```java
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        // Generate a prime number
        int prime = genPrime();
        // Find a primitive root of the prime number
        int base = primRoot(prime);

        // Generate private keys for person A and B
        int personA = genPriv(prime);
        int personB = genPriv(prime);

        // Calculate public keys for person A and B
        int pubA = calcPub(base, personA, prime);
        int pubB = calcPub(base, personB, prime);


        //Problem 1
//        int sharedA = calcSec(base, personA, prime);
//        int sharedB = calcSec(base, personB, prime);

//        System.out.println("Prime: " + prime);
//        System.out.println("Base: " + base);
//        System.out.println("Private Key Person A: " + personA);
//        System.out.println("Private Key Person B: " + personB);
//        System.out.println("Public Key Person A: " + pubA);
//        System.out.println("Public Key Person B: " + pubB);
//        System.out.println("Shared Secret at Person A's end: " +
sharedA);
//        System.out.println("Shared Secret at Person B's end: " +
sharedB);

        //Problem 2
        int personC = genPriv(prime);
        int pubC = calcPub(base, personC, prime);

        int fakePersonA = pubC;
        int fakePersonB = pubC;

        int sharedA = calcSec(fakePersonB, fakePersonA, prime);
        int sharedB = calcSec(fakePersonA, fakePersonB, prime);

        int CsharedA = calcSec(personA, personC, prime);
        int CsharedB = calcSec(personA, personC, prime);

        System.out.println("Person A Public: " + pubA);
        System.out.println("Person B Public: " + pubB);
        System.out.println("Person C Public: " + pubC);
        System.out.println("Shared Secret at Person A with C key: " +
sharedA);
        System.out.println("Shared Secret at Person B with C key: " +
sharedB);
        System.out.println("Shared Secret at Person C with Person A key: "
+ CsharedA);
        System.out.println("Shared Secret at Person C with Person B key: "
+ CsharedB);
    }


    //The following code below was wrote at help from the lab tudor
    //It generates a prime number, a primitive root, a private key, checks
the primitive root and prime number etc.
    public static int genPrime() {
        int p = 0;

        Random random = new Random();

        do {
            p = random.nextInt(90000) + 10000;
        }
        while (!isPrime(p));

        return p;
    }

    public static int primRoot(int p) {
        int a = 0;

        Random random = new Random();

        do {
            a = random.nextInt(p);
        }
        while (!isPrimRoot(p, a));

        return a;
    }

    public static int genPriv(int p) {
        Random random = new Random();

        return random.nextInt(p - 1) + 1;
    }

    public static int calcPub(int base, int privKey, int prime) {
        Random random = new Random();

        return modPow(base, privKey, prime);
    }

    public static int calcSec(int pubKey, int privKey, int prime) {
        return modPow(pubKey, privKey, prime);
    }

    public static boolean isPrime(int p) {
        if (p <= 1) {
            return false;
        }

        for (int i = 2; i < Math.sqrt(p); i++) {
            if (p % i == 0) {
                return false;
            }
        }
        return true;
    }

    public static boolean isPrimRoot(int p, int a) {
        boolean ret = false;
        boolean ar[] = new boolean[p];
        for (int i = 1; i <= p; i++) {
            int index = (int) (Math.pow(a, i) % p);

            if (ar[index] == false) {
                ar[index] = true;
            }
            else {
                ret = true;
                break;
            }
        }
        return ret;
    }

    public static int modPow(int base, int exponent, int modulus) {
        if (modulus == 1) {
            return 0;
        }

        int result = 1;

        base = base % modulus;

        while (exponent > 0) {
            if (exponent % 2 == 1) {
                result = (result * base) % modulus;
            }
            exponent = exponent >> 1;

            base = (base * base) % modulus;
        }
        return result;
    }
}
```