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

Written problems

1. This problem explores the reasons why the primes p and q is DSA can be chosen to have somewhat different sizes.

   Suppose that p, q, g are DSA public parameters (i.e. p, q are primes, and g has order q modulo p), and $A \equiv g^a \pmod{p}$ is Samantha's public (verification) key, while a is her private (signing) key. As we discussed in class, there are two main sorts of algorithms that Eve might use to extract a from A: collision algorithms (whose runtime depends on q), and the number field sieve (whose runtime depends on p). For simplicity, assume that Eve has a collision algorithm that can extract a in $\sqrt{q}$ steps, and an implementation of the number field sieve (a state of the art DLP algorithm; you do not need to know any details about it, but the textbook has a good overview) that can extract a in $e^{2(\ln p)^{1/3}(\ln \ln p)^{2/3}}$ steps (the true runtimes would involve a constant factor that would depend on implementation, and various other factors depending on the cost of arithmetic modulo p and of finding collisions).

   (a) Suppose that Samantha is confident that her private key will be safe as long as Eve does not have time to perform more than 2^{64} steps in either algorithm. How many bits long should she choose p to be? How many bits long should q be?

   (b) What if she instead wants to be safe as long as Eve doesn't have time for 2^{128} steps?

   (c) The NSA's recommendation for "Top Secret" government communications is to use 3072 bit values of p, and 384 bit values of q. How does this compare to your answers above? If the difference is significant, what might explain the discrepancy?

   For parts (a) and (b), it is sufficient to write a short script to find the minimum safe numbers of bits by trial and error (there are more efficient ways, of course).

2. Textbook exercise 6.15 (5.14 in 1st edition). (A multi-transmission cryptosystem with Elliptic Curves)

3. Textbook exercise 6.16 (5.15 in 1st edition). (A more concise way to send EC points)

4. (This problem concerns ECDSA, which we will not cover until Monday's class. You may want to either wait until next week to work on it, or read the description of ECDSA in the textbook first.)

   Samantha and Victor agree to the following digital signature scheme. The public parameters and key creation are identical to those of ECDSA. The verification procedure is different: to decide whether (s_1, s_2) is a valid signature for a document d, Victor computes

   $$\begin{aligned} w_1 &\equiv s_1^{-1} d \pmod{q} \\ w_2 &\equiv s_1^{-1} s_2 \pmod{q}, \end{aligned}$$

   then he checks to see whether or not

   $$x(w_1 G \oplus w_2 V)\%q = s_1.$$

   If so, he regards (s_1, s_2) as a valid signature for d.

Determine a signing procedure for this signature scheme, i.e. a procedure that Samantha can follow to produce a valid signature on a specific document. Your procedure should be non-deterministic, like in ECDSA.

Programming problems

1. Write a function `ecMult(n,P,A,B,p)` that computes an integer multiple $n \cdot P$ of a point P on an elliptic curve $Y^2 \equiv X^3 + AX + B \pmod{p}$. Points will be formatted (x, y), with $0 \leq x, y < p$, while the point at infinity should be denoted simply as 0. Your code will need to be able to scale to very large values of n; I suggest adapting the fast-powering algorithm from modular arithmetic to elliptic curves.

2. Write a function `ecDLP(P,Q,A,B,p,q)` to solve the elliptic curve discrete logarithm problem, in cases where the prime p is up to 28 bits. Here, P, Q are points on the curve $Y^2 \equiv X^3 + AX + B \pmod{p}$, and you are given, for convenience, the number q of points on the curve (which you may assume to be prime). The function should return the minimum nonnegative n such that $n \cdot P = Q$. Note that if you find any such solution n', then you can find the minimum solution by computing $n' \pmod{q}$.

   A naive trial-and-error approach will earn partial credit, but to solve all test cases I recommend adapting the BSGS algorithm to the setting of elliptic curves. Note: one issue you may encounter is that it is not possible to place a "list" (e.g. `[2,3]`) into a Python `dict`. To resolve this, make sure all of your elliptic curve points (besides $\mathcal{O}$) are represented as "tuples" instead (e.g. `(2,3)`, with parentheses instead of brackets).

3. (You may want to wait until we discuss ECDSA on Monday to work on this problem) Write a function `verifyECDSA(V,d,s1,s2)` that determines whether (s_1, s_2) is a valid ECDSA signature for the document d and the public (verification) key V (see p. 461 in the 1st edition or p. 322 in the 2nd edition for notation regarding ECDSA), using Elliptic curve "P-384." You should look up the specifics of curve P-384, e.g. by finding the relevant information in the standards document at the link below. You will also need to look up how to convert hexadecimal strings to integers in Python.

   `https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf`

   You may enjoy looking through the standards document to see what other sorts of information it contains.