```java
import java.math.BigInteger;
import java.net.*;
import java.io.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

public class SigningClientTCP{
    static InetAddress aHost;
    static int serverPort;
    static Socket clientSocket = null;
    static String replyString = "";


    public static void main(String args[]) throws NoSuchAlgorithmException {
        System.out.println("The client is running.\n");

        // ---------------------- Generate the key ----------------------
        // from https://github.com/CMU-Heinz-95702/Project-2-Client-Server

        // Each public and private key consists of an exponent and a modulus
        BigInteger n; // n is the modulus for both the private and public keys
        BigInteger e; // e is the exponent of the public key
        BigInteger d; // d is the exponent of the private key

        Random rnd = new Random();

        // Step 1: Generate two large random primes.
        // We use 400 bits here, but best practice for security is 2048 bits.
        // Change 400 to 2048, recompile, and run the program again and you will
        // notice it takes much longer to do the math with that many bits.
        BigInteger p = new BigInteger(400, 100, rnd);
        BigInteger q = new BigInteger(400, 100, rnd);

        // Step 2: Compute n by the equation n = p * q.
        n = p.multiply(q);

        // Step 3: Compute phi(n) = (p-1) * (q-1)
        BigInteger phi = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));

        // Step 4: Select a small odd integer e that is relatively prime to phi(n).
        // By convention the prime 65537 is used as the public exponent.
        e = new BigInteger("65537");

        // Step 5: Compute d as the multiplicative inverse of e modulo phi(n).
        d = e.modInverse(phi);

        System.out.println("RSA public key: (" + e + "," + n + ")");  // Step 6: (e,n) is the RSA public key
        System.out.println("RSA private key: (" + d + "," + n + ")\n");  // Step 7: (d,n) is the RSA private key


        // ---------------------- Generate the ID ----------------------
        // from https://github.com/CMU-Heinz-95702/Project-2-Client-Server

        String rawPublicKey = String.valueOf(e) +
String.valueOf(n);

        MessageDigest md  = MessageDigest.
getInstance("SHA-256");
        md.update(rawPublicKey.getBytes());

        byte[] digest = md.digest();

        BigInteger biID = new BigInteger(Arrays.
copyOfRange(digest, digest.length-20, digest.length
));
        String ID = biID.toString() + " ";
        // source: https://stackoverflow.com/
questions/18367539/slicing-byte-arrays-in-java


        // ---------------------- Connect socket
and create packet to send ----------------------
        try {
            // set to localhost to host on local
machine and set port
            aHost = InetAddress.getByName("
localhost");
            serverPort = 6789;

            // input server port to use
            System.out.print("Input a server side
port number: ");
            Scanner readline = new Scanner(System.
in);
            serverPort = readline.nextInt();

            clientSocket = new Socket("localhost",
serverPort);


            String operation = "";
            String value = "";

            String packet = "";

```

```java
            while (true) {
                System.out.println("\n1. Add a value to your sum.\n" +
                        "2. Subtract a value from your sum.\n" +
                        "3. Get your sum.\n" +
                        "4. Exit client");

                int in = readline.nextInt();

                if(in == 1){
                    operation = "1 ";
                    System.out.println("Enter value to add: ");
                    value = String.valueOf(readline.nextInt()) + " ";
                }
                else if(in == 2){
                    operation = "2 ";
                    System.out.println("Enter value to subtract: ");
                    value = String.valueOf(readline.nextInt() + " ");
                }
                else if(in == 3){
                    operation = "3 ";
                    value = "| ";
                }
                else if(in == 4){
                    System.out.println("Client side quitting. The remote variable server is still running."); // send reply if not halt
                    break;
                }

                packet += operation + ID + value + e + " " + n;

                // compute the digest with SHA-256
                byte[] bytesOfMessage = packet.getBytes("UTF-8");
                MessageDigest md2 = MessageDigest.getInstance("SHA-256");
                byte[] bigDigest = md2.digest(bytesOfMessage);
                byte[] messageDigest = new byte[bigDigest.length+1];

                // we only want two bytes of the hash for ShortMessageSign
                // we add a 0 byte as the most significant byte to keep
                // the value to be signed non-negative.
                messageDigest[0] = 0;   // most significant set to 0
                for(int i = 0; i < bigDigest.length; i++){
                    messageDigest[i+1] = bigDigest[i];
                }

                // From the digest, create a BigInteger
                BigInteger m = new BigInteger(messageDigest);

                // encrypt the digest with the private key
                BigInteger c = m.modPow(d, n);

                // send the packet
                packet += " " + c.toString();
                System.out.println("The result is : " + communicate(packet));

                packet = "";
            }

            // catch potential exceptions
        }catch (IOException ex) {
            System.out.println("IO Exception:" +
ex.getMessage());
        } finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException ex) {
                // ignore exception on close
            }
        }
    }

    // ---------------------- Proxy style communication code ----------------------
    public static String communicate(String in) throws IOException {
        BufferedReader read = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

        PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));

        out.println(in);
        out.flush();

        String replyString = read.readLine();

        return replyString;
    }
}
```

```java
import java.math.BigInteger;
import java.net.*;
import java.io.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;
import java.util.*;

public class VerifyingServerTCP{
    private static TreeMap<String, Integer> tree_map = new TreeMap<>();

    static String instructions = "1. Add a value to your sum.\n" +
            "2. Subtract a value from your sum.\n" +
            "3. Get your sum.\n" +
            "4. Exit client";

    public static void main(String args[]){
        System.out.println("The server is running."); // lab instructions


        Socket clientSocket = null;
        byte[] buffer = new byte[1000];  // set up packet buffer for client message
        try{
            // set up ports
            System.out.print("Input a server port number to listen on: ");
            Scanner readline = new Scanner(System.in);
            int serverPort = readline.nextInt(); // convert to int

            ServerSocket listenSocket = new ServerSocket(serverPort);


            String id;
            int value;
            String e;
            String n;

            while(true){ // loop to continue until 'halt!' is sent
                clientSocket = listenSocket.accept();

                Scanner in;
                in = new Scanner(clientSocket.getInputStream());

                PrintWriter out;
                out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));

                // operation request and reply
                while(in.hasNext()){
                    String requestString = in.nextLine();

                    String[] arrRequestString = requestString.split(" ");

                    // check id verification
                    boolean idSignatureCheck = idSignatureCheck(arrRequestString[0], arrRequestString[1], arrRequestString[2],
                            arrRequestString[3], arrRequestString[4], arrRequestString[5]);

                    if(!idSignatureCheck){
                        out.println("Error in request.");
                        out.flush();
                    }

                    else{
                        id = arrRequestString[1];

                        if(!tree_map.containsKey(id)){
                            if(arrRequestString[0].equals("1")){
                                value = Integer.valueOf(arrRequestString[2]);
                                tree_map.put(id, value);
                            }
                            else if(arrRequestString[0].equals("2")){
                                value = -1*Integer.valueOf(arrRequestString[2]);
                                tree_map.put(id, value);
                            }
                            else{
                                tree_map.put(id, 0);
                            }
                        }
                        else{
                            if(arrRequestString[0].equals("1")){
                                value = Integer.valueOf(arrRequestString[2]);
                                tree_map.replace(id, tree_map.get(id)+value);
                            }
                            else if(arrRequestString[0].equals("2")){
                                value = Integer.valueOf(arrRequestString[2]);
                                tree_map.replace(id, tree_map.get(id)-value);
                            }
                        }

                        System.out.println("\nVisitor Public Key: (" + arrRequestString[3] + ","
```

```java
 + arrRequestString[4] + ")");
                        System.out.println(" VisitorID: " + arrRequestString[1]);
                        System.out.println("ID and Signature Verified: Yes");
                        System.out.println(" Operation #: " + arrRequestString[0]);
                        System.out.println("Return Variable " + String.valueOf(tree_map.get(id)));

                        out.println(String.valueOf(tree_map.get(id)));
                        out.flush();
                    }
                }
            }
            // catch potential exceptions
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
            // If quitting (typically by you sending quit signal) clean up sockets
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        } finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }

//    operation + ID + value + e + " " + n;
    public static boolean idSignatureCheck(String operation, String id, String value, String e, String n, String signature) throws IOException, NoSuchAlgorithmException {

        // ----------------- Check ID -----------------
        String rawPublicKey = e + n;

        MessageDigest md  = MessageDigest.getInstance("SHA-256");
        md.update(rawPublicKey.getBytes());

        byte[] digest = md.digest();

        BigInteger idCheckBigInt = new BigInteger(Arrays.copyOfRange(digest, digest.length-20, digest.length));
        String idCheck = idCheckBigInt.toString();


        // ----------------- Check Message -----------------

        // Take the encrypted string and make it a big integer
        BigInteger encryptedHash = new BigInteger(signature);
        BigInteger decryptedHash = encryptedHash.modPow(new BigInteger(e), new BigInteger(n));

        // Get the bytes from messageToCheck
        String messageToCheck = operation + " " + id + " " + value + " " + e + " " + n;

        MessageDigest md2 = MessageDigest.getInstance("SHA-256");

        byte[] messageToCheckDigest = md2.digest(messageToCheck.getBytes("UTF-8"));

        // messageToCheckDigest is a full SHA-256 digest
        // take two bytes from SHA-256 and add a zero byte
        byte[] extraByte = new byte[
```

```java
messageToCheckDigest.length+1];

        extraByte[0] = 0;   // most significant set to 0
        for(int i = 1; i < extraByte.length; i++){
            extraByte[i] = messageToCheckDigest[i-1];
        }

        // Make it a big int
        BigInteger bigIntegerToCheck = new BigInteger(extraByte);

        // inform the client on how the two compare
        if((bigIntegerToCheck.compareTo(decryptedHash) == 0) && (idCheck.equals(id))) {
            return true;
        }
        else{
            return false;
        }
    }
}
```