Written problems

1. Textbook exercise 3.10 (finding a deciphering exponent can help factor a modulus)
2. Textbook exercise 4.2 (RSA signature examples)
3. Textbook exercise 4.6 (ElGamal signature examples)
4. Textbook exercise 4.7 (ElGamal "blind signatures")

Programming problems

1. We've discussed in class the need for choosing primes p such that $p-1$ has a large prime factor. It is also considered a good idea to ensure that $p+1$ also has a large prime factor (for reasons we won't discuss). In this problem, you will write a function `strongPrime(qbits,pbits)` to construct such a prime. You will be given integers `qbits` and `pbits`, and should return 3 prime numbers q_1, q_2, p such that both q_1 and q_2 are at least `qbits` bits long, p is exactly `pbits` bits long, and such that $q_1 \mid (p-1)$ and $q_2 \mid (p+1)$. As with last week's `makeQP` problem, I recommend choosing the subordinate primes q_1, q_2 first, and using these to narrow the search for the last prime p.

2. This problem concerns a modular arithmetic problem that we have not yet considered, but which is important to the last programming problem. You will be given integers m, b, and N, and your goal is to solve the congruence $mx \equiv b \pmod{N}$ for x. When m is relatively prime to N, this is accomplished by multiplying by the inverse of m; you should figure out how to solve such a congruence in cases where m may have common factors with N. It is possible that no solutions exist. If solutions exist, they can all be described in a single congruence $x \equiv r \pmod{M}$, where r, M are integers and M is not necessarily the same as N. Write a function `linearCong(m,b,N)` that either returns `None` if no solutions exist, or returns a pair `(r,M)` describing the general solution if solutions do exist.

   Hint: re-write the original congruence as an equation with one more variable, and try to convert it to a congruence (possibly with a different modulus) in which the coefficient of x is invertible.

3. When using ElGamal digital signatures, it is essential that Samantha always generates her ephemeral key at random (much like in ElGamal encryption). In this problem, you will study why it is particularly dangerous to use the same ephemeral key twice. You will be given the public ElGamal parameters `p,g`, Alice's public key `A`, two documents `d1,d2`, and valid signatures `(s11,s12),(s21,s22)` for the two documents (respectively). The two signatures were generated using the same ephemeral key. Write a function `extractKey(p,g,A,d1,s11,s12,d2,s21,s22)` that extracts and returns Alice's private key `a` from this information.

   Hint: if you carefully manipulate the two congruences Samantha used to sign the documents, you can derive a congruence of the form $ma \equiv b \pmod{p-1}$, where m and b are values you can compute and a is the private key that you are trying to find. Unfortunately, it is possible that m is not invertible modulo $p-1$. You can use the solution to the previous problem to "solve" this congruence to obtain a congruence that may not determine a uniquely; you'll need to figure out how to get from here to the specific value of a.