Dr. Phil Maguire

CS211FZ ALGORITHMS & DATA STRUCTURES II

LAB 5: PUBLIC KEY CRYPTOGRAPHY

Objectives

- Understand how public key and private key work.
- Use public key / private key.
- Reflect on the knowledge learned in the class.

NOTE:

- Do NOT use “package” in your source code.
- You must submit the source code files, i.e., the “.java” files.
- You are allowed to use course reference books or class notes during the lab.
- Sharing/copying your work is NOT allowed.

Pen and Paper Exercise

Your public key is (29, 2, 3) and your private key is 5. Bob uses your public key to send you a message. He sends you the cipher (23, 27). Use your private key to obtain the secret message code number. Read over the lecture notes to get the required formulas.

Task: Programming Exercise

Alice’s public key is (24852977, 2744, 8414508). You eavesdrop on the line and observe Bob send her the cipher (15268076, 743675). Extract the message by any means.

Warning: For the programming part, make sure to use *longs* rather than *ints* (you may need to put an ‘l’ at the end of the number). Also, make sure to keep moduloing every time the number in the calculation gets a little too big – never do large power multiplications straight off because Java cannot process large numbers like this.

As a result, it might be worthwhile to use these recursive methods for modulo multiplication and modulo raising to a power. These work efficiently and ensure that the numbers never get too big.

```
import java.util.*;

public class PublicKeyCryptography
{
   public static void main (String args[]){

//Add your code here.
}

   public static long modPow(long number, long power, long modulus){
      if(power==0)
         return 1;
      else if (power%2==0) {
         long halfpower=modPow(number, power/2, modulus);
      return modMult(halfpower,halfpower,modulus);
      }else{
         long halfpower=modPow(number, power/2, modulus);
         long firstbit = modMult(halfpower,halfpower,modulus);
         return modMult(firstbit,number,modulus);
      }
   }

   public static long modMult(long first, long second, long modulus){
      if(second==0)
         return 0;
      else if (second%2==0) {
         long half=modMult(first, second/2, modulus);
         return (half+half)%modulus;
      }else{
         long half=modMult(first, second/2, modulus);
         return (half+half+first)%modulus;
      }
   }
}
```

Comment for function modPow(): Raises a number to a power with the given modulus. When raising a number to a power, the number quickly becomes too large to handle. You need to multiply numbers in such a way that the result is consistently moduloed to keep it in the range. However, you want the algorithm to work quickly - having a multiplication loop would result in an O(n) algorithm!
The trick is to use recursion - keep breaking the problem down into smaller pieces and use the modMult method to join them back together.

Comment for function modMult(): Multiplies the first number by the second number with the given modulus. A long can have a maximum of 19 digits. Therefore, if you're multiplying two ten digits numbers the usual way, things will go wrong. You need to multiply numbers in such a way that the result is consistently moduloed to keep it in the range. However, you want the algorithm to work quickly - having an addition loop would result in an O(n) algorithm!
The trick is to use recursion - keep breaking down the multiplication into smaller pieces and mod each of the pieces individually.