CMDA 3634 Spring 2018 Homework 01

Weichen Li

February 5, 2018

You must complete the following task by 5pm on Tuesday 02/06/18.

Your write up for this homework should be presented in a LaTeX formatted PDF document. You may copy the LaTeX used to prepare this report as follows

1. Sign up for a http://sharelatex.com account.
2. Click on this link
3. Click on Menu/Copy Project.
4. Modify the HW01.tex document to respond to the following questions.
5. Remember: click the Recompile button to rebuild the document when you have made edits.
6. Remember: Change the author.

Each student must individually upload the following files to the CMDA 3634 Canvas page at https://canvas.vt.edu

1. `firstnameLastnameHW01.tex` LaTeX file.
2. Any figure files to be included by `firstnameLastnameHW01.tex` file.
3. `firstnameLastnameHW01.pdf` PDF file.
4. All source code (`.c`) files.

You must complete this assignment on your own.

100 points will be awarded for a successful completion.
Extra credit will be awarded as appropriate.

ElGamal Public-key Cyptography

Suppose there are two individuals, lets call them Alice and Bob, who wish to hold a conversation where only they can read the contents of each of their messages. That is, no outside eavesdropper can discern what the contents of each message is. Let's call Eve one such potential eavesdropper. For such a system, Alice and Bob need to agree on a way to encrypt and decrypt their messages in a completely public way such that Eve cannot decyrpt their messages.

Let consider just the case of Bob sending an encrypted message to Alice. A possible cryptographic system that Alice and Bob could agree on is a *public-key cryptographic system.* Such a system would contain the following steps.

1. Alice chooses in secret a *secret key* and publicly broadcasts a *public key.*
2. Bob uses the public key to encrypt his message and sends to Alice.
3. Alice uses her secret key to decrypt Bob's message.

In this process, Eve has access to the public key that was broadcast by Alice. She can also see the encrypted message send to Alice by Bob. The cryptographic system should therefore be such that

- Encryption is easy using the public key.
- Decryption is easy using the secret key.
- Finding the secret key is *really hard.*

The public-key cryptographic system that we will consider in this class is the ElGamal encryption. We begin by introducing some concepts regarding integers modulo prime numbers.

Integers Modulo Primes: While ElGamal encryption can work using more exotic mathematical structures, we will focus on integers modulo some prime p. We define the set $\mathbb{Z}_p$ to be the following numbers,

$$\mathbb{Z}_p = \{1, 2, 3, \ldots, p-1\},$$

for some prime number p. Since p is prime, notice that $\mathbb{Z}_p$ is closed under multiplication modulo p. That is,

$$\forall a, b \in \mathbb{Z}_p, \qquad ab \pmod{p} \in \mathbb{Z}_p.$$

In particular, the product ab cannot be zero. We'll drop the $\pmod{p}$ in products from now on, but note that all products are assumed to be modulo p so that all numbers remain in the set $\mathbb{Z}_p$.

A less obvious property of $\mathbb{Z}_p$ is that there will exist at least one number g in $\mathbb{Z}_p$ such that

$$\mathbb{Z}_p = \left\{g, g^2, g^3, \ldots, g^{p-1}\right\}.$$

We call such a number a *generator* of $\mathbb{Z}_p$.

Another less-than-obvious property is that for each number a in $\mathbb{Z}_p$, we have that $a^{p-1} = 1 \pmod{p}$. This fact is a consequence of a theorem known as Cauchy's Theorem in a field of mathematics called Group Theory. It is especially useful when we notice that, consequently, every number a in $\mathbb{Z}_p$ has a multiplicative 'inverse', denoted a^{-1} in $\mathbb{Z}_p$, such that $aa^{-1} = 1$. In particular, $a^{-1} = a^{p-2}$.

Key Generation: The key generation for ElGamal encryption works as follows.

1. Alice chooses prime number p.
2. Alice selects a generator g of the group $\mathbb{Z}_p$.
3. Alice chooses, in secret, a number x in $\mathbb{Z}_p$ and computes $h = g^x$.
4. Alice publishes the set (p, g, h) as her public key and retains x as her secret key.

Encryption: The ElGamal encryption for a message m works as follows. We'll suppose for simplicity that the message m is already a number in $\mathbb{Z}_p$ and consider how to cast actual message text into numbers in $\mathbb{Z}_p$ later.

1. Bob chooses a random y in $\mathbb{Z}_p$.
2. Bob computes $a = g^y$ and $s = h^y$.
3. Bob computes the cyphertext $\hat{m}$ as $\hat{m} = ms$.
4. Bob sends the pair $(a, \hat{m})$ to Alice.

Decryption: The ElGamal decryption of the cyphertext $\hat{m}$ works as follows.

1. Alice uses her secret key x to compute the shared secret $s = a^x = g^{xy}$.
2. Alice computes $s^{-1} = s^{p-2}$.
3. Alice decrypts the intended message as $\hat{m}s^{-1} = mss^{-1} = m$.

Security: The security of the ElGamal encryption algorithm relies on the problem of finding the secret key x being difficult. Since Eve knows both g and h from Alice's public key, this amounts to her finding x such that

$$g^x = h \pmod{p}.$$

This problem is known as the *discrete logarithm problem* and is considered to be computationally intractable for large prime numbers p. Indeed, our goal for the assignments in this class will be to implement our own ElGamal cryptographic system and determine how big the prime number p must be before we simply will not have the computational power to crack it.

Questions

Q1(5 points) Show that 2, 6, 7, and 11 are all generators of $\mathbb{Z}_{13}$.

$$\mathbb{Z}_{13} = \{1, 2, 3, \ldots, 12\} \tag{1}$$
$$2 : \{2, 2^2, 2^3, \ldots, 2^{12}\} = \{2, 4, 8, \ldots, 1\} \in \mathbb{Z}_{13} \tag{2}$$
$$6 : \{6, 6^2, 6^3, \ldots, 6^{12}\} = \{6, 10, 8 \ldots, 1\} \in \mathbb{Z}_{13} \tag{3}$$
$$7 : \{7, 7^2, 7^3, \ldots, 7^{12}\} = \{7, 10, 5, \ldots, 1\} \in \mathbb{Z}_{13} \tag{4}$$
$$11 : \{11, 11^2, 11^3, \ldots, 11^{12}\} = \{11, 4, 5, \ldots, 1\} \in \mathbb{Z}_{13} \tag{5}$$

Q2(5 points) What is 6^{-1} in $\mathbb{Z}_7$?

$$6 * 6^{-1} = 1 \tag{6}$$
$$6^{-1} = 6^5 \tag{7}$$
$$6^{-1} = 6 \tag{8}$$

Q3.1(5 points) Given an ElGamal public-key of $(11, 2, 3)$, that is the prime $p = 11$, the generator $g = 2$, and $h = 3$, encrypt the message $m = 9$ using $y = 2$.

$$a = g^y = 2^2 = 4 \tag{9}$$
$$s = h^y = 3^2 = 9 \tag{10}$$
$$\hat{m} = ms = 4 \tag{11}$$
$$(a, \hat{m}) = (4, 4) \tag{12}$$

Q3.2(10 points) Given that the secret key in the ElGamal public key in Q3.1 is $x = 8$, decrypt the cyphertext you computed in Q3.1 and confirm that you recover the original message.

$$s = a^x = 4^8 = 9 = g^{xy} = 2^{8y} = 2^{16} \tag{13}$$
$$s^{-1} = s^{p-2} = 9^9 = 5 \tag{14}$$
$$\hat{m}s^{-1} = mss^{-1} = m \tag{15}$$
$$m = 4 \times 5 = 9 \tag{16}$$

Q4.1(20 points) In order to write a C program to perform an ElGamal encryption, we'll need a few simple functions to help out with prime numbers. One such function is the greatest common divisor (GCD) of two numbers. This function inputs two numbers a and b and returns the greatest integer z such that z divides both a and b. For example $GCD(15, 25) = 5$ since 5 divides both 15 and 25.

Without loss of generality let's assume $a \geq b$. The algorithm to compute the GCD of a and b can be written compactly as

$$GCD(a,b) = \begin{cases} GCD(b, a \pmod b) & \text{if } b \neq 0, \\ a & \text{if } b = 0. \end{cases}$$

Write a C program `gcd.c` which replicates the following functionality.

```
>./gcd
Enter the first number: 15
Enter the second number: 25
The greatest common divisor of 15 and 25 is 5.
```

That is, your program should read in two values from the user and print the greatest common divisor of the two values. To read in values use the `scanf` function.

HINT: In C it is legal for functions to call themselves, i.e. make recursive functions.

Q4.2(5 points) Another useful function is the least common multiple (LCM) of two numbers. This function inputs two numbers a and b and returns the smallest number z such that both a and b divide z. For example $LCM(32, 20) = 160$.

The algorithm to compute the LCM of a and b can be written using the GCD function as follows

$$LCM(a,b) = \frac{ab}{GCD(a,b)}.$$

Write a C program `lcm.c` which replicates the following functionality.

```
>./lcm
Enter the first number: 32
Enter the second number: 20
The least common multiple of 32 and 20 is 160.
```

Q4.3(5 points) Two numbers a and b are considered coprime if the only positive number which divides both a and b is 1, i.e. $GCD(a, b) = 1$. Write a C program `isCoprime.c` which replicates the following functionality,

```
>./isCoprime
Enter the first number: 16
Enter the second number: 9
16 and 9 are coprime.
```

and

```
>./isCoprime
Enter the first number: 32
Enter the second number: 12
32 and 12 are not coprime.
```

Q5(20 points) To generate public keys in the ElGamal encryption we'll need a way to select prime numbers. Write a C program `isPrime.c` which replicates the following functionality,

```
>./isPrime
Enter a number: 32
32 is not prime.
```

and

```
>./isPrime
Enter a number: 17
17 is prime.
```

There are many sophisticated ways to accomplish this but we are only interested in whether your program can correctly operate for the first several hundred prime numbers. Don't worry about being inefficient for now.

Q6(25 points) Once we have a prime number, the next task is to find a generator of $\mathbb{Z}_p$. If you experiment with some other numbers in Q1 you may notice that if a is not a generator of $\mathbb{Z}_p$ then there will exist a number r, where $0 < r < p - 1$, such that $a^r = 1$. Therefore, for a number g to be a generator of $\mathbb{Z}_p$ it must satisfy

$$g^r \neq 1, \quad \forall 0 < r < p - 1.$$

Write a C program `findGenerator.c` which replicates the following functionality,

```
>./findGenerator
Enter a prime number: 11
2 is a generator of Z_11.
```

Bonus(20 points) We are going to need to examine each part of our code to make sure we don't run into trouble when using very large integers. For each of your codes in this assignment, give some comments on whether you will encounter issues when your inputs become very large numbers. Will your program seem to 'hang' due to computational load? Or fail to function correctly at all?

```
>./gcd
Enter the first number: 2
Enter the second number: 2222222222222222222222222222
The greatest common divisor of -1 and 2 is -1.
```

gcd fails to function.

```
>./lcm
Enter the first number: 5
Enter the second number: 5555555555555555555555555555
The least common multiple of 5 and -1 is 5.
```

lcm has a similar error comparing to gcd.

```
>./isCoprime
Enter the first number: 31364
Enter the second number: 1346234623465
1909859817 and 31364 are coprime.
```

isCoprime fails as well.

```
>./isPrime
Enter a number: 2835768726358764758
-809063210 is prime.
```

isPrime fails to function when the input gets very large. It uses a for loop to check if the number can be divided by from 2 to half of itself, and the efficiency is pretty low. Also when the input gets large.

```
>./findGenerator
Enter a prime number: 28374234567293845
1277705109 is not prime. You have to enter a prime number
>./findGenerator
Enter a prime number: 2384523423453245
530387005 is not prime. You have to enter a prime number
>./findGenerator
Enter a prime number: 1273461782634871263478
```

findGenerator involves isPrime to check if the input is a prime number, so it is similar to the previous one. If I remove the check and enters a large number, the program hangs.