Homework 3

This homework is due on *Monday, February 17, 2014* by 10:39:00AM. There are **25 possible points**. N.B.: The submission process is different for this homework. Submit problem 3-2 parts A and C (next generation) both electronically and on paper. The electronic method is CTools, not email. We recommend performing test submissions early on CTools to avoid losing points at the last minute if you should find your CTools account is not setup properly. You may submit updated code as many times as you wish up until the deadline. We do not accept emailed submissions. Late submissions or submissions that do not follow the strict policies from the syllabus will receive a zero. Please contact the TA well ahead of the deadline if you have a question about these procedures.

Submit each homework[1] problem stapled separately. We will have separate boxes at the front of the lecture hall for submitting each problem. Points will be deducted for solutions that do not follow this process. On the front page of **each** stapled problem, the top right corner must provide the following items:

- **Print your discussion section time**
- Print your name
- Print your uniqname
- Print the homework-problem number.

Example:

Discussion: Fri 10:30AM
Name: John P. Doe
Uniqname: johnpdoe
Problem: 3-1

If your solution spans multiple pages for a problem, mark the top of each page with your initials. Some problems contain multiple components. For each component of a problem, write a descriptor of the component. Points may be deducted if your TA has problems understanding or reading your solution. In mathematical problems, **show all your work**. If you receive any key insights from someone else or some other resource, you must cite that person or resource.

Problem 3-1. Prime time (5 pts)

In lecture, we learned about several ways to test for primality. Explain the FLT-based primality test in your own words, why it works, and what is its main shortcoming. Discuss the advantages and disadvantages relative to the quadratic residuosity test and the AKS test. Explain in your own words why cryptosystems generally use probabilistic algorithms rather than deterministic algorithms when randomly selecting prime numbers. Submit your response on paper only.

[1] We will distribute our favorite solution for each problem to the class as the "official" solution—this is your chance to become famous!

Problem 3-2. The next generation (15 pts)

This problem involves a non-trivial amount of algorithmically intense programming. Get started early!

$\Longrightarrow$ Submit this problem on both paper and CTools. $\Longleftarrow$

Programming administrivia. Please do not plagiarize. Every year our department has to prosecute cases of plagiarized code. It's so easy to make a dishonorable choice by copying code that we heavily penalize plagiarism (see Handout 1). We check for plagiarism with automated source code analysis tools. Students referred to the honor council will not receive a final grade until the honor council resolves the case. Please do not plagiarize.

We recommend using C++ and `libgmp` to implement the modular arithmetic. The bit lengths of the integers in this problem set are much larger than what most programming languages (and microprocessors!) natively support. You'll likely experience integer overflow error if you attempt to use plain old integers. If you wish to request permission to use a different programming language, please first get approval from the TAs via Piazza. C++ is the only supported language. Some students have successfully used Java's BigInteger abstraction for multi-precision arithmetic.

For this particular problem, you are free to use any built-in procedures for primality testing, prime number generation, or modular reductions. But you may not use any built-in functions for producing generators and you may not use other algorithms that you must implement. You can (and should) assume you have a source of randomness. You may not borrow any other code for cryptography beyond a "big number" or multi-precision number package.

TAs will be using an autograder to test various inputs to your function. The TAs will post a more specific function prototype on Piazza. Make sure to follow the spec.

(a) Generators of subgroups of Z_p^* (6 pts)

Submit your documented code on both paper and electronically via CTools.

Write a function to perform the following task. Given large positive integers g, p and the prime factorization of $p-1$ where q is the largest prime factor of $p-1$ and p is prime, the function will test whether the element g is a generator of a *subgroup* of Z_p^* of order q.

That is, you will be producing a generator of a subgroup $G \subset Z_p^*$ where $p = 2qt+1$ for some primes p, q and a positive integer t where no prime factor of t is larger than q.

For instance, $7 = 2 * 3 + 1$. Here, our $q = 3$. In lecture, we will see that modulo 7 there are two elements of order 3. Namely, the elements 2 and 4 generate the subgroup $\{1, 2, 4\}$, which has order 3.

Using these functions, write a program that given a list of first names, a number b, and a source of randomness, produces a b-bit prime number p for such that each name (treated as an element of Z_p) is a generator of a subgroup of Z_p^* with order q.

Note, it may be helpful to write a function to generate a b-bit prime given parameters b and a source of randomness. Your function would then produce a b-bit number that has an overwhelming probability of being prime. Or you may use a built-in primality testing function.

A potentially comforting fact is that there are many generators of a subgroup of Z_p^* of order q. In the special case where your prime $p = 2q + 1$ and q is prime, then $\phi(q) = q - 1$. If you generate a prime p with this form, then you already know the factors of $p - 1$.

To encode names as elements, take the hexadecimal ASCII representation. Then combine the digits into a single number. The first letter of a name is the high order byte, the last letter of the name is the low order byte, etc. For example:

Name	ASCII Hex	Element in decimal
Kevin	0x4B 65 76 69 6E	323824806254
Emily	0x45 6D 69 6C 79	298188369017
Alex	0x41 6C 65 78	1097622904

Your program should work efficiently for an arbitrarily long list of names.

(b) Expected amount of work to find primes and generators (4 pts)

For a randomly generated prime number p, what is the approximate probability of a randomly selected element $g \in Z_p$ being a generator of a subgroup of Z_p^* of order q as defined in part (a)? Why? You may assume p is large.

How many trials do you expect to make for a list of three names to find a suitable prime p such that each name is a generator of a subgroup of Z_p^* with order q? Give your estimate in terms of b (the size of a b-bit prime p).

Submit on paper only.

(c) Find a prime (3 pts)

Submit this problem both on paper and electronically via CTools. Make sure to tell us the p and q in decimal in both submissions. Also provide your selected names in ASCII, hex, and decimal.

Pick two of your favorite names in addition to your first name. Try to pick a clever set of names such that no one else picks your names. But don't share your names with your classmates!

Using your program from part (a), find a prime number p where the names are generators of a subgroup of Z_p^* with order $q = \frac{p-1}{2}$. (1) Tell us this prime p. You can try for any size prime. We recommend trying small bit sizes (e.g., 64) for testing. If you finish early, try to generate a 1024-bit prime (this could take a lot of time, so only do this for fun if you have time).

(2) In your final run of the working program, how many trials did it take to find a suitable prime? That is, how many random numbers did you try before finding a suitable prime?

(d) Finding co-Sophie-Germain primes (2 pts)

We alluded to *Sophie-Germain* primes in this problem. A number q is a Sophie-Germain prime if $2q + 1$ is also prime. We then call the value $2q + 1$ a co-Sophie-Germain prime (or sometimes a *safe prime*).

While we know there are an infinite number of primes, it's an open problem whether there are an infinite number of Sophie-Germain primes. It turns out that in practice, however, Sophie-Germain primes are not too rare.

Assuming that the likelihood of q being prime and $2q + 1$ being prime is independent, approximately how many 2048-bit Sophie-Germain primes would you expect to exist?

Submit on paper only.

Problem 3-3. The remainder of this homework (5 pts)

Do problem 5.13 from Stinson p. 227 (Chinese Remainder Theorem and RSA with three problem subparts). For the numeric subparts of the problem, feel free to solve the problem manually or use Sage or your own computer program to arrive at the answers, but you must show all the steps of your computations. That is, explain at each step why you are computing what you are computing. Submit on paper only.