Name : Sakshi Dhamapurkar

PRN : 21510073

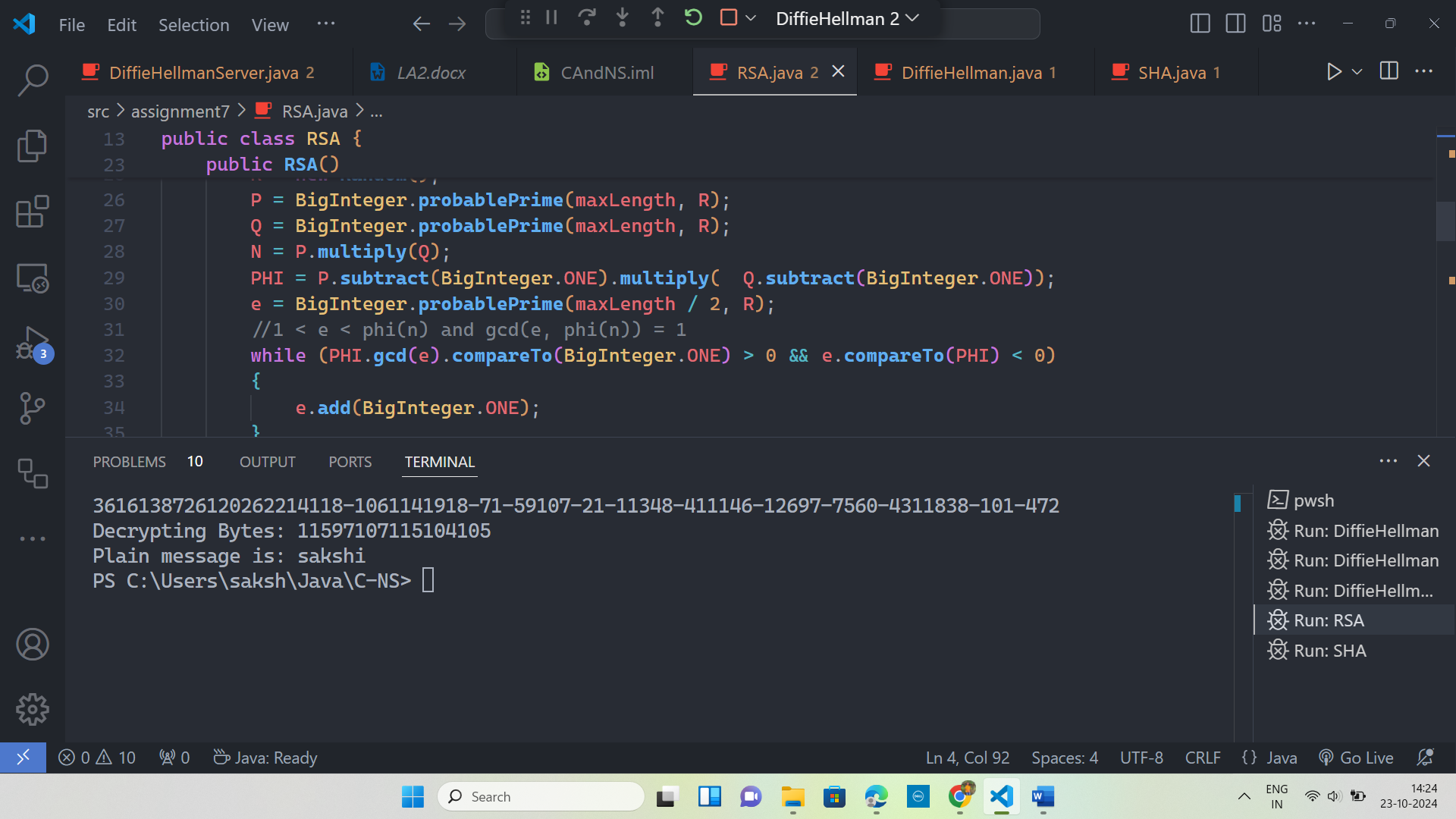
Batch : B4

Assignment 7 : RSA Implementation

CODE :

package assignment7;  
  
  
import java.io.DataInputStream;  
import java.io.IOException;  
import java.math.BigInteger;  
import java.security.\*;  
import java.util.Random;  
  
  
public class RSA {  
 // Method to encrypt the plaintext using RSA  
 private BigInteger P;  
 private BigInteger Q;  
 private BigInteger N;  
 private BigInteger PHI;  
 private BigInteger e;  
 private BigInteger d;  
 private int maxLength = 1024;  
 private Random R;  
 public RSA()  
 {  
 R = new Random();  
 P = BigInteger.*probablePrime*(maxLength, R);  
 Q = BigInteger.*probablePrime*(maxLength, R);  
 N = P.multiply(Q);  
 PHI = P.subtract(BigInteger.*ONE*).multiply( Q.subtract(BigInteger.*ONE*));  
 e = BigInteger.*probablePrime*(maxLength / 2, R);  
 while (PHI.gcd(e).compareTo(BigInteger.*ONE*) > 0 && e.compareTo(PHI) < 0)  
 {  
 e.add(BigInteger.*ONE*);  
 }  
 d = e.modInverse(PHI);  
 }  
  
 public static void main (String [] arguments) throws IOException  
 {  
 RSA rsa = new RSA();  
 DataInputStream input = new DataInputStream(System.*in*);  
 String inputString;  
 System.*out*.println("Enter message you wish to send.");  
 inputString = input.readLine();  
 System.*out*.println("Encrypting the message: " + inputString);  
 System.*out*.println("The message in bytes is:: "  
 + *bToS*(inputString.getBytes()));  
 // encryption  
 byte[] cipher = rsa.encryptMessage(inputString.getBytes());  
 System.*out*.println("Encrypted Message : " + *bToS*(cipher));  
 // decryption  
 byte[] plain = rsa.decryptMessage(cipher);  
 System.*out*.println("Decrypting Bytes: " + *bToS*(plain));  
 System.*out*.println("Plain message is: " + new String(plain));  
 }  
  
 private static String bToS(byte[] cipher)  
 {  
 String temp = "";  
 for (byte b : cipher)  
 {  
 temp += Byte.*toString*(b);  
 }  
 return temp;  
 }  
  
 // Encrypting the message  
 public byte[] encryptMessage(byte[] message)  
 {  
 return (new BigInteger(message)).modPow(e, N).toByteArray();  
 }  
  
 // Decrypting the message  
 public byte[] decryptMessage(byte[] message)  
 {  
 return (new BigInteger(message)).modPow(d, N).toByteArray();  
 }  
  
}

OUTPUT :



Assignment 8 : DiffieHellman Implementation using socket programing and without that

1.With socket programing :

Code:

DiffieHellmanServer.java code

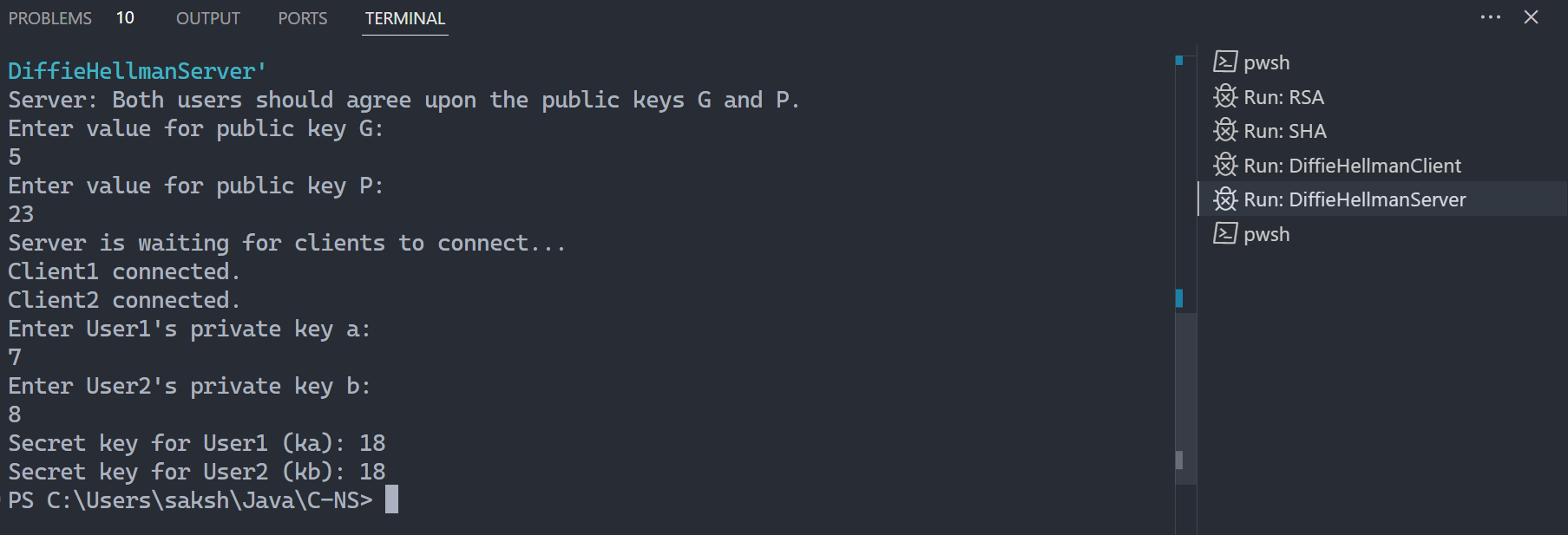
package assignment8;  
  
import java.io.\*;  
import java.net.\*;  
import java.util.Scanner;  
  
public class DiffieHellmanServer {  
 public static void main(String[] args) {  
 long P, G, y1, y2, ka, kb;  
 Scanner sc = new Scanner(System.*in*);  
  
  
 System.*out*.println("Server: Both users should agree upon the public keys G and P.");  
 System.*out*.println("Enter value for public key G:");  
 G = sc.nextLong();  
 System.*out*.println("Enter value for public key P:");  
 P = sc.nextLong();  
  
 try {  
  
 ServerSocket serverSocket = new ServerSocket(8080);  
 System.*out*.println("Server is waiting for clients to connect...");  
  
  
 Socket client1 = serverSocket.accept();  
 System.*out*.println("Client1 connected.");  
  
  
 Socket client2 = serverSocket.accept();  
 System.*out*.println("Client2 connected.");  
  
  
 DataInputStream in1 = new DataInputStream(client1.getInputStream());  
 DataInputStream in2 = new DataInputStream(client2.getInputStream());  
  
 y1 = in1.readLong();  
 y2 = in2.readLong();  
  
  
 System.*out*.println("Enter User1's private key a:");  
 long a = sc.nextLong();  
 System.*out*.println("Enter User2's private key b:");  
 long b = sc.nextLong();  
  
 ka = *calculatePower*(y2, a, P);  
 kb = *calculatePower*(y1, b, P);  
  
 DataOutputStream out1 = new DataOutputStream(client1.getOutputStream());  
 DataOutputStream out2 = new DataOutputStream(client2.getOutputStream());  
 out1.writeLong(ka);  
 out2.writeLong(kb);  
  
 System.*out*.println("Secret key for User1 (ka): " + ka);  
 System.*out*.println("Secret key for User2 (kb): " + kb);  
  
 client1.close();  
 client2.close();  
 serverSocket.close();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 private static long calculatePower(long base, long exp, long mod) {  
 long result = 1;  
 base = base % mod;  
 while (exp > 0) {  
 if (exp % 2 == 1) {  
 result = (result \* base) % mod;  
 }  
 exp = exp >> 1;  
 base = (base \* base) % mod;  
 }  
 return result;  
 }  
}

DiffieHellmanClient.java code :

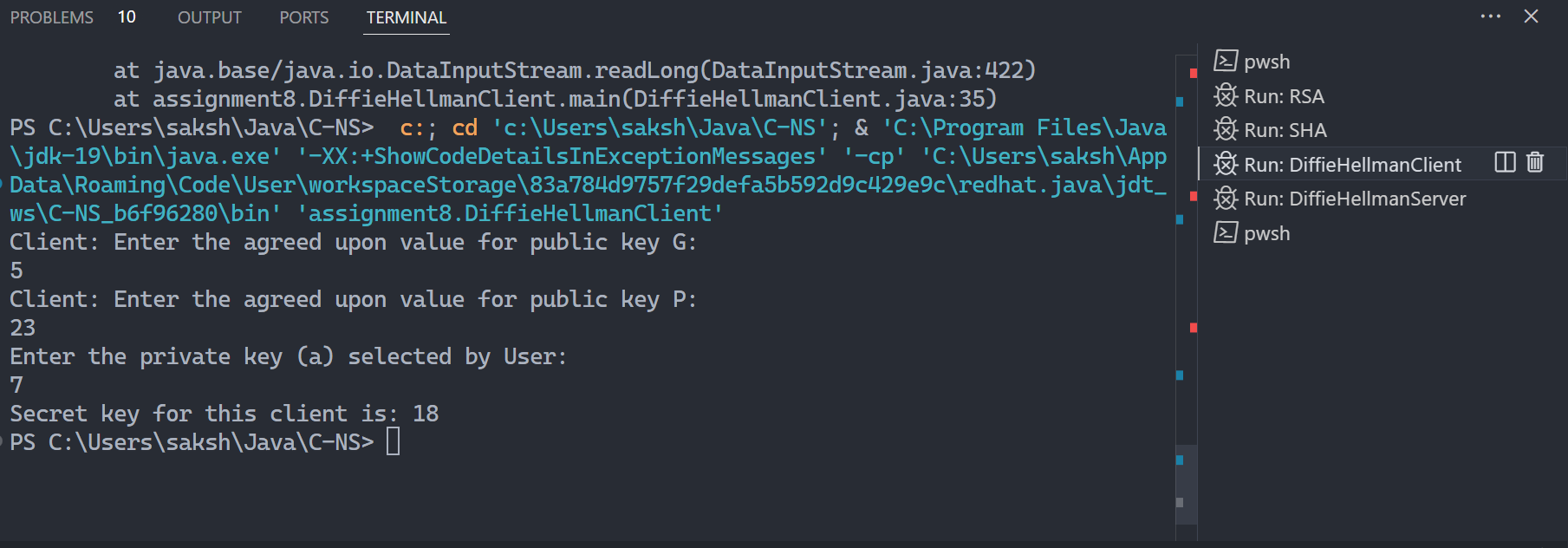
package assignment8;  
  
import java.io.\*;  
import java.net.\*;  
import java.util.Scanner;  
  
public class DiffieHellmanClient {  
 public static void main(String[] args) {  
 long P, G, a, x;  
 Scanner sc = new Scanner(System.*in*);  
  
  
 System.*out*.println("Client: Enter the agreed upon value for public key G:");  
 G = sc.nextLong();  
 System.*out*.println("Client: Enter the agreed upon value for public key P:");  
 P = sc.nextLong();  
  
  
 System.*out*.println("Enter the private key (a) selected by User:");  
 a = sc.nextLong();  
  
  
 x = *calculatePower*(G, a, P);  
  
 try {  
  
 Socket socket = new Socket("localhost", 8080);  
  
  
 DataOutputStream out = new DataOutputStream(socket.getOutputStream());  
 out.writeLong(x);  
  
  
 DataInputStream in = new DataInputStream(socket.getInputStream());  
 long secretKey = in.readLong();  
 System.*out*.println("Secret key for this client is: " + secretKey);  
  
  
 socket.close();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 private static long calculatePower(long base, long exp, long mod) {  
 long result = 1;  
 base = base % mod;  
 while (exp > 0) {  
 if (exp % 2 == 1) {  
 result = (result \* base) % mod;  
 }  
 exp = exp >> 1;  
 base = (base \* base) % mod;  
 }  
 return result;  
 }  
}

OUTPUT :

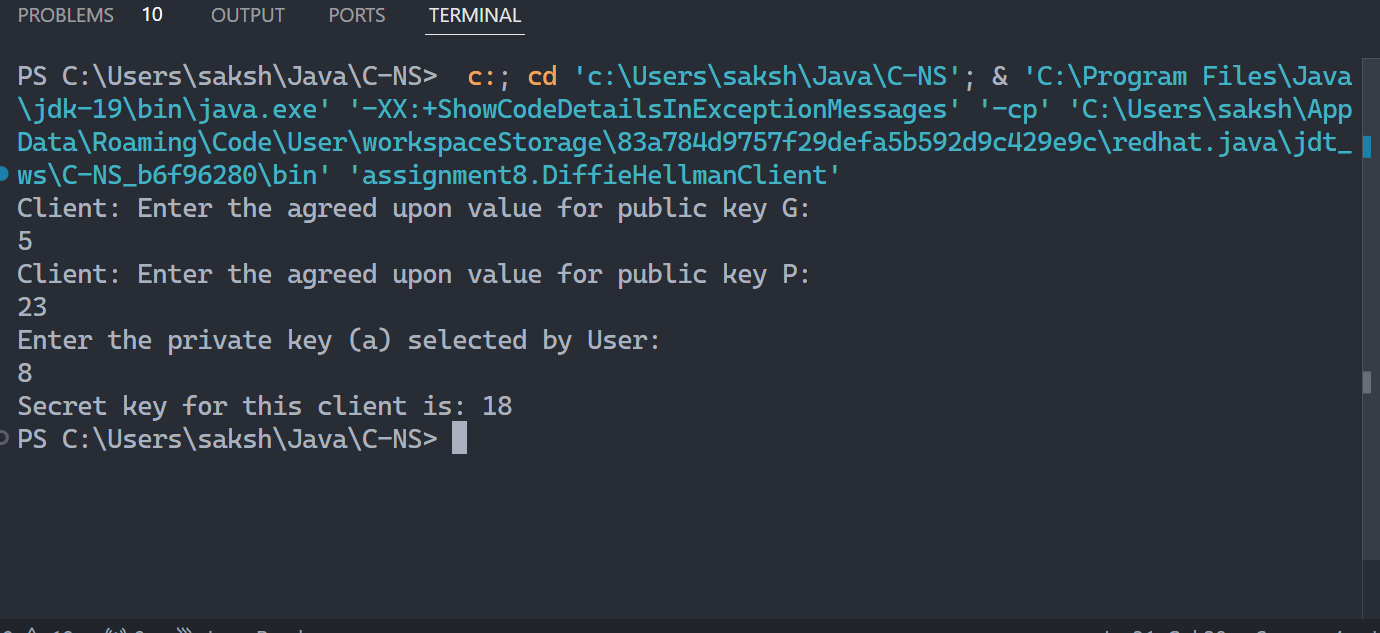
Server :



Client 1 :



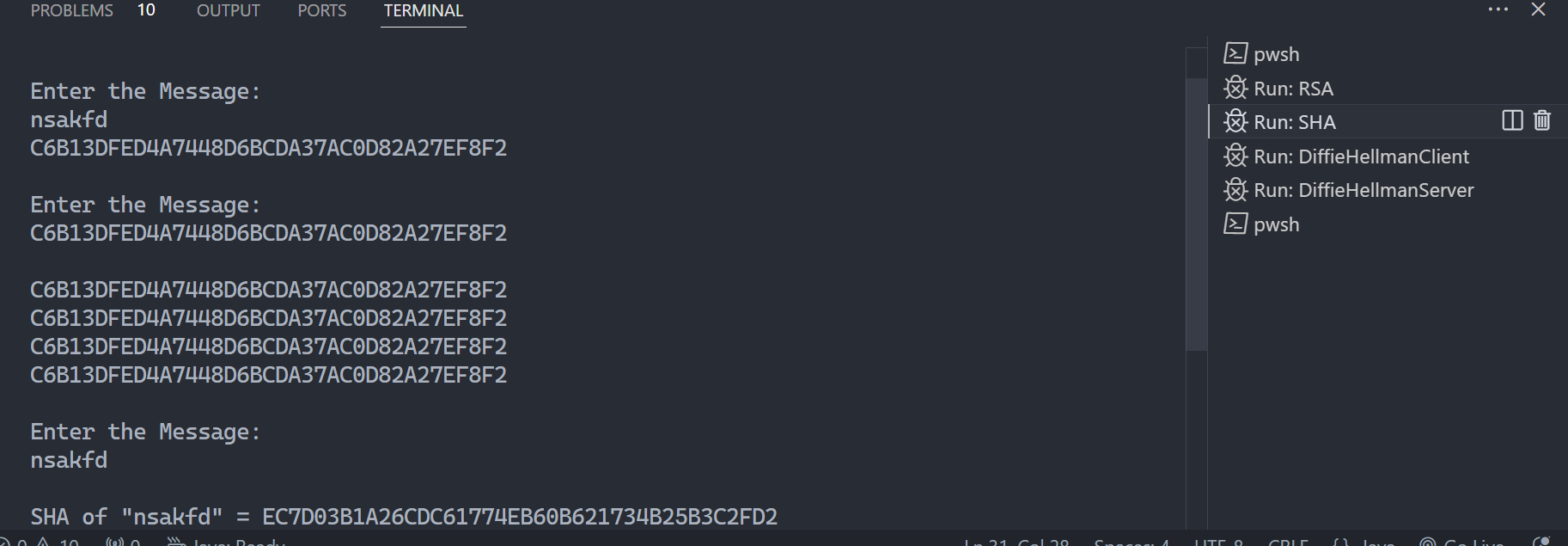
Client 2 :



Assignment 9 : SHA-1 Implementation

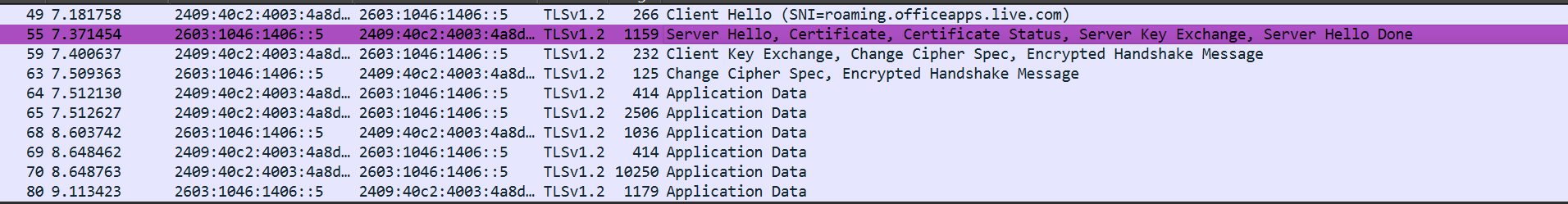
package assignment9;  
  
import java.security.\*;  
import java.util.Scanner;  
  
class SHA {  
 public static void main(String[] a) {  
 Scanner sc= new Scanner(System.*in*);  
  
 try {  
 MessageDigest md = MessageDigest.*getInstance*("SHA1");  
 System.*out*.println("Enter the Message: ");  
 String input = sc.nextLine();  
 md.update(input.getBytes());  
 byte[] output = md.digest();  
 System.*out*.println();  
 System.*out*.println("SHA of \""+input+"\" = " +*bytesToHex*(output));  
 System.*out*.println(" ");  
 System.*out*.println("Enter the Message: ");  
 String input1 = sc.nextLine();  
 md.update(input1.getBytes());  
 output = md.digest();  
 System.*out*.println();  
 System.*out*.println("SHA of \""+input1+"\" = " +*bytesToHex*(output));  
 System.*out*.println(" ");  
 System.*out*.println("Enter the Message: ");  
 String input2 = sc.nextLine();  
 md.update(input2.getBytes());  
 output = md.digest();  
 System.*out*.println();  
 System.*out*.println("SHA of \"" +input2+"\" = " +*bytesToHex*(output));  
 System.*out*.println(""); }  
 catch (Exception e) {  
 System.*out*.println("Exception: " +e);  
 }  
 }  
 public static String bytesToHex(byte[] b) {  
 char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};  
 StringBuffer buf = new StringBuffer();  
 for (int j=0; j<b.length; j++) {  
 buf.append(hexDigit[(b[j] >> 4) & 0x0f]);  
 buf.append(hexDigit[b[j] & 0x0f]); }  
 return buf.toString(); }  
}

OUTPUT :

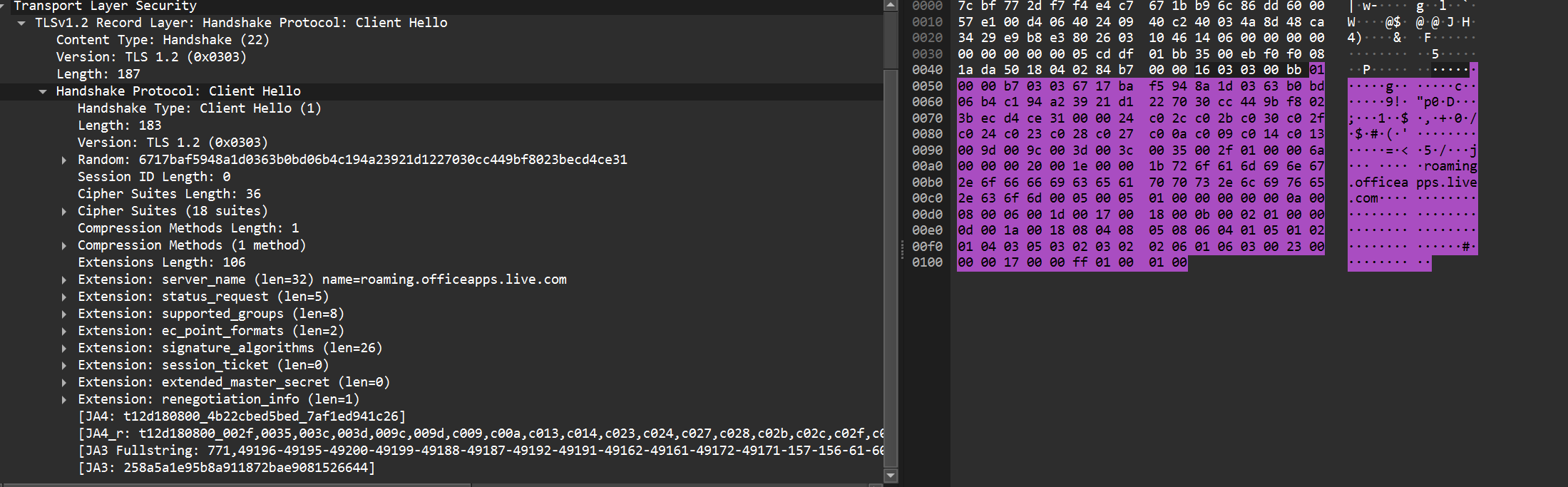


Assignment 10 : SSL through Wireshark

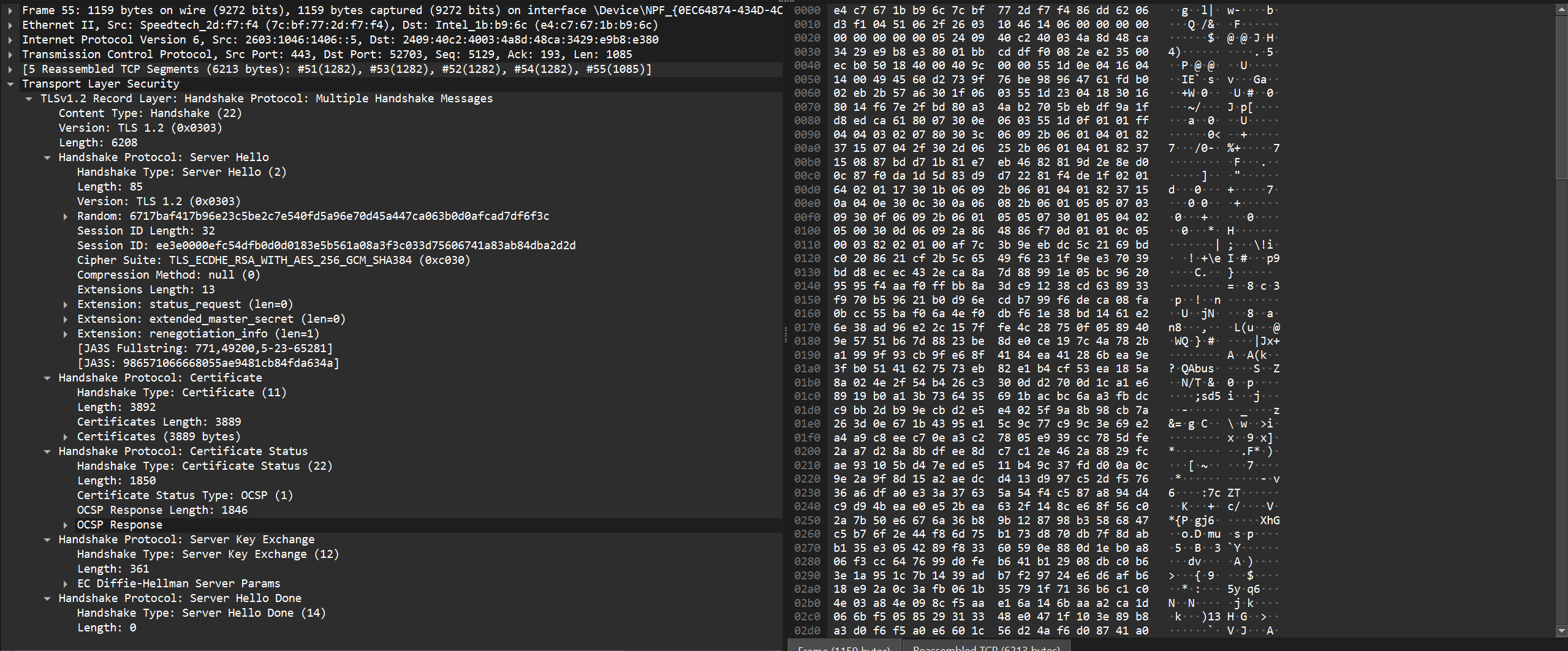
Packet Captured of handshaking :



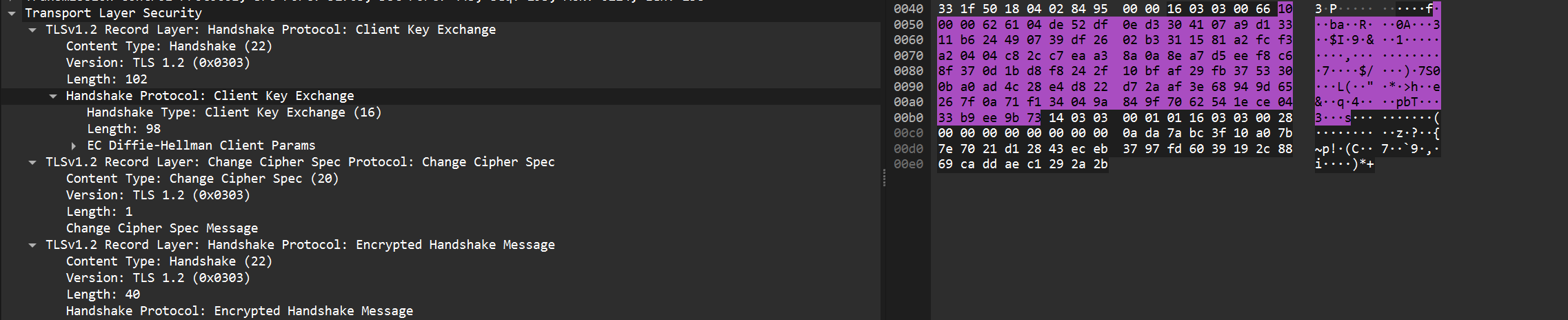
Client side Hello TLS :



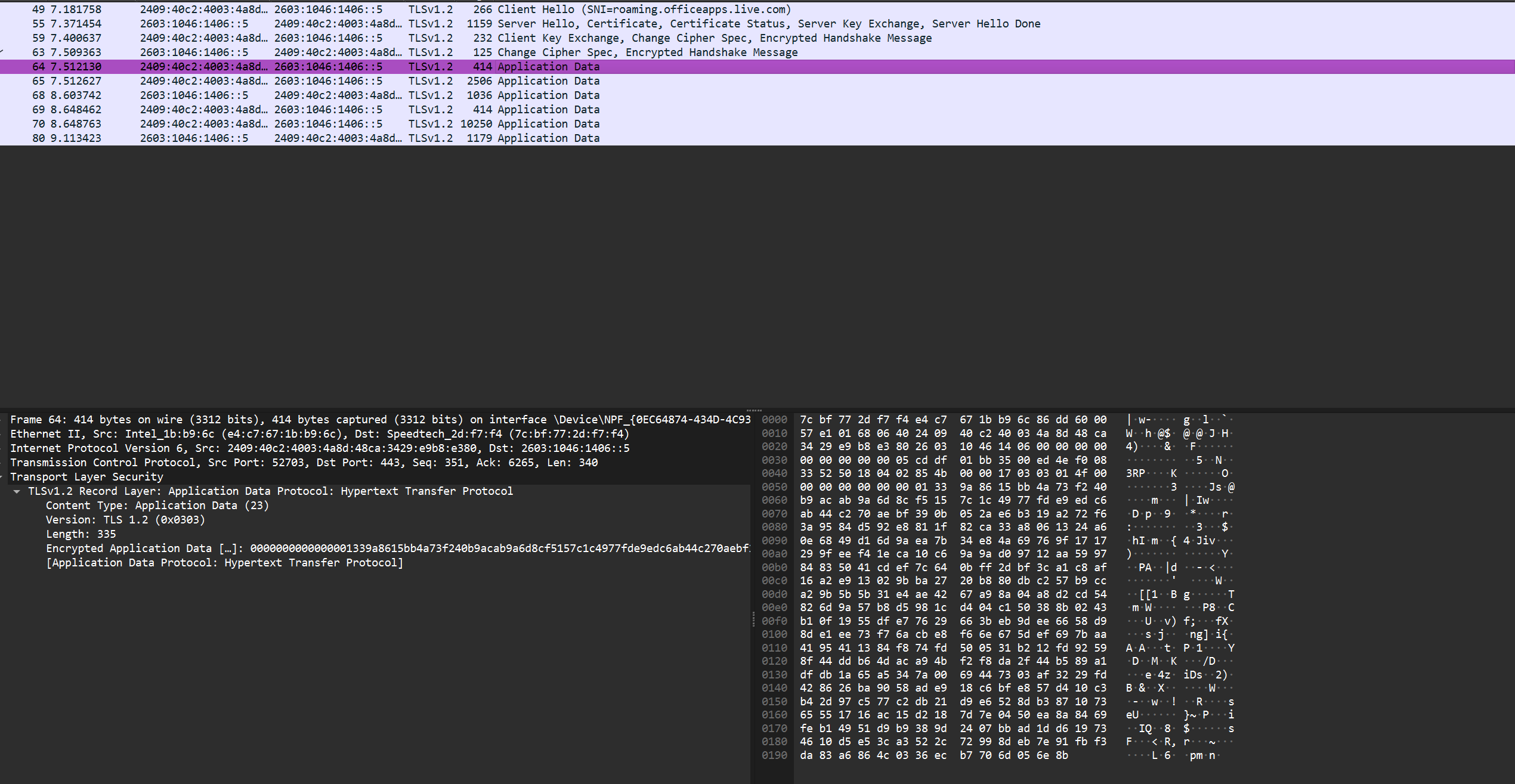
Server side Hello TLS :



Client Key Exchange :

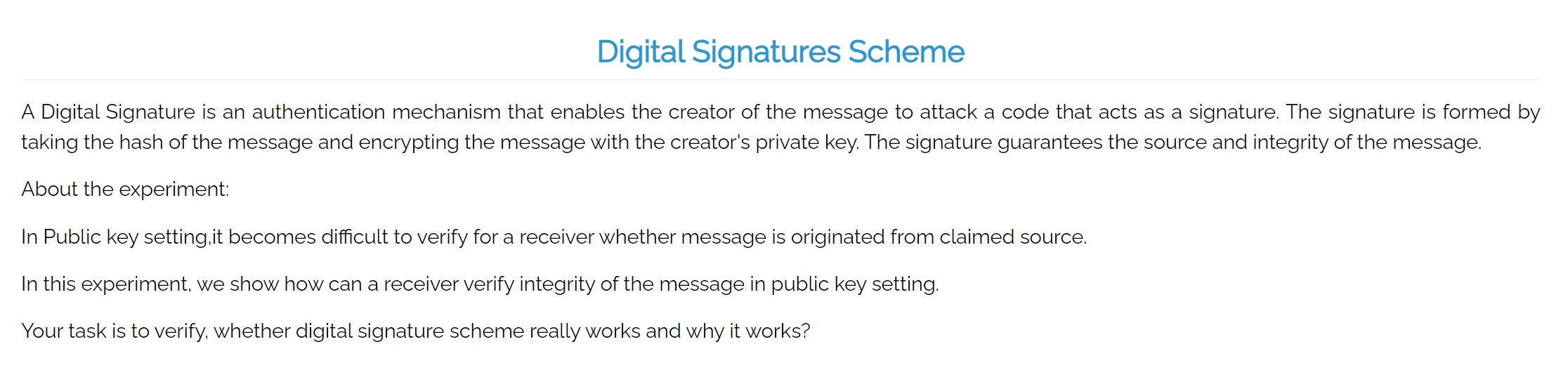


Application Data :

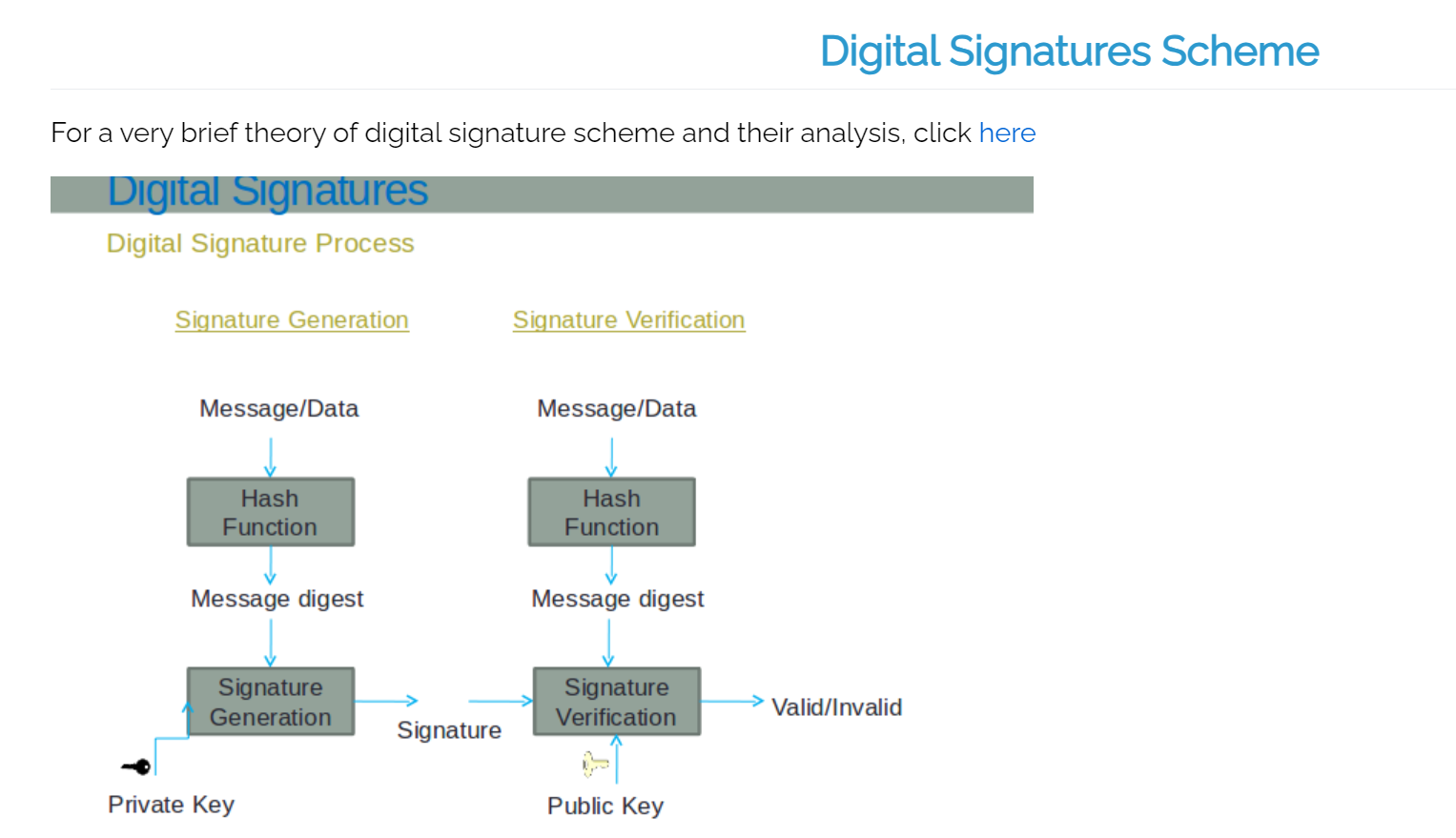


Assignment 11 : Digital Signature V-Lab

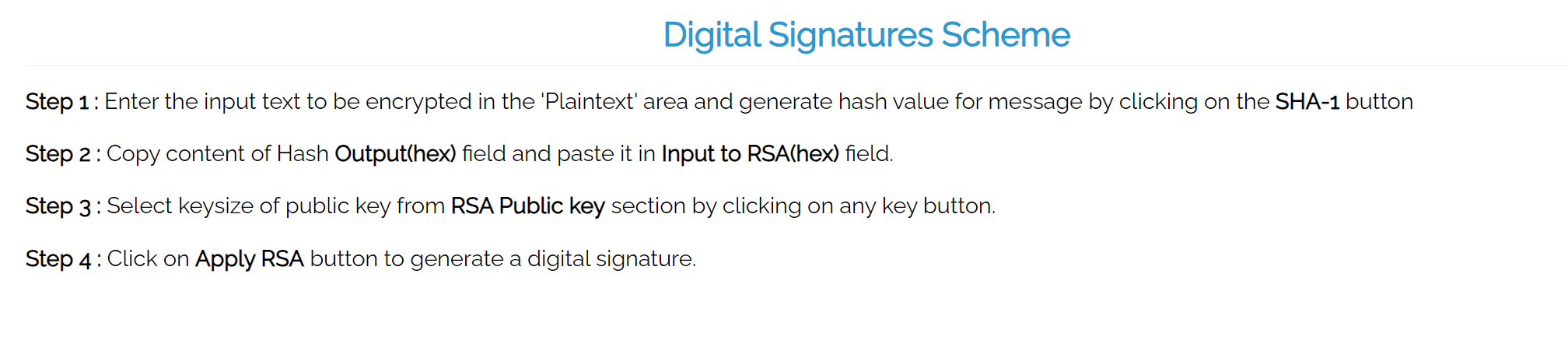
Aim :



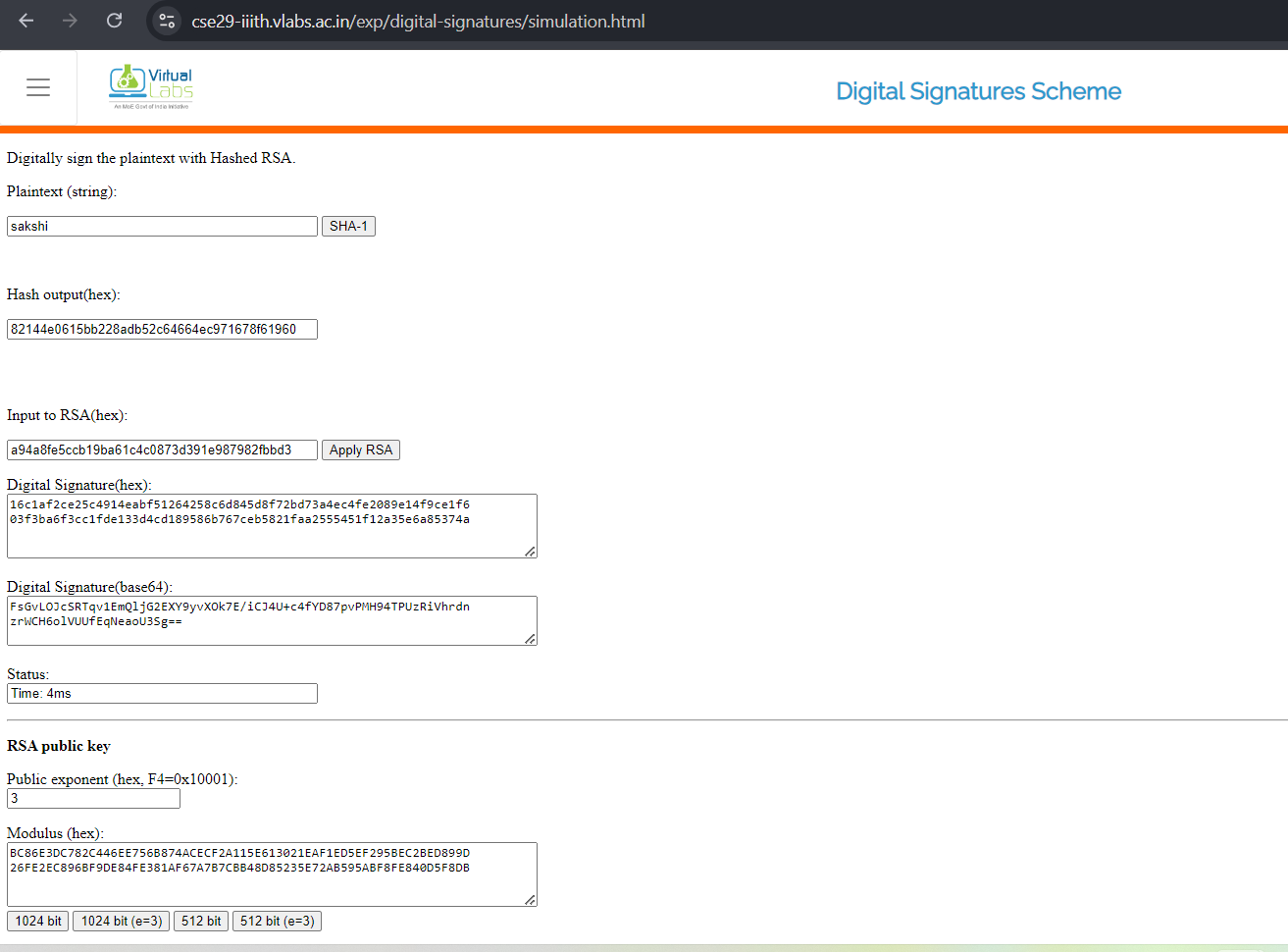
Theory :



Procedure :



Simpuliation :



Assignment :

1. Digital signature can't provide \_\_\_\_\_\_ for the message

(a) Integrity  
(b) Confidentiality  
(c) NOn repudiation  
(d) Authentication

Ans : b) Confidentiality

1. Digital signature uses \_\_\_\_\_\_ for generating valid signature  
   (a) Private key  
   (b) Public key  
   (c) Secret key  
   (d) None of the above

Ans : a) Private key

1. Verification Algorithm uses \_\_\_\_\_\_ for validating digital signature  
   (a) Private key  
   (b) Public key  
   (c) Secret key  
   (d) None of the above

Ans : b) Public key

1. Is digital signature scheme possible without public key cryptography

(a) Yes  
(b) No  
(c) May be exist  
(d) None of the above

Ans : b) No

1. Explain importance of Hashing(using experiment)and explain why Hashing is needed ?

Ans :

**Importance of Hashing**: Hashing plays a crucial role in digital signatures and cryptographic applications by ensuring the integrity of a message. A hash function takes an input (message) and produces a fixed-size string (hash), which is unique for different inputs.

**Experiment**:

1. Take a file or message and generate its hash value using a hash function (e.g., SHA-256).
2. Modify the message slightly (even changing one letter) and generate the hash again.
3. You will notice that even a tiny change in the message results in a completely different hash value. This property is called the **avalanche effect**.

**Why Hashing is Needed**:

* **Integrity**: Hashing ensures that any modification in the message can be detected. Even a small change in the input will produce a drastically different hash.
* **Efficiency**: Hashing converts a message into a short, fixed-length representation (hash), which can then be signed instead of signing the entire message, saving computational resources.
* **Uniqueness**: A good cryptographic hash function generates a unique output for different inputs, which helps in message authentication.

1. Suggest a scheme that does not use any hashing scheme

Ans :

One such scheme is the **RSA Digital Signature** scheme without hashing. In this scheme, the entire message is encrypted with the private key to generate a signature, and the public key is used for verification. However, this approach is inefficient, as it requires encrypting and decrypting the entire message, which is computationally expensive.

1. Explain why digital signature schemes works ? //non repudiation and confidentiality

Ans :

Digital signature schemes work based on **asymmetric cryptography** using a pair of keys: a **private key** (kept secret) and a **public key** (shared with everyone). Here's how it works:

1. **Signing**: The sender generates a hash of the message and then encrypts the hash with their private key. This encrypted hash forms the digital signature.
2. **Verification**: The receiver uses the sender’s public key to decrypt the signature and obtain the original hash. The receiver also generates the hash of the received message. If both hashes match, the message is authentic, unaltered, and came from the sender.

**Why it works**:

* **Authenticity**: Only the sender could have created the signature, as only they have the private key.
* **Integrity**: If the message is altered, the hashes won’t match.
* **Non-repudiation**: The sender cannot deny sending the message, as only their private key could have generated the signature.