2. Implementing symmetric key algorithms -AES
And
9. Implement an application that will be using AES algorithm with key size 192 bits.

```
package lab;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.security.Key;
import java.util.Base64;
class AESExample {

   public static void main(String[] args) throws Exception {
        // Generate a symmetric key
        SecretKey secretKey = generateAESKey();

        // Text to be encrypted
        String plainText = "Hello, AES Encryption!";

        // Encrypt the text
        byte[] encryptedText = encrypt(plainText, secretKey);

        System.out.println("Encrypted Text: " +
Base64.getEncoder().encodeToString(encryptedText));

        // Decrypt the text
        String decryptedText = decrypt(encryptedText, secretKey);

        System.out.println("Decrypted Text: " + decryptedText);
   }

   public static SecretKey generateAESKey() throws Exception {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(256); // Key size 256 bits
        return keyGenerator.generateKey();
   }

   public static byte[] encrypt(String plainText, Key key) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, key);
        return cipher.doFinal(plainText.getBytes());
   }

   public static String decrypt(byte[] cipherText, Key key) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, key);
        byte[] decryptedBytes = cipher.doFinal(cipherText);
        return new String(decryptedBytes);
   }
```

```
}
```

Output:

```
Encrypted Text: T7fV1pdM4KsBfXjPh3Ky/9PCnWqxIV7daxAWCUrR3CM=
Decrypted Text: Hello, AES Encryption!
```

3. Write a Java/C/C++ program to implement the DES algorithm logic.
And
15. Implement an application that will be using DES algorithm for encryption/decryption.

```
package lab;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;
import java.util.Base64;

class DESExample {

   public static void main(String[] args) throws Exception {
        String plainText = "Hello, DES Encryption!";
        String keyString = "12345678"; // 8-byte key

        // Encrypt the text
        byte[] encryptedText = encrypt(plainText, keyString);

        System.out.println("Encrypted Text: " +
Base64.getEncoder().encodeToString(encryptedText));

        // Decrypt the text
        String decryptedText = decrypt(encryptedText, keyString);

        System.out.println("Decrypted Text: " + decryptedText);
   }

   public static byte[] encrypt(String plainText, String keyString) throws
Exception {
        byte[] keyBytes = keyString.getBytes();
        DESKeySpec desKeySpec = new DESKeySpec(keyBytes);
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
        SecretKey secretKey = keyFactory.generateSecret(desKeySpec);

        Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
```

```
        return cipher.doFinal(plainText.getBytes());
    }

    public static String decrypt(byte[] cipherText, String keyString) throws
Exception {
        byte[] keyBytes = keyString.getBytes();
        DESKeySpec desKeySpec = new DESKeySpec(keyBytes);
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
        SecretKey secretKey = keyFactory.generateSecret(desKeySpec);

        Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);

        byte[] decryptedBytes = cipher.doFinal(cipherText);
        return new String(decryptedBytes);
    }
}
```

Output:

```
Encrypted Text: fy7lEyzVC0ilHAnkx/3yNh0erOizrD3L
Decrypted Text: Hello, DES Encryption!
```

4. Implement the SIGNATURE SCHEME - Digital Signature Standard.
And
11. Implement the SIGNATURE SCHEME - Digital Signature Standard.

```
import java.security.*;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;

class DigitalSignatureExample {

    public static void main(String[] args) throws Exception {
        // Generate key pair
        KeyPair keyPair = generateKeyPair();

        // Sign the data
        String message = "This is the message to be signed.";
        byte[] signature = sign(message, keyPair.getPrivate());

        System.out.println("Signature: " +
Base64.getEncoder().encodeToString(signature));

        // Verify the signature
        boolean isVerified = verify(message, signature, keyPair.getPublic());
```

```
        System.out.println("Signature verified: " + isVerified);
    }

    public static KeyPair generateKeyPair() throws Exception {
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
        keyPairGenerator.initialize(1024); // key size
        return keyPairGenerator.generateKeyPair();
    }

    public static byte[] sign(String message, PrivateKey privateKey) throws
Exception {
        Signature signature = Signature.getInstance("SHA256withDSA");
        signature.initSign(privateKey);
        signature.update(message.getBytes());
        return signature.sign();
    }

    public static boolean verify(String message, byte[] signature, PublicKey
publicKey) throws Exception {
        Signature verifier = Signature.getInstance("SHA256withDSA");
        verifier.initVerify(publicKey);
        verifier.update(message.getBytes());
        return verifier.verify(signature);
    }
}
```

Output:

```
Signature: MCwCFB8FdleIZVVAgAGLh8n5D6dCuEO2AhQTj4s2otrwaRSAwOEFSA9R2/wCuw==
Signature verified: true
```

6. Experiment Eavesdropping, Dictionary attacks, MITM attacks.

7.Check message integrity and confidentiality using SSL.
And
16. Check message integrity and confidentiality using SSL

```
package lab;

import java.io.*;
import java.net.*;
import javax.net.ssl.*;

class SSLClient {

    public static void main(String[] args) throws Exception {
        String host = "www.google.com";
        int port = 443;
```

```
        // Create SSL context
        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(null, null, new java.security.SecureRandom());

        // Create SSL socket factory
        SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();

        // Connect to the server
        SSLSocket sslSocket = (SSLSocket) sslSocketFactory.createSocket(host,
port);

        // Perform SSL handshake
        sslSocket.startHandshake();

        // Send and receive data over the SSL connection
        PrintWriter out = new PrintWriter(sslSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new
InputStreamReader(sslSocket.getInputStream()));

        // Example: Send a message
        String messageToSend = "Hello, Server!";
        out.println(messageToSend);

        // Example: Receive a response
        String receivedMessage = in.readLine();
        System.out.println("Received from server: " + receivedMessage);

        // Close the connection
        out.close();
        in.close();
        sslSocket.close();
    }
}
```

Output:

```
Received from server: HTTP/1.0 400 Bad Request
```

Or

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\Jet
Received from server: HTTP/1.0 501 Not Implemented

Process finished with exit code 0
```

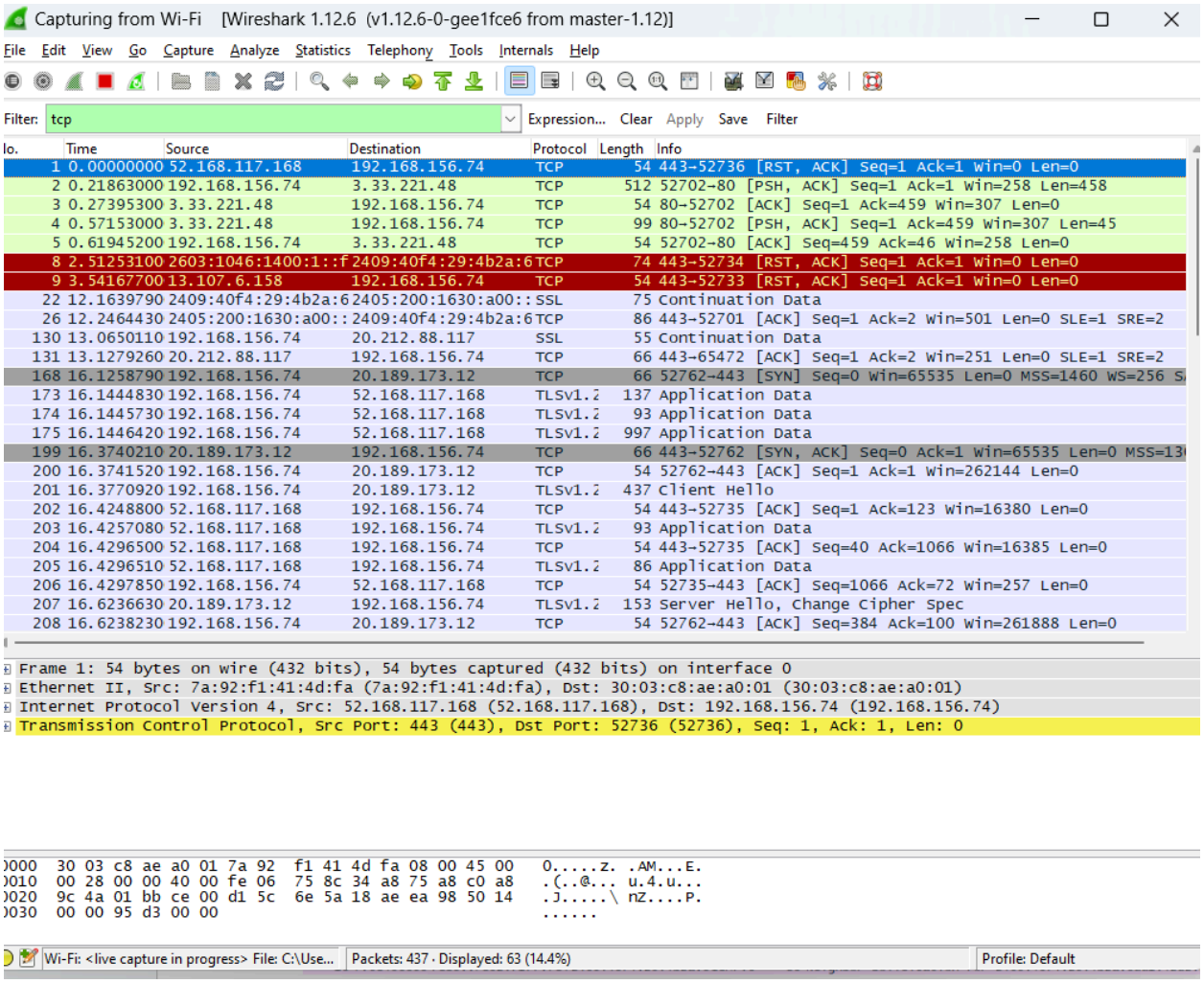
10. Installation of Wire shark, tcpdump and observe data transferred in client-server communication using UDP/TCP and identify the UDP/TCP datagram.

For Windows:

You can download the Wireshark installer from the official website: Wireshark Download

Using Wireshark:

1. Start Wireshark.
2. Select the network interface you want to capture packets from (e.g., Ethernet, Wi-Fi).
3. Click on the Start button to begin capturing packets.
4. Perform the client-server communication using UDP or TCP.
5. Stop the packet capture in Wireshark.
6. Use filters like `udp` or `tcp` to filter packets based on the protocol.



12. Calculate the message digest of a text using the SHA-1 algorithm.

```
package lab;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

class SHA1Example {

   public static void main(String[] args) {
        String text = "This is the text for which we want to calculate the SHA-1
hash.";

        try {
            byte[] digest = calculateSHA1(text.getBytes());
            String hexDigest = bytesToHex(digest);
            System.out.println("SHA-1 digest: " + hexDigest);
        } catch (NoSuchAlgorithmException e) {
```

```
            e.printStackTrace();
        }
    }

    public static byte[] calculateSHA1(byte[] input) throws
NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        return md.digest(input);
    }

    public static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}
```

Output:

```
SHA-1 digest: af3c4e1c25ab268f278378df87ab1c2a3af4db47

Process finished with exit code 0
```

13. Study to configure Firewall, VPN

Configuring a Firewall:

1. Choose a Firewall Solution: Decide whether to use a hardware firewall (physical appliance) or a software firewall (running on a server or workstation). Popular firewall solutions include pfSense, Cisco ASA, iptables (Linux), and Windows Firewall (built into Windows OS).
2. Plan Your Firewall Rules: Determine what traffic you want to allow or block. Consider factors such as the type of services running on your network, security policies, and compliance requirements.
3. Configure Firewall Rules:
    - Allow necessary incoming traffic: For example, HTTP (port 80) and HTTPS (port 443) for web servers, SSH (port 22) for remote administration, etc.

    - Block unnecessary or potentially harmful traffic: For example, ICMP (ping) requests, unused ports, known malicious IP addresses, etc.
    - Implement NAT (Network Address Translation) if needed to map internal IP addresses to external ones.
4. Enable Logging and Monitoring: Set up logging to monitor firewall activity. Monitor logs regularly to identify potential security threats or policy violations.
5. Test and Fine-Tune: Test your firewall rules to ensure they function as expected. Make adjustments as necessary based on testing results and ongoing monitoring.

Configuring a VPN:

1. Choose a VPN Solution: Decide whether to use a hardware VPN appliance, software VPN server, or VPN service provider. Common VPN solutions include OpenVPN, IPsec, WireGuard, and commercial VPN services.
2. Install and Configure VPN Software: Install the VPN server software on a dedicated server or device within your network. Configure the VPN server settings, including authentication methods, encryption algorithms, and IP address assignment (e.g., dynamic or static).
3. Set Up VPN Client Access: Configure VPN client access settings, such as user authentication credentials, VPN client software installation, and VPN connection profiles.
4. Test VPN Connectivity: Test VPN connectivity from remote client devices to ensure they can establish a secure connection to the VPN server. Troubleshoot and resolve any connectivity issues as needed.
5. Implement VPN Security Measures: Implement additional security measures to enhance VPN security, such as multi-factor authentication, client-side security policies, and regular security audits.
6. Monitor VPN Traffic: Monitor VPN traffic and log VPN connection activity to detect and respond to potential security threats or policy violations.
7. Regular Maintenance: Perform regular maintenance tasks, such as software updates, security patches, and configuration backups, to ensure the continued security and reliability of the VPN infrastructure.

17. Write a Java/C/C++ program to implement SHA-1 Hash technique.

```java
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

class SHA1Example {

    public static void main(String[] args) {
        String input = "Hello, SHA-1!";
        String sha1Hash = sha1(input);
        System.out.println("SHA-1 hash: " + sha1Hash);
    }

    public static String sha1(String input) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-1");
            byte[] hashBytes = digest.digest(input.getBytes());
            StringBuilder hexString = new StringBuilder();

            for (byte hashByte : hashBytes) {
                String hex = Integer.toHexString(0xff & hashByte);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }

            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Output:

```
SHA-1 hash: f322e078fef4f49da1618d3793d3272a91f0488c
```