Aleem Ahmed
4/3/20
Dr. Samanthula
Diffie-Hellman Report

Understanding Diffie-Hellman

Implementing Diffie-Hellman was an interesting task to do especially with the requirements of a maximum input being 1024 as a bit size for the prime. I used java to implement the key exchange protocol and programmed it on Visual Studio Code. To implement the key exchange protocol, you have to understand how it works first. 2 parties who are trying to communicate, denoted as Alice and Bob, have to communicate over an insecure network. To securely do so they will use Diffie-Hellman's key exchange protocol. There are multiple variables in this exchange. p, g, a, b, A, B, and M. out of these variables some are public and some are private. P, g, A, and B are public and a person in the middle will know what these are but will still not be able to decrypt the message. The p (prime number) usually has to be a very large number so that an eavesdropper cannot easily decrypt the data. Both parties who are trying to communicate with each other have to come up with a random number for themselves to be used as their secret variable. These variables are "a" and "b".

The first step is to generate the generator (g) and the prime number (p). After they are generated both Alice and Bob receive these variables. Alice then compute the A using the formula:

Alice: A = g^a mod p
Bob: B = g^b mod p

Bob also does the same to get B value by using b instead of a in the formula. Both parties share with each other their calculated values. A and B are public to the network. They then use the calculated values to decrypt the message (m) privately using the formula:

Alice: M = B^a mod p
Bob: M = A^b mod p

The message is decrypted privately and the eavesdropper cannot decrypt it because he or she does not have the secret values a nor b.

Implementation

At the beginning of the implementation I ran into the challenge of figuring out how I can take an input variable and convert it to binary value. I first created a function called "randomPrime" and I made this function take in the input from the user. This function then calls another function that I made called "createRangeBinaryStrings". This function takes in the same input from the user and returns an string array of binary numbers. In this function it creates the starting binary point and the ending binary point by first taking premade strings called rangeMin and rangeMax that are set to "1".

```
String rangeMin = "1";
String rangeMax = "1";
```

For the amount of the input passed, the function then adds a 0 at the end of rangeMin and a 1 at the end of rangeMax. For example if the input was five the rangeMin value would be 10000 and the rangeMax value would be 11111.

```
for (int i = 0; i < (int_numOfBits - 1); i++) {
    rangeMin = rangeMin.concat("0");
    rangeMax = rangeMax.concat("1");
}
```

This then gets put into an array to be returned to the "randomPrime" function. This is the range that the function will be looking between for prime numbers. The program could then convert the binary string to integer values. This was when i ran into the problem of the string value being too long to convert to an integer.

Due to the requirements i had to make big changes to my first implementation and use a special library for very large numbers called "bigIntegers". I was using "int" variables before which limited my program to only take inputs of 10 bits. "Int" variables have a max size of 4 bytes which are whole numbers between -2,147,483,648 and 2,147,483,647. This was not enough for 1024 bit numbers. So I looked for bigger variable types and came across "Long". "Long" was better but still not good enough even though it could store 8 bytes and whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807. I was told to use "BigInteger" variables which at first i did not understand. To test BigIntegers i declared it like this:

```
BigInteger test =
1797693134862315907729305190789024733617976978942306572734300811577326758055009
6313270847732240753602112011387987139335765878976881441662249284743063947412437
7767893424865485276302219601246094119453082952085005768838150682342462881473913
110540827237163350510684586298239947245938479716304835356329624224137215;
```

This did not work. So I emailed the professor thinking that BigInteger was not good enough. I had also read online that the max size of BigInteger was 64 Bits which is clearly below the 1024 Bit goal. I waited for a response and tried many other things in the meantime. For hours I could not find a better solution. I slept terribly. I woke up and the professor emailed me back reinforcing his initial recommendation of using "bigInteger" to run the program. So I relooked into it and found some online resource that stated information on how to properly use the "BigInteger" type. I learned that the the proper way to store large values was to do it like this:

```
BigInteger test = new
BigInteger("898846567431157953864652595394512366808988489471153286367150405788
6633790275048156635423866120376801056005693993569667882939488440720831124642371
5319737062188883946712432742638151109800623047059726541476042502884419075341171
2314407369565552704136185816752553422931491199736229692398581524176781648121120
68608");
```

This worked! But my next problem was figuring out how to store an array of these because if I was going to be collecting all primes within a given range I would need to store them somehow. I could not use a regular array because it can only store regular "int" values. I did more research and figured out that you could do an array of BigIntegers which looks like this:

```
BigInteger[] arrayOfBigNumbers = {
  new BigInteger("123"),
  new BigInteger("123")
};
```

BUT STILL this was a problem. I had no idea how to loop through something and add variables to an array using java. This was problematic because I did not know how many primes there were going to be. I researched more and found an alternative method which required me to use a "list" instead of an array. To create a list and input values into it looks like this:

```
List<BigInteger> list = new ArrayList<BigInteger>();
list.add(new BigInteger("123"));
list.add(new BigInteger("123"));
```

All i had to do now is to create a while loop and increment the values. The while loop would cycle through every single value between the given range and check if it was prime or not. To check if a value was prime or not i used a premade function within the math library called "isProbablyPrime". This function returns a true or false. I stored the value in a variable called "result"

```
boolean result = while_bigInt_rangeMin.isProbablePrime(1);
```

If it was true then I added that value to the list. I then had to create a random choice out of the list of prime numbers to return to the main function.

Once a random Prime was returned i just had to do the arithmetic with the other BigInteger variables. To do ***A = g^a mod*** p and ***B = g^b mod p*** the code looked like this:

```
// [ARITHMETIC] Alice: A = g^a mod p //
A = g.pow(int_a);
A = A.mod(p);
```

```
// [ARITHMETIC] Bob: B = g^b mod p //
B = g.pow(int_b);
B = B.mod(p);
```

And finally to compute the Message i had to do ***M = A^b mod p*** and **M = B^a mod P.** The code for this looks like this:

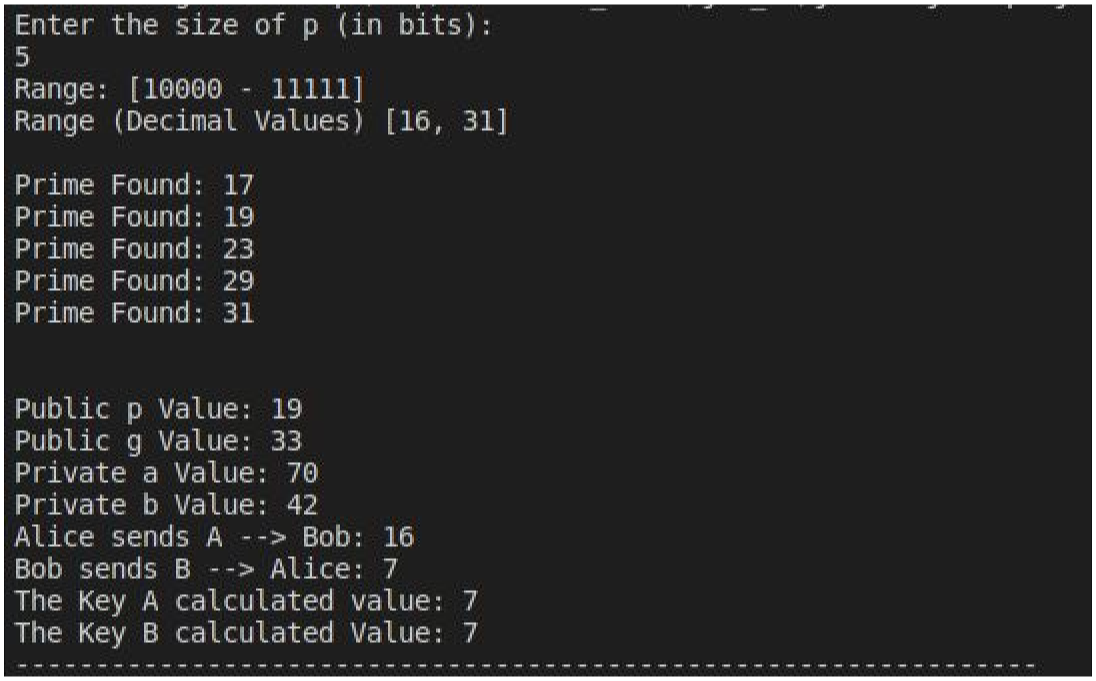
```
// [ARITHMETIC] Alice: M = B^a mod p //
messageA = B.pow(int_a);
messageA = messageA.mod(p);

// [ARITHMETIC] Bob: M = A^b mod p //
messageB = A.pow(int_b);
messageB = messageB.mod(p);
```

Both parties, Alice and Bob can now communicate securely over an insecure network! This assignment was very difficult but also very fun to do.

These are some test runs..

5 bits (Prime Found Enabled):

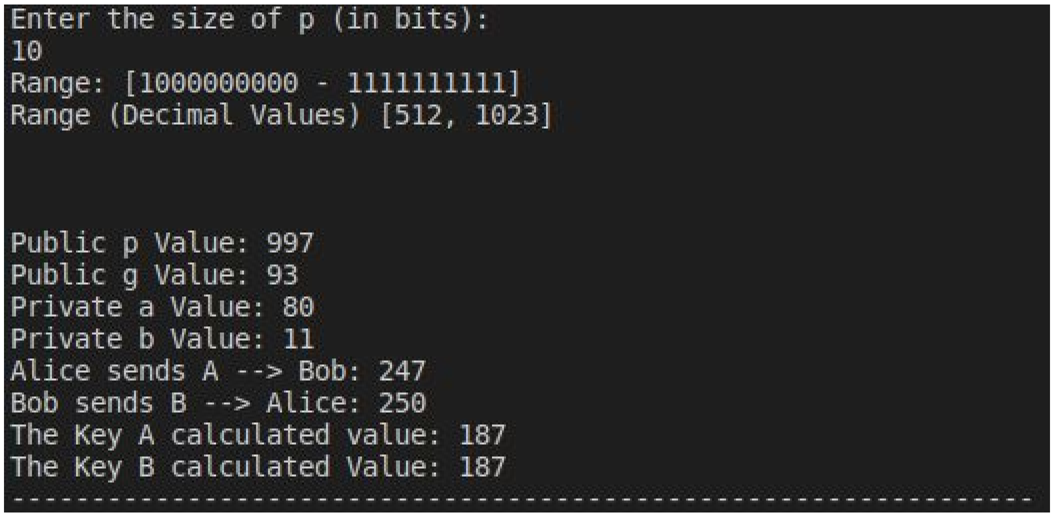


```
Enter the size of p (in bits):
5
Range: [10000 - 11111]
Range (Decimal Values) [16, 31]

Prime Found: 17
Prime Found: 19
Prime Found: 23
Prime Found: 29
Prime Found: 31


Public p Value: 19
Public g Value: 33
Private a Value: 70
Private b Value: 42
Alice sends A --> Bob: 16
Bob sends B --> Alice: 7
The Key A calculated value: 7
The Key B calculated Value: 7
---------------------------------------------------------------
```

10 bits:



```
Enter the size of p (in bits):
10
Range: [1000000000 - 1111111111]
Range (Decimal Values) [512, 1023]



Public p Value: 997
Public g Value: 93
Private a Value: 80
Private b Value: 11
Alice sends A --> Bob: 247
Bob sends B --> Alice: 250
The Key A calculated value: 187
The Key B calculated Value: 187
---------------------------------------------------------------
```