Cryptography and Network Security

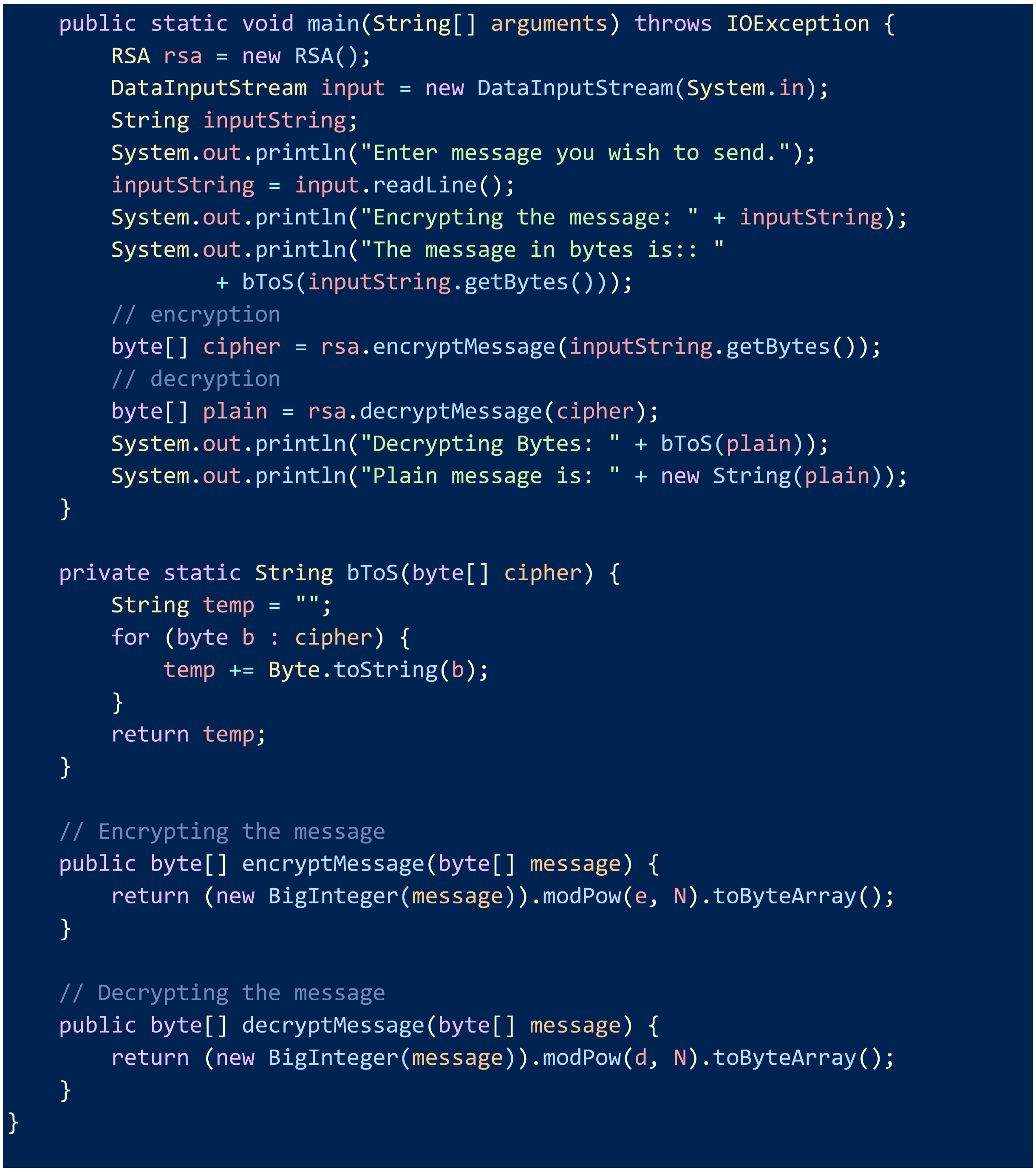
LA-2

Name	**:**	**Sourabh Shankar Patil**
PRN	**:**	**21510045**
Batch	**:**	**B2**

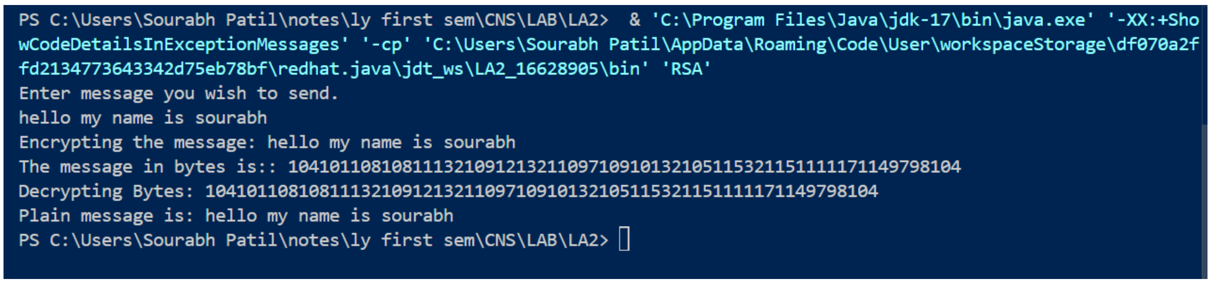
Assignment no 07

RSA:

```java
import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;

public class 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 RSA(BigInteger e, BigInteger d, BigInteger N) {
        this.e = e;
        this.d = d;
        this.N = N;
    }
```



```java
    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());
        // 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();
    }
}
```



```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\LA2>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+Sho
wCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Sourabh Patil\AppData\Roaming\Code\User\workspaceStorage\df070a2f
fd2134773643342d75eb78bf\redhat.java\jdt_ws\LA2_16628905\bin' 'RSA'
Enter message you wish to send.
hello my name is sourabh
Encrypting the message: hello my name is sourabh
The message in bytes is:: 104101108108111321091213211097109101321051153211511111171149798104
Decrypting Bytes: 104101108108111321091213211097109101321051153211511111171149798104
Plain message is: hello my name is sourabh
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\LA2> 
```

Explanation-

The provided code is an implementation of the **RSA encryption** algorithm in Java. RSA (Rivest-Shamir-Adleman) is a public-key encryption algorithm used to securely transmit messages over the internet. It involves generating two keys: a public key for encryption and a private key for decryption. Below is an explanation of each part of the code:

1. Class Variables:

- P and Q: Two large prime numbers generated randomly.
- N: The product of P and Q (N = P * Q), used as part of the public and private keys.
- PHI: The totient of N, calculated as PHI = (P - 1) * (Q - 1). It's required to generate the public and private keys.
- e: The public exponent (part of the public key). It is a random number chosen such that gcd(e, PHI) = 1 and e < PHI.
- d: The private exponent (part of the private key). It is the modular inverse of e modulo PHI, such that d * e ≡ 1 (mod PHI).
- maxLength: The bit-length used to generate the prime numbers P and Q. Here, the value is set to 1024 bits.
- R: A Random object used to generate random prime numbers.

2. Constructor:

There are two constructors:

- **Default Constructor RSA()**:
    - Randomly generates two large primes P and Q.
    - Computes N = P * Q and PHI = (P - 1) * (Q - 1).
    - Selects a random public exponent e such that gcd(e, PHI) = 1 (i.e., e is coprime to PHI).
    - Calculates the private key exponent d using the formula d = e^(-1) mod PHI, which is the modular inverse of e modulo PHI.
- **Parameterized Constructor RSA(BigInteger e, BigInteger d, BigInteger N)**:
    - Initializes an RSA object with predefined public key components (e and N) and private key (d).

3. Main Method:

The main method allows a user to input a message, encrypt it, and then decrypt it back to the original message:

- The message is entered by the user via DataInputStream.
- The message is encrypted using the encryptMessage method.
- The encrypted message is then decrypted back to the original form using the decryptMessage method.
- Both the encrypted and decrypted messages are printed.

4. Helper Methods:

- **bToS(byte[] cipher)**: Converts a byte array (ciphertext or plaintext) to a string for easy display by converting each byte to a string and appending it to temp.

5. Encryption (encryptMessage):

Encrypts a message using the formula:

ciphertext=messageemod N\text{ciphertext} = \text{message}^e \mod Nciphertext=messageemodN

Here, the message is treated as a BigInteger, raised to the power of e (public exponent), and taken modulo N.

6. Decryption (decryptMessage):

Decrypts a message using the formula:

plaintext=ciphertextdmod N\text{plaintext} = \text{ciphertext}^d \mod Nplaintext=ciphertextdmodN

Here, the ciphertext is raised to the power of d (private exponent), and taken modulo N to retrieve the original plaintext message.

7. Important Concepts:

- **Public Key (e, N)**: Used to encrypt messages.
- **Private Key (d, N)**: Used to decrypt messages.
- **Modular Arithmetic**: Both encryption and decryption use modular exponentiation (modPow) to ensure messages can be encrypted and decrypted efficiently, even for large numbers.

8. Execution Flow:

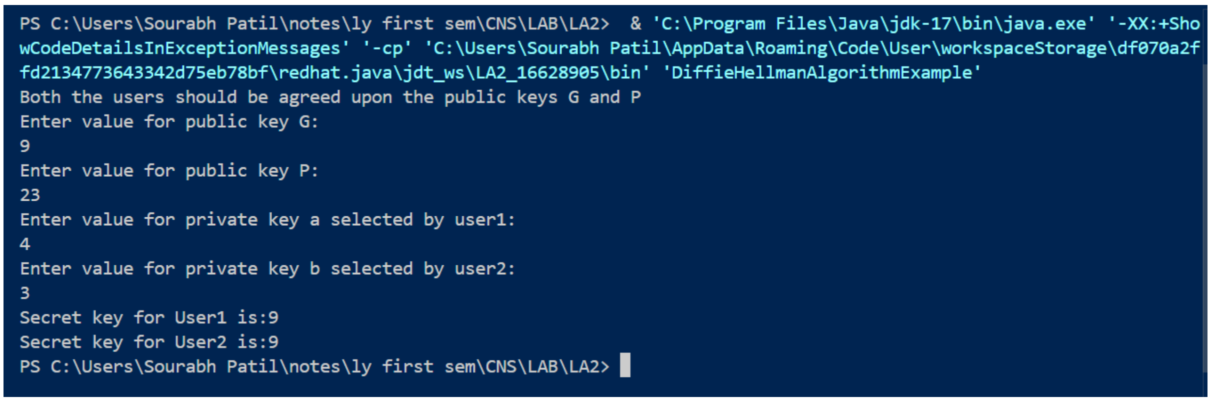
1. RSA object is created using the default constructor (new RSA()).
2. The user inputs a message.
3. The message is converted to a byte array and encrypted.
4. The encrypted byte array (ciphertext) is decrypted back to the original message.

This code is a basic demonstration of how RSA encryption and decryption work. It's not production-ready, as additional security measures (such as padding and key management) are typically required for real-world applications.

Assignment no 08

DiffieHellmanAlgorithm:

```
import java.util.Scanner;

public class DiffieHellmanAlgorithmExample {
    public DiffieHellmanAlgorithmExample() {
    }

    public static void main(String[] var0) {
        try (Scanner var17 = new Scanner(System.in)) {
            System.out.println("Both the users should be agreed upon the
public keys G and P");
            System.out.println("Enter value for public key G:");
            long var3 = var17.nextLong();
            System.out.println("Enter value for public key P:");
            long var1 = var17.nextLong();
            System.out.println("Enter value for private key a selected by
user1:");
            long var7 = var17.nextLong();
            System.out.println("Enter value for private key b selected by
user2:");
            long var11 = var17.nextLong();
            long var5 = calculatePower(var3, var7, var1);
            long var9 = calculatePower(var3, var11, var1);
            long var13 = calculatePower(var9, var7, var1);
            long var15 = calculatePower(var5, var11, var1);
            System.out.println("Secret key for User1 is:" + var13);
            System.out.println("Secret key for User2 is:" + var15);
        }
    }

    private static long calculatePower(long var0, long var2, long var4) {
        long var6 = 0L;
        if (var2 == 1L) {
            return var0;
        } else {
            var6 = (long) Math.pow((double) var0, (double) var2) % var4;
            return var6;
        }
    }
}
```


```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\LA2>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Sourabh Patil\AppData\Roaming\Code\User\workspaceStorage\df070a2ffd2134773643342d75eb78bf\redhat.java\jdt_ws\LA2_16628905\bin' 'DiffieHellmanAlgorithmExample'
Both the users should be agreed upon the public keys G and P
Enter value for public key G:
9
Enter value for public key P:
23
Enter value for private key a selected by user1:
4
Enter value for private key b selected by user2:
3
Secret key for User1 is:9
Secret key for User2 is:9
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\LA2>
```

T
his Java code demonstrates the **Diffie-Hellman Key Exchange Algorithm**, a method used for securely exchanging cryptographic keys over a public channel. The two parties, User1 and User2, agree on public values and then each selects a private value to compute a shared secret key.

Explanation of the Code:

1. **Public Keys:**
   - P: A prime number chosen publicly.
   - G: A base (primitive root modulo P), another public value.
2. **Private Keys:**
   - a: Private key selected by **User1**.
   - b: Private key selected by **User2**.
3. **Intermediate Values:**
   - x: Calculated by User1 as Gamod PG^a \mod PGamodP, then sent to User2.
   - y: Calculated by User2 as Gbmod PG^b \mod PGbmodP, then sent to User1.
4. **Shared Secret Keys:**
   - After exchanging the intermediate values (x and y):
     - User1 calculates the shared key ka as yamod Py^a \mod PyamodP (using the value y received from User2).
     - User2 calculates the shared key kb as xbmod Px^b \mod PxbmodP (using the value x received from User1).

Code Walkthrough:

1. **User Input for Public and Private Keys**:
   - P (a prime number) and G (a generator) are public keys, agreed upon by both users.
   - a (User1's private key) and b (User2's private key) are input by the users.
2. **Calculation of Intermediate Keys**:
   - x = G^a mod P: This value is sent from User1 to User2.
   - y = G^b mod P: This value is sent from User2 to User1.
3. **Calculation of Secret Keys**:
   - User1 calculates their shared secret ka = y^a mod P (using User2's intermediate value y).
   - User2 calculates their shared secret kb = x^b mod P (using User1's intermediate value x).
4. **Output**:

  - Both users will have the same secret key (ka == kb), as the Diffie-Hellman algorithm ensures both parties derive the same shared secret.

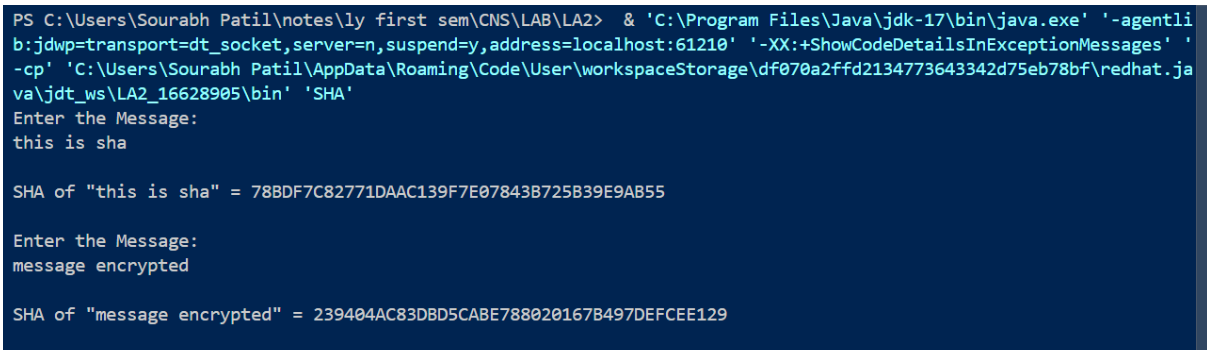
Key Methods:

- **calculatePower(long x, long y, long P)**: This method computes xymod Px^y \mod PxymodP. If y equals 1, it directly returns x. Otherwise, it computes the power using Math.pow() and then applies modulo P.

Assignment no 09

SHA:

```
import java.security.MessageDigest;
import java.util.Scanner;

public class SHA {
    public static void main(String[] a) {
        try (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();
    }
}
```



```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\LA2>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-agentli
b:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:61210' '-XX:+ShowCodeDetailsInExceptionMessages' '
-cp' 'C:\Users\Sourabh Patil\AppData\Roaming\Code\User\workspaceStorage\df070a2ffd2134773643342d75eb78bf\redhat.ja
va\jdt_ws\LA2_16628905\bin' 'SHA'
Enter the Message:
this is sha

SHA of "this is sha" = 78BDF7C82771DAAC139F7E07843B725B39E9AB55

Enter the Message:
message encrypted

SHA of "message encrypted" = 239404AC83DBD5CABE788020167B497DEFCEE129
```

Explanation-

This Java program implements **SHA-1 hashing** using the MessageDigest class from the java.security package. It allows the user to input multiple messages, generates their SHA-1 hash values, and outputs them in hexadecimal format.

Explanation of the Code:

1. **Importing Libraries**:
    - java.security.*: Provides classes such as MessageDigest for cryptographic hash functions.
    - java.util.Scanner: To take user input from the console.
2. **Main Class (SHA)**:
    - The main method prompts the user to input a message, computes its SHA-1 hash, and prints the result.
3. **Key Concepts**:
    - **SHA-1 (Secure Hash Algorithm 1)**: A cryptographic hash function that takes an input and produces a 160-bit (20-byte) hash value. It is no longer considered secure for cryptographic purposes but is still used in legacy systems.
    - **MessageDigest**: Used to compute the SHA-1 hash of input messages.

Code Walkthrough:

1. **User Input**:
    - A Scanner object is created to read input messages from the user.
2. **Message Digest Initialization**:
    - The MessageDigest.getInstance("SHA1") method creates a MessageDigest object initialized to use the SHA-1 hashing algorithm.

3. **Hashing Process**:
    - The md.update(input.getBytes()) method feeds the user's input (converted to bytes) into the digest algorithm.
    - md.digest() computes the final hash value for the input message, returning it as a byte array.
4. **Hexadecimal Conversion**:
    - The bytesToHex(byte[] b) method converts the resulting byte array into a hexadecimal string for easy readability.
5. **Multiple Inputs**:
    - The program allows the user to input three separate messages, calculates the SHA-1 hash for each, and prints the results.

Example Output:

mathematica

Copy code

```
Enter the Message:
Hello World

SHA of "Hello World" = 2EF7BDE608CE5404E97D5F042F95F89F1C232871

Enter the Message:
Test

SHA of "Test" = A94A8FE5CCB19BA61C4C0873D391E987982FBBD3

Enter the Message:
Java Programming

SHA of "Java Programming" = 12E1215D4C9A3E8B3FFBE2E9F9939B642AA4290D
```

Key Method:

- **bytesToHex(byte[] b)**:
    - This method converts a byte array into a hexadecimal string.
    - The byte values are processed in two steps:

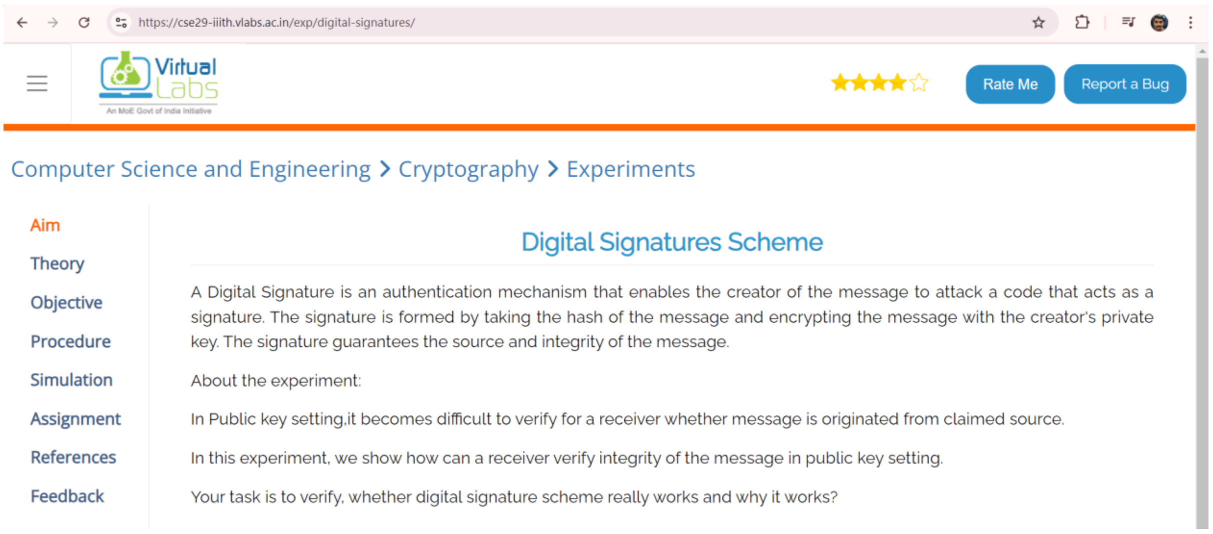
    - (b[j] >> 4) & 0x0f extracts the higher 4 bits (hex digit).
    - b[j] & 0x0f extracts the lower 4 bits (hex digit).
  - The hexadecimal characters are appended to a StringBuffer for efficient string concatenation.

Improvements:

1. **Use StringBuilder**:
   - Replace StringBuffer with StringBuilder in bytesToHex for better performance since synchronization is not needed in this case.
2. **SHA-256 Update**:
   - Since SHA-1 is deprecated in security-critical contexts, you could update the program to use SHA-256 by replacing "SHA1" with "SHA-256" in MessageDigest.getInstance().

Assignment no 10

1]Aim:


https://cse29-iiith.vlabs.ac.in/exp/digital-signatures/

Digital Signatures Scheme

A Digital Signature is an authentication mechanism that enables the creator of the message to attack a code that acts as a signature. The signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees the source and integrity of the message.

About the experiment:

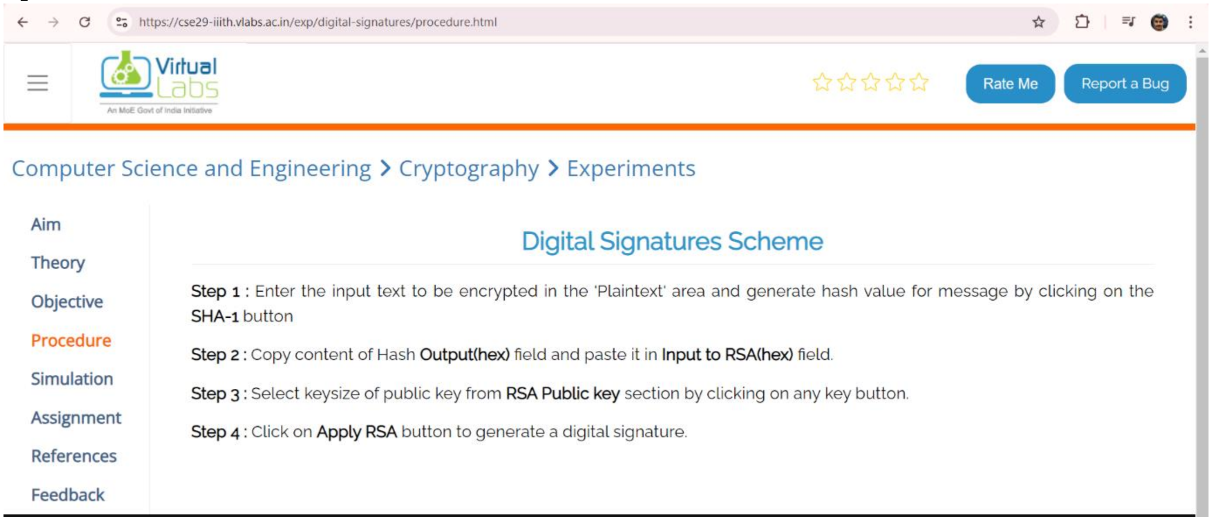
In Public key setting,it becomes difficult to verify for a receiver whether message is originated from claimed source.

In this experiment, we show how can a receiver verify integrity of the message in public key setting.

Your task is to verify, whether digital signature scheme really works and why it works?

2]Objective: To understand "How and Why Digital signature schemes?"

3]Procedure:


https://cse29-iiith.vlabs.ac.in/exp/digital-signatures/procedure.html

Digital Signatures Scheme

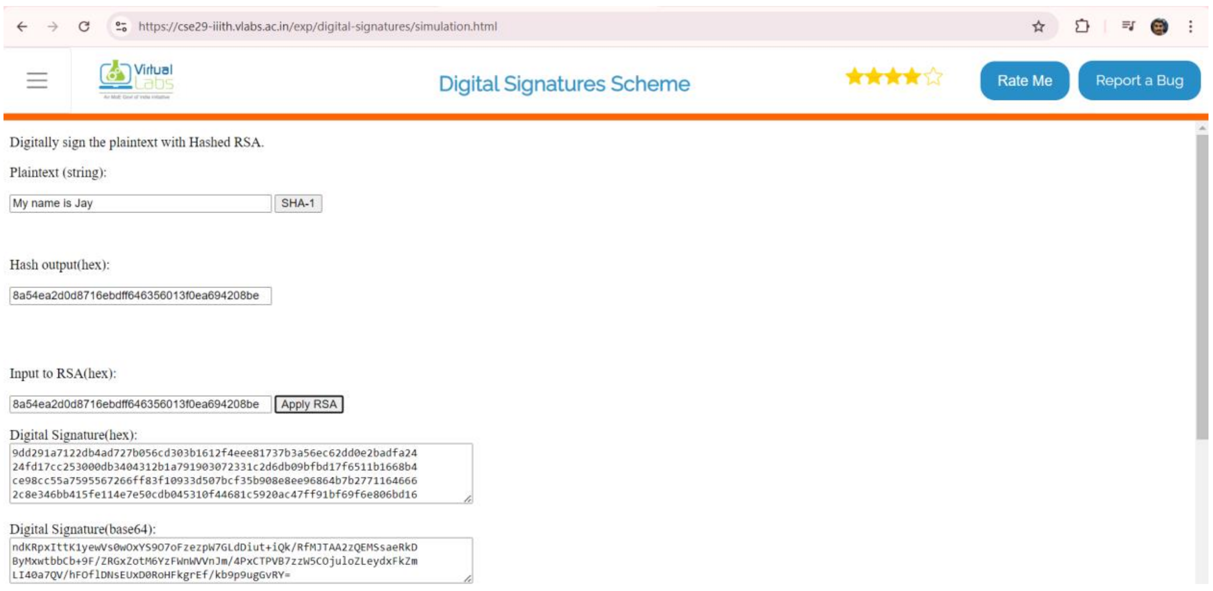
Step 1 : Enter the input text to be encrypted in the 'Plaintext' area and generate hash value for message by clicking on the **SHA-1** button

Step 2 : Copy content of Hash **Output(hex)** field and paste it in **Input to RSA(hex)** field.

Step 3 : Select keysize of public key from **RSA Public key** section by clicking on any key button.

Step 4 : Click on **Apply RSA** button to generate a digital signature.

4]Simulation:


https://cse29-iiith.vlabs.ac.in/exp/digital-signatures/simulation.html
Virtual Labs
Digital Signatures Scheme
Rate Me
Report a Bug
SHA-1
Apply RSA

Digitally sign the plaintext with Hashed RSA.

Plaintext (string):

My name is Jay

Hash output(hex):

8a54ea2d0d8716ebdff646356013f0ea694208be

Input to RSA(hex):

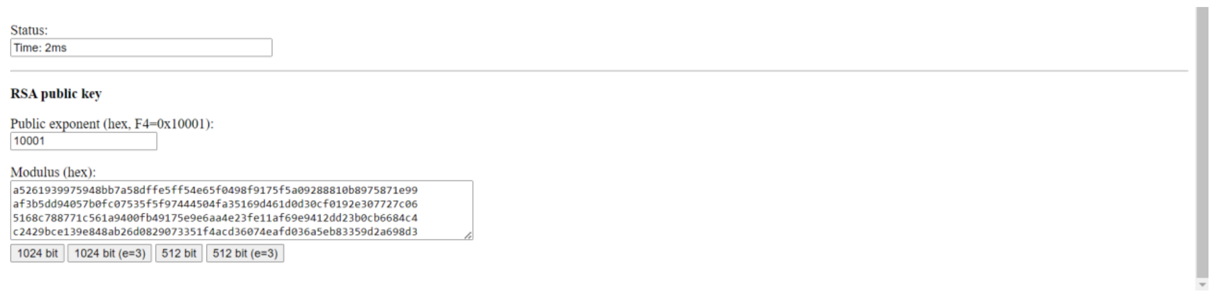
8a54ea2d0d8716ebdff646356013f0ea694208be

Digital Signature(hex):

```
9dd291a7122db4ad727b056cd303b1612f4eee81737b3a56ec62dd0e2badfa24
24fd17cc253000db3404312b1a791903072331c2d6db09bfbd17f6511b1668b4
ce98cc55a7595567266ff83f10933d507bcf35b908e8ee96864b7b2771164666
2c8e346bb415fe114e7e50cdb045310f44681c5920ac47ff91bf69f6e806bd16
```

Digital Signature(base64):

```
ndKRpxIttK1yewVs0wOxYS9O7oFzezpW7GLdDiut+iQk/RfMJTAA2zQEMSsaeRkD
ByMxwtbbCb+9F/ZRGxZotM6YzFWnWVVnJm/4PxCTPVB7zzW5COjuloZLeydxFkZm
LI40a7QV/hFOflDNsEUxD0RoHFkgrEf/kb9p9ugGvRY=
```



Status:

Time: 2ms

RSA public key

Public exponent (hex, F4=0x10001):

10001

Modulus (hex):

```
a5261939975948bb7a58dffe5ff54e65f0498f9175f5a09288810b8975871e99
af3b5dd94057b0fc07535f5f97444504fa35169d461d0d30cf0192e307727c06
5168c788771c561a9400fb49175e9e6aa4e23fe11af69e9412dd23b0cb6684c4
c2429bce139e848ab26d0829073351f4acd36074eafd036a5eb83359d2a698d3
```

Explanation-

This program demonstrates how to create and verify a **digital signature** using the RSA algorithm and the SHA-256 hash function in Java. Here's an explanation of each part:

Explanation:

1. **Imports**:
   - The program uses various security-related classes (KeyPair, Signature, PrivateKey, PublicKey, etc.) from the java.security package.
   - HexFormat: From java.util (Java 17+), it provides utility methods for converting bytes into hexadecimal strings for easy readability.
2. **Key Concepts**:
   - **Digital Signature**: A way to verify the authenticity and integrity of a message. It involves generating a signature using a private key, which can be verified using the corresponding public key.
   - **RSA Algorithm**: A widely used asymmetric encryption algorithm where a pair of keys (public and private) is used for signing and verifying.

- **SHA-256 with RSA**: The hashing algorithm (SHA-256) is used to hash the input message, and RSA is used to sign the hash.

Code Breakdown:

1. **Constants**:
   - SIGNING_ALGORITHM: Specifies the cryptographic signing algorithm to be used (SHA256withRSA).
   - RSA: Specifies the RSA algorithm for key pair generation.
2. **Create_Digital_Signature Method**:
   - This method takes a byte[] input and a PrivateKey as parameters.
   - It initializes a Signature object with the specified algorithm (SHA256withRSA), signs the data using the private key, and returns the generated signature as a byte array.
3. **Generate_RSA_KeyPair Method**:
   - It generates a new RSA key pair (private and public key).
   - A SecureRandom object is used to ensure cryptographically strong randomness.
   - The key size is set to 2048 bits for adequate security.
4. **Verify_Digital_Signature Method**:
   - This method takes the input data, the signature to verify, and the public key.
   - It initializes the Signature object for verification using the public key.
   - It then verifies whether the provided signature matches the one generated by the Create_Digital_Signature method.
5. **main Method**:
   - The main method generates an RSA key pair.
   - It signs a string message "Hello I am Sujan" using the private key.
   - The generated signature is displayed in hexadecimal format for easy reading.
   - The program then verifies the signature using the corresponding public key and prints the verification result (true or false).

Detailed Example:

1. **Key Pair Generation**:
   - A new RSA key pair is generated using KeyPairGenerator with a key size of 2048 bits. The key pair consists of a private key (for signing) and a public key (for verification).
2. **Signing Process**:

    - The input message ("Hello I am Sujan") is signed using the private key. This signature ensures that the message comes from the owner of the private key and that the message has not been tampered with.
    - The output is printed in hexadecimal format using the HexFormat.of().formatHex(signature) function.

3. **Verification Process**:
    - The program verifies whether the message matches the generated signature using the public key. If the verification is successful, it prints true, indicating that the signature is valid.

Example Output:

yaml

Copy code

```
Signature Value:

3046022100e3b2cf5a5f9f54f89ba8e19364fa2fe92420a48c50b15479ab47a6bc9f11ab0220221
00b08f5b988d9fb6ae9c91f3a8ddf74fe40293d7a95a2162d51b484bdb30d09

Verification: true
```

Key Classes Used:

- **KeyPairGenerator**: Generates a public-private key pair.
- **Signature**: Used for both signing and verifying digital signatures.
- **HexFormat**: Converts byte arrays into a human-readable hexadecimal format.

Improvements:

1. **Exception Handling**:
    - The code can be improved by adding more detailed exception handling to catch specific errors related to key generation, signing, or verification.
2. **User Input**:
    - To make the program more dynamic, it can be extended to take user input for the message instead of hardcoding the message in the program.

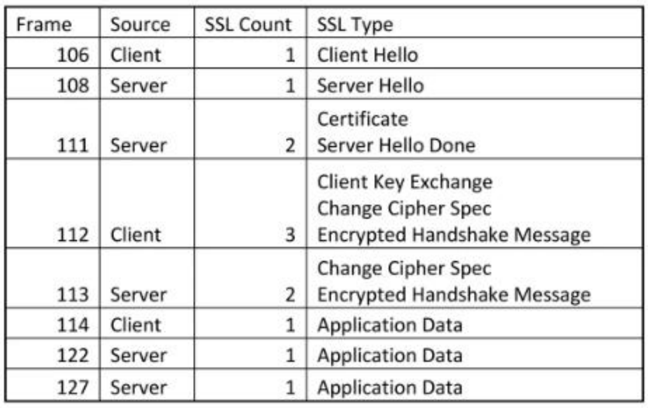
With these key concepts, this program demonstrates how digital signatures can be used to verify the integrity and authenticity of messages.

Assignment no 11

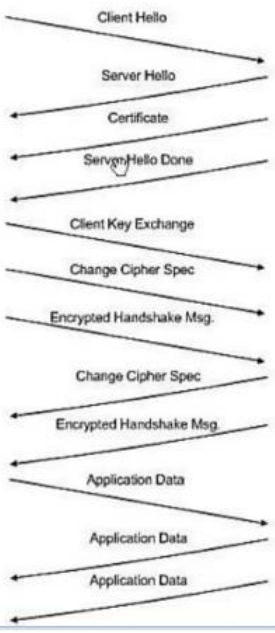
SSL using Wireshark packets

Q1] For each of the first 8 Ethernet frames, specify the source of the frame (client or server), determine the number of SSL records that are included in the frame, and list the SSL record types that are included in the frame. Draw a timing diagram between client and server, with one arrow for each SSL record.

Ans-



Frame	Source	SSL Count	SSL Type
106	Client	1	Client Hello
108	Server	1	Server Hello
111	Server	2	Certificate Server Hello Done
112	Client	3	Client Key Exchange Change Cipher Spec Encrypted Handshake Message
113	Server	2	Change Cipher Spec Encrypted Handshake Message
114	Client	1	Application Data
122	Server	1	Application Data
127	Server	1	Application Data



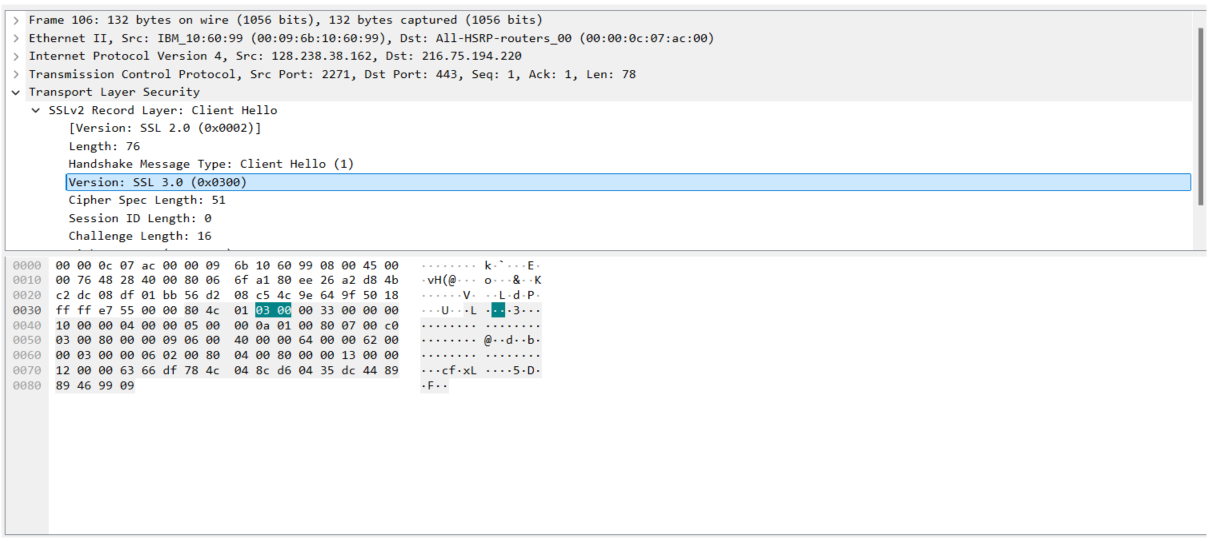
Q2] Each of the SSL records begins with the same three fields (with possibly different values). One of these fields is "content type" and has length of one byte. List all three fields and their lengths.

Ans-

Content Type : 1 byte

Version : 2bytes

Length : 2bytes

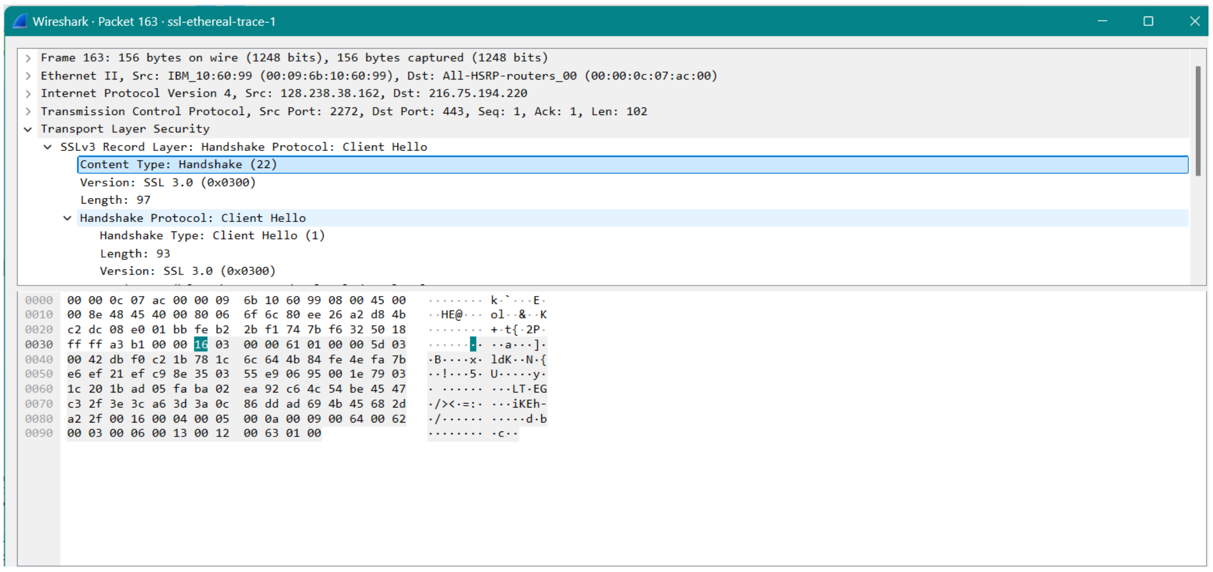


```
> Frame 106: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits)
> Ethernet II, Src: IBM_10:60:99 (00:09:6b:10:60:99), Dst: All-HSRP-routers_00 (00:00:0c:07:ac:00)
> Internet Protocol Version 4, Src: 128.238.38.162, Dst: 216.75.194.220
> Transmission Control Protocol, Src Port: 2271, Dst Port: 443, Seq: 1, Ack: 1, Len: 78
v Transport Layer Security
  v SSLv2 Record Layer: Client Hello
      [Version: SSL 2.0 (0x0002)]
      Length: 76
      Handshake Message Type: Client Hello (1)
      Version: SSL 3.0 (0x0300)
      Cipher Spec Length: 51
      Session ID Length: 0
      Challenge Length: 16
```

```
0000  00 00 0c 07 ac 00 00 09  6b 10 60 99 08 00 45 00   ········ k·`···E·
0010  00 76 48 28 40 00 80 06  6f a1 80 ee 26 a2 d8 4b   ·vH(@··· o···&··K
0020  c2 dc 08 df 01 bb 56 d2  08 c5 4c 9e 64 9f 50 18   ······V· ··L·d·P·
0030  ff ff e7 55 00 00 80 4c  01 03 00 00 33 00 00 00   ···U···L ···3···
0040  10 00 00 04 00 00 05 00  00 0a 01 00 80 07 00 c0   ········ ········
0050  03 00 80 00 00 09 06 00  40 00 00 64 00 00 62 00   ········ @··d··b·
0060  00 03 00 00 06 02 00 80  04 00 80 00 00 13 00 00   ········ ········
0070  12 00 00 63 66 df 78 4c  04 8c d6 04 35 dc 44 89   ···cf·xL ····5·D·
0080  89 46 99 09                                        ·F··
```

Q3] Expand the ClientHello record. (If your trace contains multiple ClientHello records, expand the frame that contains the first one.) What is the value of the content type?

Ans- The content type is 22,for Handshake message ,with a handshake type of 01,Client Hello.



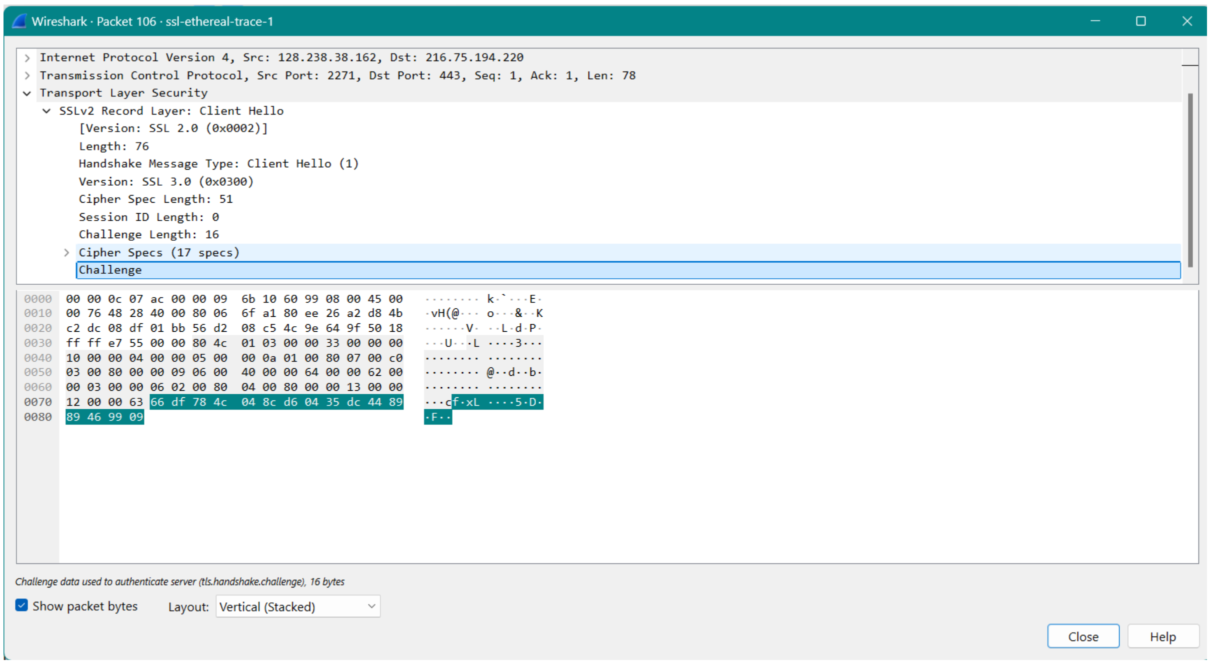
Wireshark · Packet 163 · ssl-ethereal-trace-1

```
> Frame 163: 156 bytes on wire (1248 bits), 156 bytes captured (1248 bits)
> Ethernet II, Src: IBM_10:60:99 (00:09:6b:10:60:99), Dst: All-HSRP-routers_00 (00:00:0c:07:ac:00)
> Internet Protocol Version 4, Src: 128.238.38.162, Dst: 216.75.194.220
> Transmission Control Protocol, Src Port: 2272, Dst Port: 443, Seq: 1, Ack: 1, Len: 102
v Transport Layer Security
  v SSLv3 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 97
    v Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 93
        Version: SSL 3.0 (0x0300)
```

```
0000  00 00 0c 07 ac 00 00 09  6b 10 60 99 08 00 45 00   ········ k·`···E·
0010  00 8e 48 45 40 00 80 06  6f 6c 80 ee 26 a2 d8 4b   ··HE@··· ol··&··K
0020  c2 dc 08 e0 01 bb fe b2  2b f1 74 7b f6 32 50 18   ········ +·t{·2P·
0030  ff ff a3 b1 00 00 16 03  00 00 61 01 00 00 5d 03   ········ ··a···]·
0040  00 42 db f0 c2 1b 78 1c  6c 64 4b 84 fe 4e fa 7b   ·B····x· ldK··N·{
0050  e6 ef 21 ef c9 8e 35 03  55 e9 06 95 00 1e 79 03   ··!···5· U·····y·
0060  1c 20 1b ad 05 fa ba 02  ea 92 c6 4c 54 be 45 47   · ······ ···LT·EG
0070  c3 2f 3e 3c a6 3d 3a 0c  86 dd ad 69 4b 45 68 2d   ·/><·=:· ···iKEh-
0080  a2 2f 00 16 00 04 00 05  00 0a 00 09 00 64 00 62   ·/······ ·····d·b
0090  00 03 00 06 00 13 00 12  00 63 01 00               ········ ·c··
```

Q4] Does the ClientHello record contain a nonce (also known as a "challenge")? If so, what is the value of the challenge in hexadecimal notation?

Ans-The client hello challenge is –



Wireshark · Packet 106 · ssl-ethereal-trace-1

```
> Internet Protocol Version 4, Src: 128.238.38.162, Dst: 216.75.194.220
> Transmission Control Protocol, Src Port: 2271, Dst Port: 443, Seq: 1, Ack: 1, Len: 78
v Transport Layer Security
  v SSLv2 Record Layer: Client Hello
      [Version: SSL 2.0 (0x0002)]
      Length: 76
      Handshake Message Type: Client Hello (1)
      Version: SSL 3.0 (0x0300)
      Cipher Spec Length: 51
      Session ID Length: 0
      Challenge Length: 16
    > Cipher Specs (17 specs)
      Challenge
```

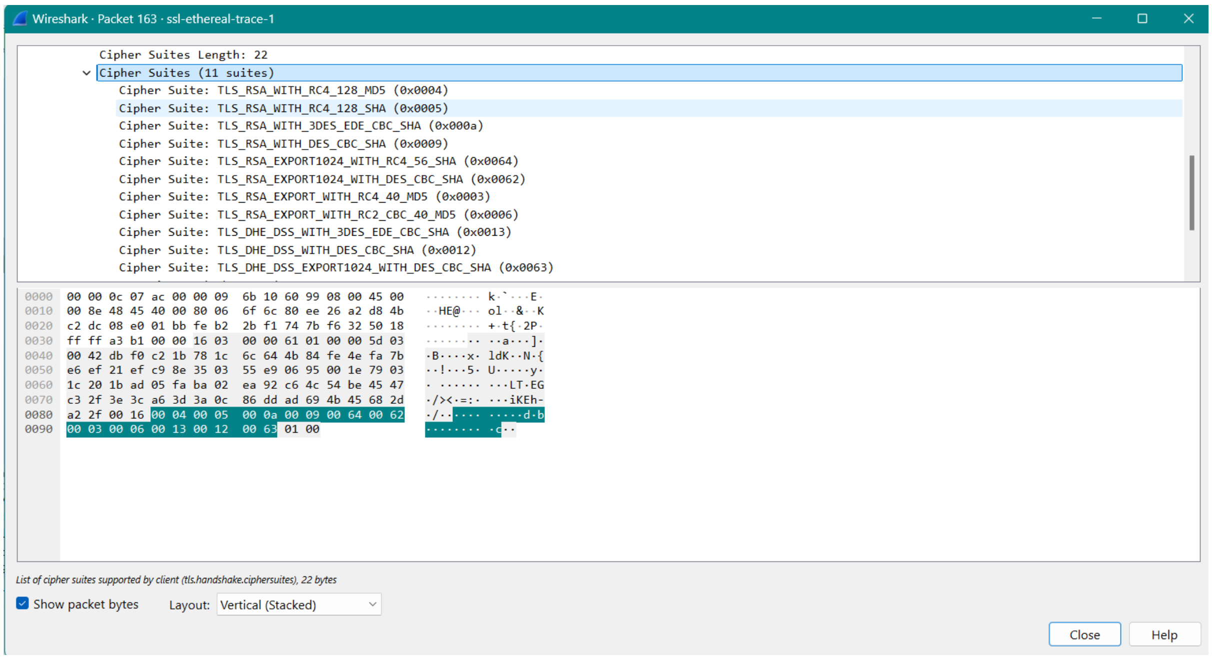
```
0000  00 00 0c 07 ac 00 00 09  6b 10 60 99 08 00 45 00   ········ k·`···E·
0010  00 76 48 28 40 00 80 06  6f a1 80 ee 26 a2 d8 4b   ·vH(@··· o···&··K
0020  c2 dc 08 df 01 bb 56 d2  08 c5 4c 9e 64 9f 50 18   ······V· ··L·d·P·
0030  ff ff e7 55 00 00 80 4c  01 03 00 00 33 00 00 00   ···U···L ····3···
0040  10 00 00 04 00 00 05 00  00 0a 01 00 80 07 00 c0   ········ ········
0050  03 00 80 00 00 09 06 00  40 00 00 64 00 00 62 00   ········ @··d··b·
0060  00 03 00 00 06 02 00 80  04 00 80 00 00 13 00 00   ········ ········
0070  12 00 00 63 66 df 78 4c  04 8c d6 04 35 dc 44 89   ···cf·xL ····5·D·
0080  89 46 99 09                                        ·F··
```

Challenge data used to authenticate server (tls.handshake.challenge), 16 bytes

☑ Show packet bytes   Layout: Vertical (Stacked)

Q5] Does the ClientHello record advertise the cyber suites it supports? If so, in the first listed suite, what are the public-key algorithm, the symmetric-key algorithm, and the hash algorithm?

Ans-The first suite uses RSA for public key cryptograpgy,RC4 for the symmetric-key cipher and uses MD5 hash algorithm.



Wireshark · Packet 163 · ssl-ethereal-trace-1

```
        Cipher Suites Length: 22
      v Cipher Suites (11 suites)
          Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
          Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
          Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
          Cipher Suite: TLS_RSA_WITH_DES_CBC_SHA (0x0009)
          Cipher Suite: TLS_RSA_EXPORT1024_WITH_RC4_56_SHA (0x0064)
          Cipher Suite: TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA (0x0062)
          Cipher Suite: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x0003)
          Cipher Suite: TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (0x0006)
          Cipher Suite: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x0013)
          Cipher Suite: TLS_DHE_DSS_WITH_DES_CBC_SHA (0x0012)
          Cipher Suite: TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA (0x0063)
```

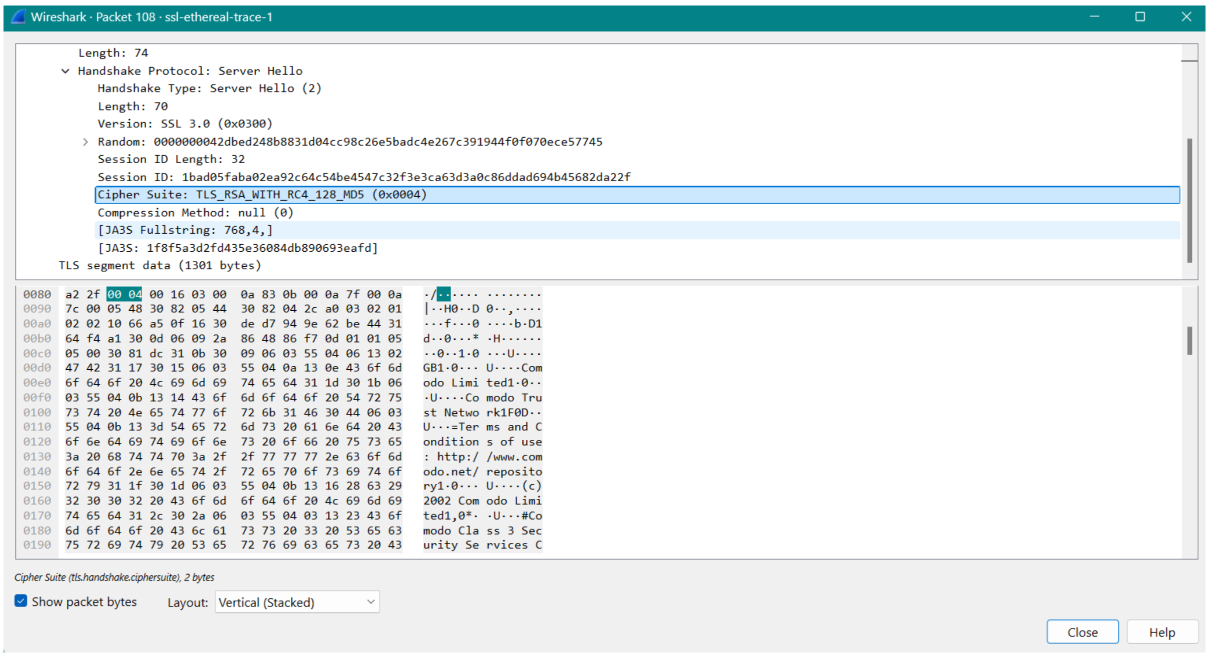
```
0000  00 00 0c 07 ac 00 00 09  6b 10 60 99 08 00 45 00   ········ k·`···E·
0010  00 8e 48 45 40 00 80 06  6f 6c 80 ee 26 a2 d8 4b   ··HE@··· ol··&··K
0020  c2 dc 08 e0 01 bb fe b2  2b f1 74 7b f6 32 50 18   ········ +·t{·2P·
0030  ff ff a3 b1 00 00 16 03  00 00 61 01 00 00 5d 03   ········ ··a···]·
0040  00 42 db f0 c2 1b 78 1c  6c 64 4b 84 fe 4e fa 7b   ·B····x· ldK··N·{
0050  e6 ef 21 ef c9 8e 35 03  55 e9 06 95 00 1e 79 03   ··!···5· U·····y·
0060  1c 20 1b ad 05 fa ba 02  ea 92 c6 4c 54 be 45 47   · ······ ···LT·EG
0070  c3 2f 3e 3c a6 3d 3a 0c  86 dd ad 69 4b 45 68 2d   ·/><·=:· ···iKEh-
0080  a2 2f 00 16 00 04 00 05  00 0a 00 09 00 64 00 62   ·/······ ·····d·b
0090  00 03 00 06 00 13 00 12  00 63 01 00               ········ ·c··
```

List of cipher suites supported by client (tls.handshake.ciphersuites), 22 bytes

☑ Show packet bytes   Layout: Vertical (Stacked)

Q6] Locate the ServerHello SSL record. Does this record specify a chosen cipher suite? What are the algorithms in the chosen cipher suite?

Ans- The first suite uses RSA for public key cryptograpgy,RC4 for the symmetric-key cipher and uses MD5 hash algorithm.


Wireshark · Packet 108 · ssl-ethereal-trace-1

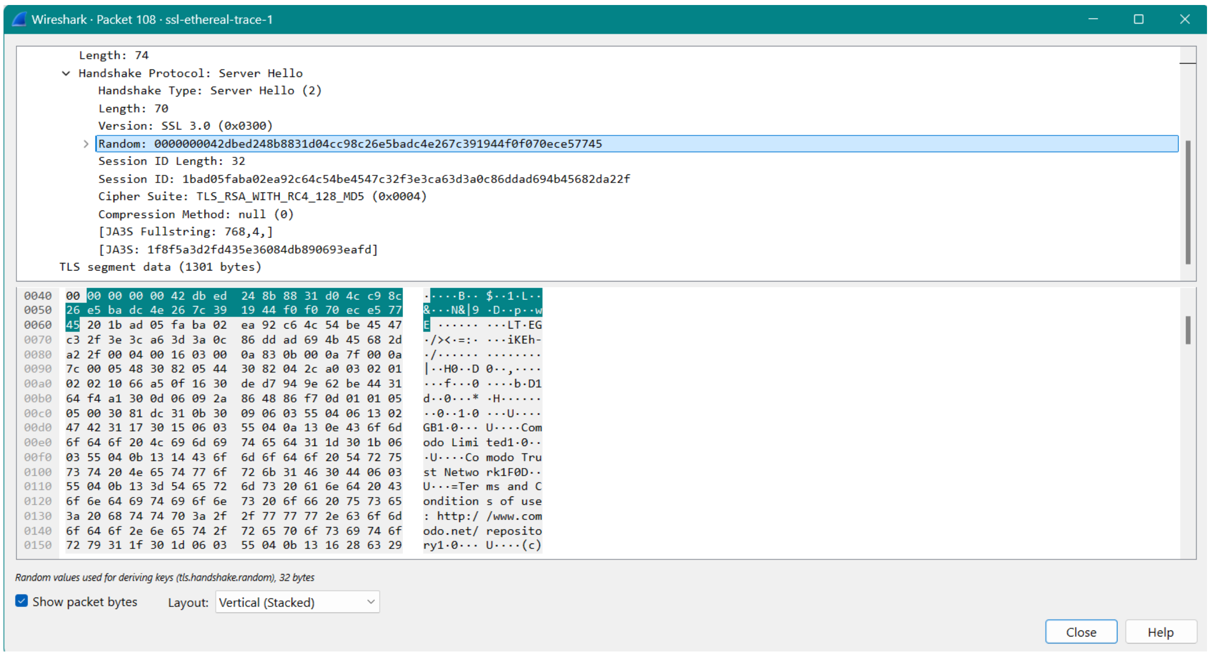
```
        Length: 74
      ˅ Handshake Protocol: Server Hello
            Handshake Type: Server Hello (2)
            Length: 70
            Version: SSL 3.0 (0x0300)
          > Random: 0000000042dbed248b8831d04cc98c26e5badc4e267c391944f0f070ece57745
            Session ID Length: 32
            Session ID: 1bad05faba02ea92c64c54be4547c32f3e3ca63d3a0c86ddad694b45682da22f
            Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
            Compression Method: null (0)
            [JA3S Fullstring: 768,4,]
            [JA3S: 1f8f5a3d2fd435e36084db890693eafd]
     TLS segment data (1301 bytes)

0080  a2 2f 00 04 00 16 03 00  0a 83 0b 00 0a 7f 00 0a   ·/······ ········
0090  7c 00 05 48 30 82 05 44  30 82 04 2c a0 03 02 01   |··H0··D 0··,····
00a0  02 02 10 66 a5 0f 16 30  de d7 94 9e 62 be 44 31   ···f···0 ····b·D1
00b0  64 f4 a1 30 0d 06 09 2a  86 48 86 f7 0d 01 01 05   d··0···* ·H······
00c0  05 00 30 81 dc 31 0b 30  09 06 03 55 04 06 13 02   ··0··1·0 ···U····
00d0  47 42 31 17 30 15 06 03  55 04 0a 13 0e 43 6f 6d   GB1·0··· U····Com
00e0  6f 64 6f 20 4c 69 6d 69  74 65 64 31 1d 30 1b 06   odo Limi ted1·0··
00f0  03 55 04 0b 13 14 43 6f  6d 6f 64 6f 20 54 72 75   ·U····Co modo Tru
0100  73 74 20 4e 65 74 77 6f  72 6b 31 46 30 44 06 03   st Netwo rk1F0D··
0110  55 04 0b 13 3d 54 65 72  6d 73 20 61 6e 64 20 43   U···=Ter ms and C
0120  6f 6e 64 69 74 69 6f 6e  73 20 6f 66 20 75 73 65   ondition s of use
0130  3a 20 68 74 74 70 3a 2f  2f 77 77 77 2e 63 6f 6d   : http:/ /www.com
0140  6f 64 6f 2e 6e 65 74 2f  72 65 70 6f 73 69 74 6f   odo.net/ reposito
0150  72 79 31 1f 30 1d 06 03  55 04 0b 13 16 28 63 29   ry1·0··· U····(c)
0160  32 30 30 32 20 43 6f 6d  6f 64 6f 20 4c 69 6d 69   2002 Com odo Limi
0170  74 65 64 31 2c 30 2a 06  03 55 04 03 13 23 43 6f   ted1,0*· ·U···#Co
0180  6d 6f 64 6f 20 43 6c 61  73 73 20 33 20 53 65 63   modo Cla ss 3 Sec
0190  75 72 69 74 79 20 53 65  72 76 69 63 65 73 20 43   urity Se rvices C
```

Cipher Suite (tls.handshake.ciphersuite), 2 bytes

Show packet bytes   Layout: Vertical (Stacked)

Q7] Does this record include a nonce? If so, how long is it? What is the purpose of the client and server nonces in SSL?

Ans-Yes,the record includes nonce listed under random.The nonce is 32 bits long,28 for the data and 4 for the time.The purpose is to prevent replay attack.


Wireshark · Packet 108 · ssl-ethereal-trace-1

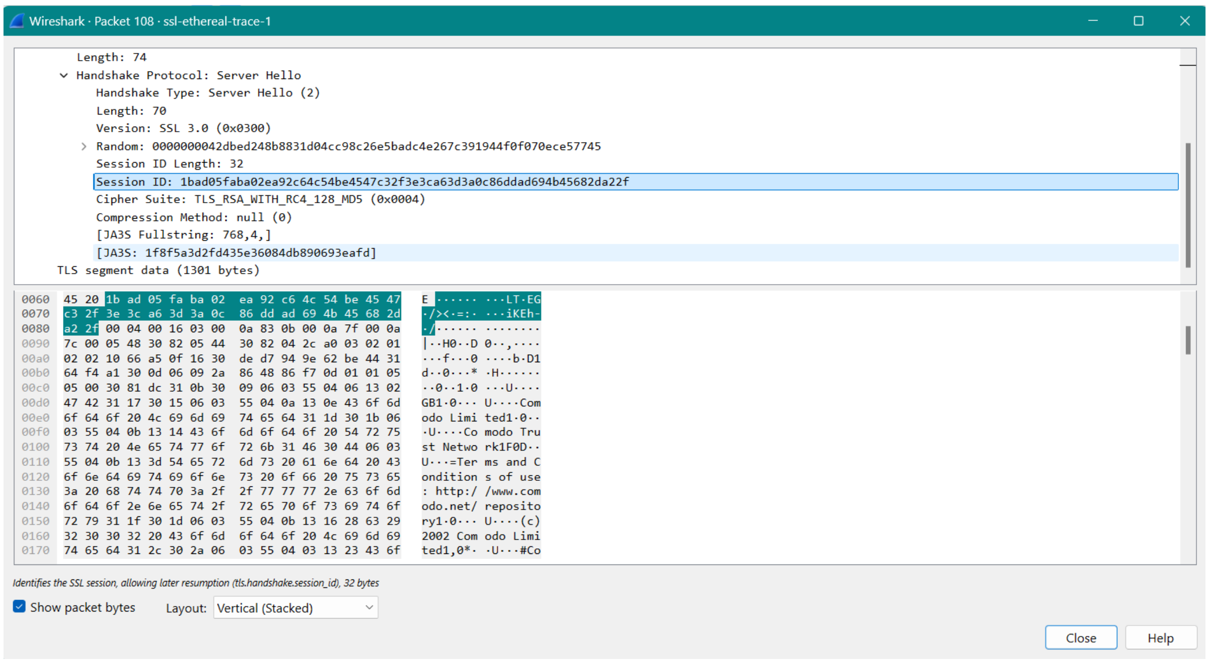
```
        Length: 74
      ˅ Handshake Protocol: Server Hello
            Handshake Type: Server Hello (2)
            Length: 70
            Version: SSL 3.0 (0x0300)
          > Random: 0000000042dbed248b8831d04cc98c26e5badc4e267c391944f0f070ece57745
            Session ID Length: 32
            Session ID: 1bad05faba02ea92c64c54be4547c32f3e3ca63d3a0c86ddad694b45682da22f
            Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
            Compression Method: null (0)
            [JA3S Fullstring: 768,4,]
            [JA3S: 1f8f5a3d2fd435e36084db890693eafd]
     TLS segment data (1301 bytes)

0040  00 00 00 00 00 42 db ed  24 8b 88 31 d0 4c c9 8c   ·····B·· $··1·L··
0050  26 e5 ba dc 4e 26 7c 39  19 44 f0 f0 70 ec e5 77   &···N&|9 ·D··p··w
0060  45 20 1b ad 05 fa ba 02  ea 92 c6 4c 54 be 45 47   E ······ ···LT·EG
0070  c3 2f 3e 3c a6 3d 3a 0c  86 dd ad 69 4b 45 68 2d   ·/><·=:· ···iKEh-
0080  a2 2f 00 04 00 16 03 00  0a 83 0b 00 0a 7f 00 0a   ·/······ ········
0090  7c 00 05 48 30 82 05 44  30 82 04 2c a0 03 02 01   |··H0··D 0··,····
00a0  02 02 10 66 a5 0f 16 30  de d7 94 9e 62 be 44 31   ···f···0 ····b·D1
00b0  64 f4 a1 30 0d 06 09 2a  86 48 86 f7 0d 01 01 05   d··0···* ·H······
00c0  05 00 30 81 dc 31 0b 30  09 06 03 55 04 06 13 02   ··0··1·0 ···U····
00d0  47 42 31 17 30 15 06 03  55 04 0a 13 0e 43 6f 6d   GB1·0··· U····Com
00e0  6f 64 6f 20 4c 69 6d 69  74 65 64 31 1d 30 1b 06   odo Limi ted1·0··
00f0  03 55 04 0b 13 14 43 6f  6d 6f 64 6f 20 54 72 75   ·U····Co modo Tru
0100  73 74 20 4e 65 74 77 6f  72 6b 31 46 30 44 06 03   st Netwo rk1F0D··
0110  55 04 0b 13 3d 54 65 72  6d 73 20 61 6e 64 20 43   U···=Ter ms and C
0120  6f 6e 64 69 74 69 6f 6e  73 20 6f 66 20 75 73 65   ondition s of use
0130  3a 20 68 74 74 70 3a 2f  2f 77 77 77 2e 63 6f 6d   : http:/ /www.com
0140  6f 64 6f 2e 6e 65 74 2f  72 65 70 6f 73 69 74 6f   odo.net/ reposito
0150  72 79 31 1f 30 1d 06 03  55 04 0b 13 16 28 63 29   ry1·0··· U····(c)
```

Random values used for deriving keys (tls.handshake.random), 32 bytes

Show packet bytes   Layout: Vertical (Stacked)

Q8] Does this record include a session ID? What is the purpose of the session ID?

Ans- Yes,it provides a unique persistent identifier for the SSL session which is sent.The client may resume the same session later by using the server provided session id when it sends the ClientHello.



Wireshark · Packet 108 · ssl-ethereal-trace-1

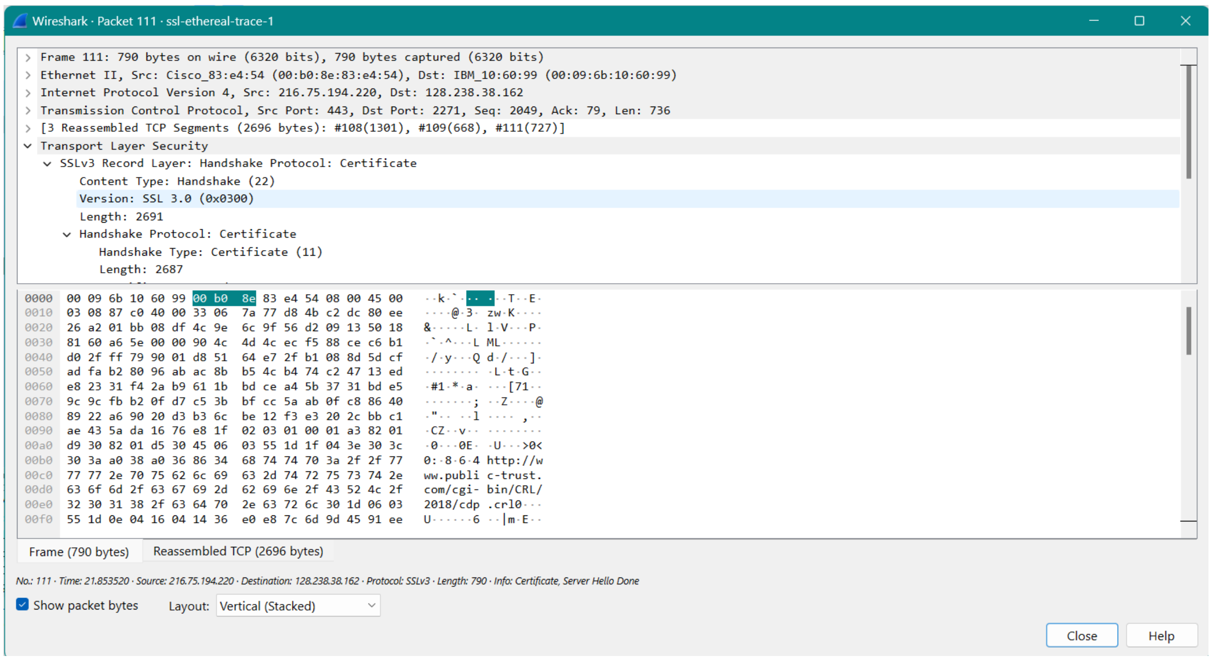
```
        Length: 74
    ⌄ Handshake Protocol: Server Hello
          Handshake Type: Server Hello (2)
          Length: 70
          Version: SSL 3.0 (0x0300)
        > Random: 0000000042dbed248b8831d04cc98c26e5badc4e267c391944f0f070ece57745
          Session ID Length: 32
          Session ID: 1bad05faba02ea92c64c54be4547c32f3e3ca63d3a0c86ddad694b45682da22f
          Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
          Compression Method: null (0)
          [JA3S Fullstring: 768,4,]
          [JA3S: 1f8f5a3d2fd435e36084db890693eafd]
  TLS segment data (1301 bytes)
```

```
0060  45 20 1b ad 05 fa ba 02  ea 92 c6 4c 54 be 45 47   E ······ ···LT·EG
0070  c3 2f 3e 3c a6 3d 3a 0c  86 dd ad 69 4b 45 68 2d   ·/><·=:· ···iKEh-
0080  a2 2f 00 04 00 16 03 00  0a 83 0b 00 0a 7f 00 0a   ·/······ ········
0090  7c 00 05 48 30 82 05 44  30 82 04 2c a0 03 02 01   |··H0··D 0··,····
00a0  02 02 10 66 a5 0f 16 30  de d7 94 9e 62 be 44 31   ···f···0 ····b·D1
00b0  64 f4 a1 30 0d 06 09 2a  86 48 86 f7 0d 01 01 05   d··0···* ·H······
00c0  05 00 30 81 dc 31 0b 30  09 06 03 55 04 06 13 02   ··0··1·0 ···U····
00d0  47 42 31 17 30 15 06 03  55 04 0a 13 0e 43 6f 6d   GB1·0··· U····Com
00e0  6f 64 6f 20 4c 69 6d 69  74 65 64 31 1d 30 1b 06   odo Limi ted1·0··
00f0  03 55 04 0b 13 14 43 6f  6d 6f 64 6f 20 54 72 75   ·U····Co modo Tru
0100  73 74 20 4e 65 74 77 6f  72 6b 31 46 30 44 06 03   st Netwo rk1F0D··
0110  55 04 0b 13 3d 54 65 72  6d 73 20 61 6e 64 20 43   U···=Ter ms and C
0120  6f 6e 64 69 74 69 6f 6e  73 20 6f 66 20 75 73 65   ondition s of use
0130  3a 20 68 74 74 70 3a 2f  2f 77 77 77 2e 63 6f 6d   : http:/ /www.com
0140  6f 64 6f 2e 6e 65 74 2f  72 65 70 6f 73 69 74 6f   odo.net/ reposito
0150  72 79 31 1f 30 1d 06 03  55 04 0b 13 16 28 63 29   ry1·0··· U····(c)
0160  32 30 30 32 20 43 6f 6d  6f 64 6f 20 4c 69 6d 69   2002 Com odo Limi
0170  74 65 64 31 2c 30 2a 06  03 55 04 03 13 23 43 6f   ted1,0*· ·U···#Co
```

Identifies the SSL session, allowing later resumption (tls.handshake.session_id), 32 bytes

Q9] Does this record contain a certificate, or is the certificate included in a separate record. Does the certificate fit into a single Ethernet frame?

Ans-There is no certificate it is another record.It fits into a single ethernet frame.



Wireshark · Packet 111 · ssl-ethereal-trace-1

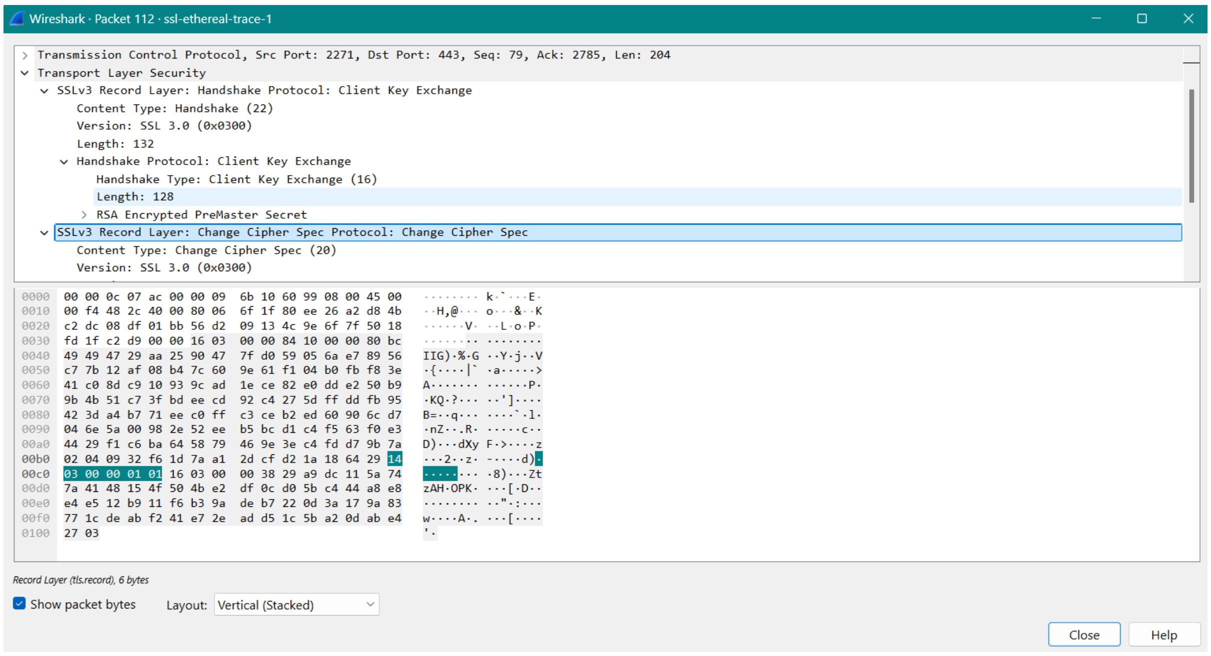
```
> Frame 111: 790 bytes on wire (6320 bits), 790 bytes captured (6320 bits)
> Ethernet II, Src: Cisco_83:e4:54 (00:b0:8e:83:e4:54), Dst: IBM_10:60:99 (00:09:6b:10:60:99)
> Internet Protocol Version 4, Src: 216.75.194.220, Dst: 128.238.38.162
> Transmission Control Protocol, Src Port: 443, Dst Port: 2271, Seq: 2049, Ack: 79, Len: 736
> [3 Reassembled TCP Segments (2696 bytes): #108(1301), #109(668), #111(727)]
⌄ Transport Layer Security
  ⌄ SSLv3 Record Layer: Handshake Protocol: Certificate
        Content Type: Handshake (22)
        Version: SSL 3.0 (0x0300)
        Length: 2691
      ⌄ Handshake Protocol: Certificate
          Handshake Type: Certificate (11)
          Length: 2687
```

```
0000  00 09 6b 10 60 99 00 b0  8e 83 e4 54 08 00 45 00   ··k·`··· ···T··E·
0010  03 08 87 c0 40 00 33 06  7a 77 d8 4b c2 dc 80 ee   ····@·3· zw·K····
0020  26 a2 01 bb 08 df 4c 9e  6c 9f 56 d2 09 13 50 18   &·····L· l·V···P·
0030  81 60 a6 5e 00 00 90 4c  4d 4c ec f5 88 ce c6 b1   ·`·^···L ML······
0040  d0 2f ff 79 90 01 d8 51  64 e7 2f b1 08 8d 5d cf   ·/·y···Q d·/···]·
0050  ad fa b2 80 96 ab ac 8b  b5 4c b4 74 c2 47 13 ed   ········ ·L·t·G··
0060  e8 23 31 f4 2a b9 61 1b  bd ce a4 5b 37 31 bd e5   ·#1·*·a· ···[71··
0070  9c 9c fb b2 0f d7 c5 3b  bf cc 5a ab 0f c8 86 40   ·······; ··Z····@
0080  89 22 a6 90 20 d3 b3 6c  be 12 f3 e3 20 2c bb c1   ·"·· ··l ···· ,··
0090  ae 43 5a da 16 76 e8 1f  02 03 01 00 01 a3 82 01   ·CZ··v·· ········
00a0  d9 30 82 01 d5 30 45 06  03 55 1d 1f 04 3e 30 3c   ·0···0E· ·U···>0<
00b0  30 3a a0 38 a0 36 86 34  68 74 74 70 3a 2f 2f 77   0:·8·6·4 http://w
00c0  77 77 2e 70 75 62 6c 69  63 2d 74 72 75 73 74 2e   ww.publi c-trust.
00d0  63 6f 6d 2f 63 67 69 2d  62 69 6e 2f 43 52 4c 2f   com/cgi- bin/CRL/
00e0  32 30 31 38 2f 63 64 70  2e 63 72 6c 30 1d 06 03   2018/cdp .crl0···
00f0  55 1d 0e 04 16 04 14 36  e0 e8 7c 6d 9d 45 91 ee   U······6 ··|m·E··
```

No.: 111 · Time: 21.853520 · Source: 216.75.194.220 · Destination: 128.238.38.162 · Protocol: SSLv3 · Length: 790 · Info: Certificate, Server Hello Done

Q10] Locate the client key exchange record. Does this record contain a pre-master secret? What is this secret used for? Is the secret encrypted? If so, how? How long is the encrypted secret?

Ans-Yes it contains a premaster secret.Used on both sides client and server to make a secret which is used to generate keys for mac and encryption.The server is encrypted with the public key of server,which cleient extracts from the certificate sent.



Wireshark · Packet 112 · ssl-ethereal-trace-1

```
> Transmission Control Protocol, Src Port: 2271, Dst Port: 443, Seq: 79, Ack: 2785, Len: 204
v Transport Layer Security
  v SSLv3 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 132
    v Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 128
      > RSA Encrypted PreMaster Secret
  v SSLv3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: SSL 3.0 (0x0300)
```

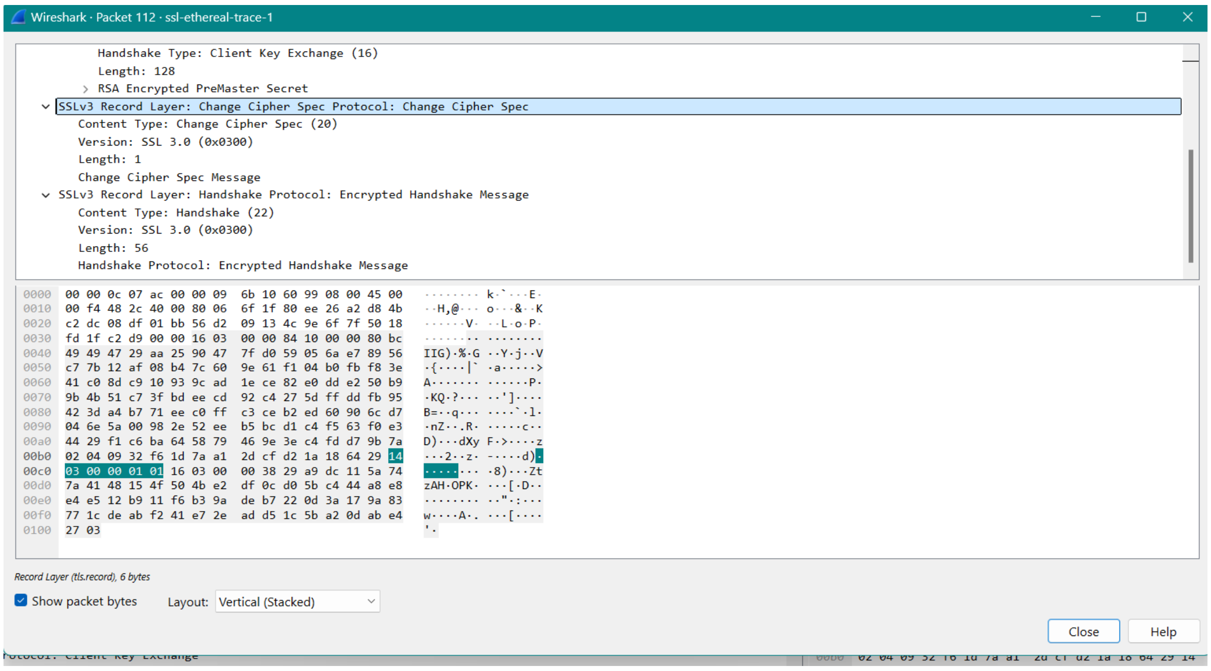
```
0000  00 00 0c 07 ac 00 00 09  6b 10 60 99 08 00 45 00   ········ k·`···E·
0010  00 f4 48 2c 40 00 80 06  6f 1f 80 ee 26 a2 d8 4b   ··H,@··· o···&··K
0020  c2 dc 08 df 01 bb 56 d2  09 13 4c 9e 6f 7f 50 18   ······V· ··L·o·P·
0030  fd 1f c2 d9 00 00 16 03  00 00 84 10 00 00 80 bc   ········ ········
0040  49 49 47 29 aa 25 90 47  7f d0 59 05 6a e7 89 56   IIG)·%·G ··Y·j··V
0050  c7 7b 12 af 08 b4 7c 60  9e 61 f1 04 b0 fb f8 3e   ·{····|` ·a·····>
0060  41 c0 8d c9 10 93 9c ad  1e ce 82 e0 dd e2 50 b9   A······· ······P·
0070  9b 4b 51 c7 3f bd ee cd  92 c4 27 5d ff dd fb 95   ·KQ·?··· ··']····
0080  42 3d a4 b7 71 ee c0 ff  c3 ce b2 ed 60 90 6c d7   B=··q··· ····`·l·
0090  04 6e 5a 00 98 2e 52 ee  b5 bc d1 c4 f5 63 f0 e3   ·nZ··.R· ·····c··
00a0  44 29 f1 c6 ba 64 58 79  46 9e 3e c4 fd d7 9b 7a   D)···dXy F·>····z
00b0  02 04 09 32 f6 1d 7a a1  2d cf d2 1a 18 64 29 14   ···2··z· -····d)·
00c0  03 00 00 01 01 16 03 00  00 38 29 a9 dc 11 5a 74   ········ ·8)···Zt
00d0  7a 41 48 15 4f 50 4b e2  df 0c d0 5b c4 44 a8 e8   zAH·OPK· ···[·D··
00e0  e4 e5 12 b9 11 f6 b3 9a  de b7 22 0d 3a 17 9a 83   ········ ··"·:···
00f0  77 1c de ab f2 41 e7 2e  ad d5 1c 5b a2 0d ab e4   w····A·. ···[····
0100  27 03                                              '.
```

Record Layer (tls.record), 6 bytes

☑ Show packet bytes   Layout: Vertical (Stacked)

Q11] What is the purpose of the Change Cipher Spec record? How many bytes is the record in your trace?

Ans-The purpose of the Change Cipher Spec record is to indicate that the contents of the following SSL records sent by the client (data, not header) will be encrypted. This record is 6 bytes long: 5 for the header and 1 for the message segment.


Wireshark · Packet 112 · ssl-ethereal-trace-1

```
        Handshake Type: Client Key Exchange (16)
        Length: 128
      > RSA Encrypted PreMaster Secret
  v SSLv3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: SSL 3.0 (0x0300)
      Length: 1
      Change Cipher Spec Message
  v SSLv3 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 56
      Handshake Protocol: Encrypted Handshake Message
```

```
0000  00 00 0c 07 ac 00 00 09  6b 10 60 99 08 00 45 00   ········ k·`···E·
0010  00 f4 48 2c 40 00 80 06  6f 1f 80 ee 26 a2 d8 4b   ··H,@··· o···&··K
0020  c2 dc 08 df 01 bb 56 d2  09 13 4c 9e 6f 7f 50 18   ······V· ··L·o·P·
0030  fd 1f c2 d9 00 00 16 03  00 00 84 10 00 00 80 bc   ········ ········
0040  49 49 47 29 aa 25 90 47  7f d0 59 05 6a e7 89 56   IIG)·%·G ··Y·j··V
0050  c7 7b 12 af 08 b4 7c 60  9e 61 f1 04 b0 fb f8 3e   ·{····|` ·a·····>
0060  41 c0 8d c9 10 93 9c ad  1e ce 82 e0 dd e2 50 b9   A······· ······P·
0070  9b 4b 51 c7 3f bd ee cd  92 c4 27 5d ff dd fb 95   ·KQ·?··· ··']····
0080  42 3d a4 b7 71 ee c0 ff  c3 ce b2 ed 60 90 6c d7   B=··q··· ····`·l·
0090  04 6e 5a 00 98 2e 52 ee  b5 bc d1 c4 f5 63 f0 e3   ·nZ··.R· ·····c··
00a0  44 29 f1 c6 ba 64 58 79  46 9e 3e c4 fd d7 9b 7a   D)···dXy F·>····z
00b0  02 04 09 32 f6 1d 7a a1  2d cf d2 1a 18 64 29 14   ···2··z· -····d)·
00c0  03 00 00 01 01 16 03 00  00 38 29 a9 dc 11 5a 74   ········ ·8)···Zt
00d0  7a 41 48 15 4f 50 4b e2  df 0c d0 5b c4 44 a8 e8   zAH·OPK· ···[·D··
00e0  e4 e5 12 b9 11 f6 b3 9a  de b7 22 0d 3a 17 9a 83   ········ ··"·:···
00f0  77 1c de ab f2 41 e7 2e  ad d5 1c 5b a2 0d ab e4   w····A·. ···[····
0100  27 03                                              '·
```

Record Layer (tls.record), 6 bytes

☑ Show packet bytes   Layout: Vertical (Stacked)

Q12] In the encrypted handshake record, what is being encrypted? How?

Ans-In the encrypted handshake record, a MAC of the concatenation of all the previous handshake messages sent from this client is generated and sent to the server.

Q13] Does the server also send a change cipher record and an encrypted handshake record to the client? How are those records different from those sent by the client?

Ans-Yes, the server will also send a Change Cipher Spec record and encrypted handshake to the client. The server's encrypted handshake record is different from that sent by the client because it contains the concatenation of all the handshake messages sent from the server rather than from the client. Otherwise, the records would end up being the same.

Q14] How is the application data being encrypted? Do the records containing application data include a MAC? Does Wireshark distinguish between the encrypted application data and the MAC?

Ans-Application data is encrypted using symmetric key encryption algorithm chosen in the handshake phase (RC4) using the keys generated using the pre-master key and nonces from both client and server. The client encryption key is used to encrypt the data being sent from client to server and the server encryption key is used to encrypt the data being sent from the server to the client.

Q15] Comment on and explain anything else that you found interesting in the trace.

Ans-NIL.