

**Written problems:**

1. Textbook exercise 4.6 (ElGamal signature examples)
2. Textbook exercise 4.10 (DSA verification examples)
3. Textbook exercise 4.7 (ElGamal “blind signatures”)
4. This problem explores the reasons why the primes  $p$  and  $q$  in 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).

**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. 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  $d_1, d_2$ , and valid signatures  $(s_{11}, s_{12}), (s_{21}, s_{22})$  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 an earlier 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  $a$ .