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

Written problems

1. Textbook exercise 4.10 (7.9 in 1st edition) (DSA verification examples)
2. Consider the following implementation of one trial of Pollard's algorithm.

```
def pollardTrial(N):
        a = random.randint(1,N-1)
        # Check first whether a is a unit. If not, you have a factor.
        if math.gcd(a,N) != 1:
                return math.gcd(a,N)
        j = 2
        while math.gcd(a-1,N) == 1:
                a = pow(a,j,N)
                j += 1
        return math.gcd(a-1,N)
```

   (a) Suppose that this function is called on an input $N = pq$, a product of two distinct primes. Prove that in principle (i.e. given an unbounded amount of time), this function will always return some factor of N other than 1. Under what circumstances will it return N, rather than a proper factor?

   (b) Suppose that this function is called on $N = pq$, where both $p - 1$ and $q - 1$ have at least one prime factor greater than 2^{256}. Estimate how large you expect j to grow before this function will return an answer, and explain why. The result will depend on the random value of a this is chosen; try to justify why the estimate you give will be correct with very high probability.

3. Textbook exercise 6.1 (5.1 in 1st edition) (Elliptic curve arithmetic over $\mathbb{R}$)
4. Textbook exercise 6.5 (5.5 in 1st edition), parts (a) and (b) (Elliptic curve arithmetic over $\mathbb{F}_p$)

   Hint. You can save some time by making two lists in advance: values of y^2 for various y and values of $x^3 + Ax + B$ for various values of x, then checking for numbers occurring in both lists)

5. Textbook exercise 6.9 (5.9 in 1st edition) (listing all solutions n to an equation $Q = n \cdot P$ on an elliptic curve).

Programming problems

1. Write a function `verifyDSA(p,q,g,A,d,s1,s2)` that verifies DSA signatures. Here, p, q, g are public parameters, A is the public (verification) key, d is the document, and (s_1, s_2) is the signature. The function should return `True` or `False`.
2. Suppose that Samantha and Victor are using a variant of Elgamal signatures, in which the verification congruence that Victor will use is $s_1^{s_1} \cdot g^{s_2} \equiv A^d \pmod{p}$. Write a function `signElGamalVariation(p,g,a,d)`, which produces a valid signature in this system, given the public parameters p, g, Samantha's secret signing key a, and a document d.

3. Devise a method to create "blind forgeries" for a given DSA public key. That is, write a function `dsaBlind(p,q,g,A)` given p, g and A as in DSA, generate integers (d, s_1, s_2) such that (s_1, s_2) is a valid signature for d for the verification key A. You will likely want to adapt the strategy from one of last weeek's problems from Elgamal to DSA. Your method should be non-deterministic; the grading script will give the same test case multiple times to check that the same answer is not returned each time.

4. Write a function `ecAdd(P,Q,A,B,p)` to compute the sum $P \oplus Q$ of two points on the Elliptic Curve over $\mathbb{F}_p$ defined by $Y^2 \equiv X^3 + AX + B \pmod{p}$. You may assume that P and Q are both valid points on the curve[1]. The points P and Q will be either pairs (x, y) of elements of $\mathbb{Z}/p\mathbb{Z}$, or the integer `0` (as a stand-in for the point $\mathcal{O}$ at infinity), and the function should return the result in the same format.

[1] Though of course if you were using this code in real life, you should add some error handling that checks this.