CT255 Assignment 4

Problem 1

```
import java.util.HashSet;
import java.util.Random;

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

        // code for problem 1
        problem1();
        // Man in the middle - problem 2
        mitm();
    }

    public static void problem1() {
        int p = generatePrime();
        int a = findPrimitive(p);

        int Alice = generator(a, 3, p);
        int Bob = generator(a, 5, p);

        int key_Alice = generator(Bob, 3, p);
        int key_Bob = generator(Alice, 5, p);

        System.out.println("Bob's public value: " + Bob);
        System.out.println("Alice's public value: " + Alice);
        System.out.println("Their secret key is (calculated by Alice) " + key_Alice);
        System.out.println("This is the same as (calculated by Bob) " + key_Bob);
    }

    public static void mitm() {
        // example of the man in the middle;
        int p = generatePrime();
        int a = findPrimitive(p);

        // public values
        int Alice = generator(a, 3, p);
        int Bob = generator(a, 5, p);
        int Malory = generator(a, 6, p);

        // secret

        int keys_malory_alice = generator(Alice, 6, p);
        int keys_alice_malory = generator(Malory, 3, p);

        int keys_malory_bob = generator(Bob, 6, p);
        int keys_bob_malory = generator(Malory, 5, p);
        System.out.println("\n\nMan in the middle attack\n");
        System.out.println("Alice's public key is " + Alice);
        System.out.println("Bob's public key is " + Bob);
        System.out.println("Malory's public key is " + Malory);
        System.out.println("\nTheir secret key is (Between Alice and Malory) " +
                keys_alice_malory + " which is the same as (Between Malory and Alice) " +
 keys_malory_alice);
        System.out.println("\nTheir secret key is (Between Bob and Malory) " +
                keys_bob_malory + " which is the same as (Between Malory and Bob) " + keys
_malory_bob);

    }

    public static int generator(int a, int x, int p) {
        return (int) Math.pow(a, x) % p;
    }

    // generate Prime Number
    public static int generatePrime() {
        // we pick a random number between 10^4 < p < 10^5
        Random random = new Random();
        int prime = random.nextInt((int) (Math.pow(10, 5) - Math.pow(10, 4))) + (int) Mat
h.pow(10, 4);
        // repeat random process until we find a prime number
        while (!checkPrime(prime)) {
            prime = random.nextInt((int) (Math.pow(10, 5) - Math.pow(10, 4))) + (int) Mat
h.pow(10, 4);
        }

        // return prime number
        return prime;
    }

    // checks if a number num is prime
    public static boolean checkPrime(int num) {
        // if num  is divisible by 2
        // if true it is not a prime number
        if (num % 2 == 0)
            return false;

        // generally in prime number checkers we check for num = 2
        // but we are looking for a number 10^4 < p < 10^5 so,
        // we don't have to check for it

        // then we run a for loop until i reached the number num
        // we check if it is a factor of i
        // if it is, then again it is not a prime number
        for (int i = 2; i < num; i++) {
            if (num % i == 0) {
                return false;
            }
        }
        // if it passes till here it is a prime number and we return true
        return true;
    }

    static int power(int x, int y, int p) {
        int res = 1; // Initialize result

        // we check if x is greater than or equal to p
        // and make x either 0 or less than p
        x = x % p;


        while (y > 0) {
            // If y is odd, multiply x with result ensuring it is less than p
            if (y % 2 == 1) {
                res = (res * x) % p;
            }

            // halving y will result in even number (integer)
            y /= 2;
            // x = x^2 mod p
            x = (x * x) % p;
        }
        return res;
    }

    // Utility function to store prime factors of a number
    static void findPrimefactors(HashSet<Integer> s, int n) {
        // Print the number of 2s that divide n
        while (n % 2 == 0) {
            s.add(2);
            n = n / 2;
        }

        // n must be odd at this point. So we can skip
        // one element (Note i = i +2)
        for (int i = 3; i <= Math.sqrt(n); i = i + 2) {
            // While i divides n, print i and divide n
            while (n % i == 0) {
                s.add(i);
                n = n / i;
            }
        }

        // This condition is to handle the case when
        // n is a prime number greater than 2
        if (n > 2) {
            s.add(n);
        }
    }

    // Function to find smallest primitive root of n
    static int findPrimitive(int n) {
        HashSet<Integer> s = new HashSet<Integer>();
        // Check if n is prime or not
        if (checkPrime(n) == false) {
            return -1;
        }

        // Find value of Euler Totient function of n
        // Since n is a prime number, the value of Euler
        // Totient function is n-1 as there are n-1
        // relatively prime numbers.
        int phi = n - 1;

        // Find prime factors of phi and store in a set
        findPrimefactors(s, phi);

        // Check for every number from 2 to phi
        for (int r = 2; r <= phi; r++) {
            // Iterate through all prime factors of phi.
            // and check if we found a power with value 1
            boolean flag = false;
            for (Integer a : s) {

                // Check if r^((phi)/primefactors) mod n
                // is 1 or not
                if (power(r, phi / (a), n) == 1) {
                    flag = true;
                    break;
                }
            }

            // If there was no power with value 1.
            if (flag == false) {
                return r;
            }
        }

        // If no primitive root found
        return -1;
    }

}
    }
```