an unknown integer m. You may assume that m is a unit modulo N. Write a function `mFromPowers(N,e,f,me,mf)` that computes and returns the unknown integer m (you should return m reduced modulo N, i.e. $0 \leq m < N$). The integer N will be 1000 bits long in the largest test cases, but a naive approach will earn partial credit.

Note: this algorithm has peaceful uses as well. In fact, you can think of RSA decryption as a special case: when Alice receives an RSA message, she knows $m^e \pmod{N}$ and $m^f \pmod{N}$, where $f = (p-1)(q-1)$ ($m^f \equiv 1 \pmod{N}$ in this case). Since $\gcd(e, (p-1)(q-1)) = 1$, this function would be able to decipher the message. Take some time to think about why only Alice can do this, and not Eve.

2. Implement the Miller-Rabin primality test (or another primality test of your choice): write a function `isPrime(n)` that returns `True` or `False` according to whether or not n is prime. The starter code will also define a functi`checkList` that applies your function to a list of integers; you do not need to modify that part. Each test case will give your function ten integers of the same size; to pass the test case your function must give the correct answer for all ten.

3. Write a function `makeQP(qbits,pbits)` that generates two primes numbers q and p such that q is exactly `qbits` bits long, p is exactly `pbits` bits long, and $p \equiv 1 \pmod{q}$. Recall that p is "exactly n bits long" means that $2^{n-1} \leq p < 2^n$.

   Take a moment to remind yourself why it is useful to construct primes this way, even if it's only the prime p that you want (e.g. for Diffie-Hellman parameters).

4. Alice decides that she wants to receive messages using a non-standard variant of RSA. Like in the usual RSA, she will choose a public key N, e, where N is a number whose factorization she knows, and $\gcd(e, \phi(N)) = 1$. In this case, she will take $N = pqr$, where p, q, r are distinct primes. To encrypt a message m for Alice ($0 \leq m < N$), Bob computes $c \equiv m^e \pmod{N}$. Write a function `rsaThreePrimes(p,q,r,e,c)` to do the following: given the three primes p, q, r, the number e, and the ciphertext c sent by Bob, recover the original plaintext m.

   Note. While this setup is perfectly functional, in practice it is more efficient to use products of two primes, hence that is the standard. I encourage you to think about why it is more efficient to use only two primes.