Problem Set 3

This problem set is due **at 10:00pm** on **March 24, 2019**. Total Points: **110**

- Remember that the problem set must be submitted on Gradescope. If you haven't done so already, please sign up for CS 302 on Gradescope, with the entry code 9WB58J to submit this assignment.

  We suggest that you perform a trial submission with a zip file prior to the deadline to make sure that everything works for you – you can overwrite that submission with a new one up to the deadline.

- We require that written solutions are submitted as a PDF file, **typeset on LaTeX**, using the **updated template** available on Piazza. You must **show your work** for written solutions. Each problem should start on a new page.

- We will occasionally ask you to "give an algorithm" to solve a problem. Your write-up should take the form of a short essay. Start by defining the problem you are solving and stating what your results are. Then provide: (a) a description of the algorithm in English and, if helpful, pseudo-code; (b) a proof (or proof sketch) for the correctness of the algorithm; and (c) an analysis of the running time.

- We will give full credit **only** for correct solutions that are described clearly and convincingly.

- Note to all students: Make sure you go over all the number theory (e.g., lecture notes 05-01 and 05-02) we have discussed as well as the review material (lecture notes 8 and 8-02) before starting this. **In all cases, show your work and justify your answers.**

Problem 1-1. Zero divisors [10 points]

An element $a \in Z_n - \{0\}$ is said to be a zero divisor modulo n if $ab \equiv 0 \mod n$ for some $b \in Z_n - \{0\}$.

(a) Explain why there are no zero divisors in Z_p when p is prime.

(b) Find a zero divisor in Z_{39}.

(c) What is the value of the first element to repeat in the sequence: $(5^1 \mod 39), (5^2 \mod 39), (5^3 \mod 39), (5^4 \mod 39), \ldots$?

(d) Do you think it is possible to find a non-zero number $x \in Z_{39}$ and a number $k \geq 0$ such that $x^k \equiv 0(\mod 39)$? Why or why not? Would your answer change for some RSA modulus n other than 39?

(e) Suppose n is not required to be an RSA modulus. Can you find numbers n, x, k such that $x \not\equiv 0$ but $x^k \equiv 0(\mod n)$?

Problem 1-2. Greatest common divisor [5 points]

The definition of the *greatest common divisor* can be extended to a sequence of numbers $a_1, a_2, \ldots, a_k$, not all of which are zero; namely, it is the largest integer $d \geq 1$ such that $d|a_j$ for all $j = 1, 2, \ldots, k$.

Describe an efficient algorithm for computing $gcd(a_1, a_2, \ldots, a_k)$, and explain why it computes the correct answer.

Problem 1-3. Euler's totient function [5 points]

Compute $\phi(2200)$.

Problem 1-4. Euler's theorem [5 points]

Compute $3^{591207} \mod 2200$.

Problem 1-5. Extended Euclidean algorithm [5 points]

Use the extended Euclidean algorithm to solve the Diaphantine equation: $539x - 1387y = 1$

Show the resulting table of triples as in slide 15 of lecture 8 (review) notes. [Note: You may write a program to produce the table if you wish, but these numbers are small enough to make it quite feasible to carry out the computation by hand or with the aid of a phone/calculator.]

Problem 1-6. RSA [5 points]

Factor $n = 2491$ if $\phi(n) = 2392$.

Problem 1-7. Primitive Roots [5 points]

Find a primitive root g for p = 761. (Use the Lucas test to make sure you have a primitive root!)

Similarly, find a non-trivial $g \in Z_{761}^*$ (g is not 1 or p-1) which fails to be a primitive root of p (i.e., it fails the Lucas test!)

Problem 1-8. Factoring the RSA modulus knowing the public and private keys [10 points]

Bob's public RSA key is $n = 1501, e = 323$.

Eve manages to learn that his decryption key is $d = 539$.

Implement the randomized factoring algorithm shown in slide 49 of lecture note 08-02 (review). Make sure you understand what is happening at each step! In the discussion on page 50, we have deliberately not written why the probability is "at least 0.5" – think about why. (This is just a suggestion for you to ponder; you need not write this in your solution.)

Use your program to factor n. Once you have the factorization of n, compute $\phi(n)$, and check your answer by verifying that $ed \equiv 1(\mod \phi(n))$.

Note:It is recommended to use the following algorithms for RSA and ElGamal. You may use whatever you wish, but performance will be evaluated, so your choice of algorithm should be *at least* as efficient as the following implementations.

- Miller-Rabin Algorithm for prime generation
- Square-and-multiply algorithm for modular exponentiation
- Extended Euclidean algorithm for modular inverse computation
- Lucas test for finding primitve root. Note that we require a special form prime $q = 2p + 1$ for ElGamal.

Problem 1-9. Short man before a French article and group of males drinking English beverage is template-less? (9) [30 points]

Make sure you look through the various code snippets (e.g., for exponentiation). As usual, your code must be accompanied by a clear README. You may, of course, use any language you desire, as long as you indicate clearly how I can run your program in the README. $n = pq$, your exponent, must have a minimum *bit length* of 1024.

(a) Your code must contain three files, keygenerator, encrypt, and decrypt

(b) On running keygenerator, it must output two files, public_key.txt and private_key.txt

(c) public_key.txt must contain e (the public key) on the first line, and n (the exponent), on the second line. You're welcome to use the more standard ASN.1 notation if you prefer, though.

(d) private_key.txt contains only one line, which is the value of d, the private key.

(e) encrypt may assume the existence of two files, public_key.txt (which may not have been generated by your program, but will follow the same output format as **the one you have mentioned in your readme**), and plaintext.txt, which will be a **plaintext string**. encrypt is expected to output a file called ciphertext.txt.

(f) decrypt may assume the existence of three files, ciphertext.txt, public_key.txt, private_key.txt, and should output deciphertext.txt, which should be identical to plaintext.txt. This should be obvious, but you may not assume the existence of plaintext.txt in order to decrypt to plaintext.txt.

(g) Please submit the time it takes for your keygen.py to run. This can be performed on the terminal by running *time (program_run_command)* (note that your program should not take any input through the terminal, or the results will be invalid). Do this five times, and note the results (as well as the average), in your readme.

Problem 1-10. Implement Elgamal [30 points]

Write a program to implement the following variant of the ElGamal public key encryption scheme.

- **KeyGen** : This algorithm proceeds as follows,
  - –Choose two primes p and q such that $q = 2p + 1$, and p is a 300-bit prime.
  - –Choose g such that g is the square of a primitive root mod q
  - –Choose a number a uniformly at random from the set $\{2, 3, o(g) - 1\}$.
  - –Compute $h = g^a \mod q$.
  - –Set $\mathsf{PK} = (q, g, h); \mathsf{SK} = a$.
- **Encrypt** : To encrypt a message m under the public key $\mathsf{PK} = (q, g, h)$, this algorithm proceeds as follows:
  - –Choose a random element $r \in \{2, ..., q - 1\}$
  - –Compute $C_1 = g^r \mod q$
  - –Compute $C_2 = (h^r \times m) \mod q$
  - –Output, ciphertext $C = (C_1, C_2)$
- **Decrypt** : To decryt a ciphertext C = (C_1, C_2) under the secret $\mathsf{SK} = a$, this algorithm proceeds as follows:
  - –Compute $t_1 = (C_1)^a \mod q$.
  - –Compute $t_2 = (C_2 \times t_1^{-1}) \mod q$
  - –Output decrypted message as t_2