```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class AESExample {
    public static void main(String[] args) throws Exception {
        String plainText = "Hello, World!";

        // Generate a new AES key
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128); // for example, 128-bit key
        SecretKey secretKey = keyGen.generateKey();

        // Encrypt the plaintext
        String encryptedText = encrypt(plainText, secretKey);
        System.out.println("Encrypted Text: " + encryptedText);

        // Decrypt the ciphertext
        String decryptedText = decrypt(encryptedText, secretKey);
        System.out.println("Decrypted Text: " + decryptedText);
    }

    public static String encrypt(String plainText, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }
```

```
    public static String decrypt(String encryptedText, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
        return new String(decryptedBytes);
    }
}

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class DESExample {
    public static void main(String[] args) throws Exception {
        String plainText = "Hello, World!";

        // Generate a new DES key
        KeyGenerator keyGen = KeyGenerator.getInstance("DES");
        keyGen.init(56); // DES uses a 56-bit key
        SecretKey secretKey = keyGen.generateKey();

        // Encrypt the plaintext
        String encryptedText = encrypt(plainText, secretKey);
        System.out.println("Encrypted Text: " + encryptedText);

        // Decrypt the ciphertext
        String decryptedText = decrypt(encryptedText, secretKey);
        System.out.println("Decrypted Text: " + decryptedText);
    }
```

```
    public static String encrypt(String plainText, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("DES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    public static String decrypt(String encryptedText, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("DES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
        return new String(decryptedBytes);
    }
}
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.util.Base64;

public class DigitalSignatureExample {

    public static void main(String[] args) throws Exception {
        String data = "Hello, this is a secure message.";

        // Generate RSA key pair
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(2048);
        KeyPair keyPair = keyGen.generateKeyPair();
```

```
        PrivateKey privateKey = keyPair.getPrivate();
        PublicKey publicKey = keyPair.getPublic();

        // Create digital signature
        String signature = sign(data, privateKey);
        System.out.println("Digital Signature: " + signature);

        // Verify digital signature
        boolean isVerified = verify(data, signature, publicKey);
        System.out.println("Signature Verification: " + isVerified);
    }

    public static String sign(String data, PrivateKey privateKey) throws Exception {
        Signature rsa = Signature.getInstance("SHA256withRSA");
        rsa.initSign(privateKey);
        rsa.update(data.getBytes());
        byte[] signatureBytes = rsa.sign();
        return Base64.getEncoder().encodeToString(signatureBytes);
    }

    public static boolean verify(String data, String signature, PublicKey publicKey) throws Exception {
        Signature rsa = Signature.getInstance("SHA256withRSA");
        rsa.initVerify(publicKey);
        rsa.update(data.getBytes());
        byte[] signatureBytes = Base64.getDecoder().decode(signature);
        return rsa.verify(signatureBytes);
    }
}
```

```
keytool -genkeypair -keyalg RSA -keysize 2048 -validity 365 -alias myserverkey -keystore
samlKeystore.jks -storepass password -keypass password -dname
"CN=localhost,OU=Unknown,O=Unknown,L=Unknown,ST=Unknown,C=Unknown"
```

```
import javax.crypto.Cipher;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.Base64;

public class RSAExample {
    public static void main(String[] args) throws Exception {
        String plainText = "Hello, World!";

        // Generate RSA key pair
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(2048);
        KeyPair keyPair = keyGen.generateKeyPair();
        PublicKey publicKey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();

        // Encrypt the plaintext
        String encryptedText = encrypt(plainText, publicKey);
        System.out.println("Encrypted Text: " + encryptedText);

        // Decrypt the ciphertext
        String decryptedText = decrypt(encryptedText, privateKey);
        System.out.println("Decrypted Text: " + decryptedText);
    }

    public static String encrypt(String plainText, PublicKey publicKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
```

```
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    public static String decrypt(String encryptedText, PrivateKey privateKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
        return new String(decryptedBytes);
    }
}
```

Client.java:

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

public class Client {
    public static void main(String[] args) throws Exception {
        // Load the truststore
        char[] truststorePassword = "password".toCharArray();
        KeyStore trustStore = KeyStore.getInstance("JKS");
        FileInputStream fis = new FileInputStream("samlKeystore.jks");
        trustStore.load(fis, truststorePassword);

        // Set up the trust manager factory
        TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
        tmf.init(trustStore);

        // Set up the SSL context
        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(null, tmf.getTrustManagers(), null);
```

```
        // Create the socket factory
        SSLSocketFactory sf = sslContext.getSocketFactory();
        SSLSocket socket = (SSLSocket) sf.createSocket("localhost", 9999);

        // Set up input and output streams
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

        // Send message to server
        out.println("Hello from client");

        // Read response from server
        String response = in.readLine();
        System.out.println("Response from server: " + response);

        // Close streams and socket
        out.close();
        in.close();
        socket.close();
    }
}
```

Server.java:

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

public class Server {
    public static void main(String[] args) {
        try {
            // Load the keystore
```

```
        char[] keystorePassword = "password".toCharArray();
        char[] keyPassword = "password".toCharArray();
        KeyStore keyStore = KeyStore.getInstance("JKS");
        try (FileInputStream fis = new FileInputStream("samlKeystore.jks")) {
           keyStore.load(fis, keystorePassword);
        }

        // Set up the key manager factory
        KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
        kmf.init(keyStore, keyPassword);

        // Set up the SSL context
        SSLContext sslContext = SSLContext.getInstance("TLS");
        sslContext.init(kmf.getKeyManagers(), null, null);

        // Create the server socket factory
        SSLServerSocketFactory ssf = sslContext.getServerSocketFactory();
        SSLServerSocket serverSocket = (SSLServerSocket) ssf.createServerSocket(9999);

        System.out.println("Server started. Waiting for client connection...");

        // Accept client connections
        SSLSocket socket = (SSLSocket) serverSocket.accept();

        // Set up input and output streams
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        // Read message from client
        String message = in.readLine();
```

```
            System.out.println("Received message from client: " + message);

            // Send response back to client
            out.println("Message received by server");

            // Close streams and socket
            out.close();
            in.close();
            socket.close();
            serverSocket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
import java.io.*;
import java.math.BigInteger;
class dh
{
  public static void main(String[]args)throws IOException
  {
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    System.out.println("Enter prime number:");
    BigInteger p=new BigInteger(br.readLine());
    System.out.print("Enter primitive root of "+p+":");
    BigInteger g=new BigInteger(br.readLine());
    System.out.println("Enter value for x less than "+p+":");
    BigInteger x=new BigInteger(br.readLine());
    BigInteger R1=g.modPow(x,p);
    System.out.println("R1="+R1);
    System.out.print("Enter value for y lessthan "+p+":");
```

```
    BigInteger y=new BigInteger(br.readLine());
    BigInteger R2=g.modPow(y,p);
    System.out.println("R2="+R2);
    BigInteger k1=R2.modPow(x,p);
    System.out.println("Key calculated at Sender's side:"+k1);
    BigInteger k2=R1.modPow(y,p);
    System.out.println("Key calculated at Receiver's side:"+k2);
    System.out.println("Diffie-Hellman secret key was calculated.");
  }
}
```