write a java program to implement data encryption standard algorithm for a practical application like user message encryption

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

public class DESEncryptionExample {

    public static void main(String[] args) {
        try {
            String originalMessage = "Hello, this is a secret message.";

            // Generate a DES key
            SecretKey secretKey = generateDESSecretKey();

            // Encrypt the message
            String encryptedMessage = encrypt(originalMessage, secretKey);
            System.out.println("Encrypted Message: " + encryptedMessage);

            // Decrypt the message
            String decryptedMessage = decrypt(encryptedMessage, secretKey);
            System.out.println("Decrypted Message: " + decryptedMessage);

        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }

    private static SecretKey generateDESSecretKey() throws Exception {
        // You can use a more secure way to generate a key, like using a KeyGenerator
        String keyString = "abcdefgh";
        byte[] keyData = keyString.getBytes();

        // Generate a DES key using DESKeySpec
        DESKeySpec desKeySpec = new DESKeySpec(keyData);
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
        return keyFactory.generateSecret(desKeySpec);
    }

    private static String encrypt(String message, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = cipher.doFinal(message.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    private static String decrypt(String encryptedMessage, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedMessage));
        return new String(decryptedBytes);
    }
}
```

---

write a java program to implement advanced encryption standard algorithm for a practical application like url encryption

```
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.io.UnsupportedEncodingException;

public class URLEncryption {

    private static final String KEY = "mySecretKey12345"; // Replace with a secure random key
    private static final String IV = "myInitializationVector"; // Replace with a secure random IV

    public static String encryptURL(String plainText) {
        try {
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            SecretKeySpec secretKey = new SecretKeySpec(KEY.getBytes("UTF-8"), "AES");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey, new IvParameterSpec(IV.getBytes("UTF-8")));

            byte[] encryptedBytes = cipher.doFinal(plainText.getBytes("UTF-8"));
            return Base64.getEncoder().encodeToString(encryptedBytes);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

```
    public static String decryptURL(String encryptedText) {
        try {
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            SecretKeySpec secretKey = new SecretKeySpec(KEY.getBytes("UTF-8"), "AES");
            cipher.init(Cipher.DECRYPT_MODE, secretKey, new IvParameterSpec(IV.getBytes("UTF-8")));

            byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
            return new String(decryptedBytes, "UTF-8");
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public static void main(String[] args) {
        String originalURL = "https://www.example.com";
        System.out.println("Original URL: " + originalURL);

        String encryptedURL = encryptURL(originalURL);
        System.out.println("Encrypted URL: " + encryptedURL);

        String decryptedURL = decryptURL(encryptedURL);
        System.out.println("Decrypted URL: " + decryptedURL);
    }
}
```

---

write a java program to implement RSA algorithm using HTML and javascript

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>RSA Encryption/Decryption</title>
</head>
<body>
    <h1>RSA Encryption/Decryption</h1>

    <label for="input">Enter Message:</label>
    <textarea id="input" rows="4" cols="50"></textarea>

    <button onclick="encrypt()">Encrypt</button>
    <button onclick="decrypt()">Decrypt</button>

    <h3>Encrypted Message:</h3>
    <p id="encrypted"></p>

    <h3>Decrypted Message:</h3>
    <p id="decrypted"></p>

    <script>
        // Basic RSA algorithm for educational purposes
        function generateKeyPair() {
            // Generate key pair logic goes here
```

```
    // For simplicity, let's assume we have pre-generated public and private keys
    const publicKey = { e: 65537, n: 1189 }; // Example values
    const privateKey = { d: 937, n: 1189 }; // Example values
    return { publicKey, privateKey };
}

function encrypt() {
    const message = document.getElementById('input').value;
    const { publicKey, _ } = generateKeyPair();
    const encryptedMessage = rsaEncrypt(message, publicKey);
    document.getElementById('encrypted').innerText = encryptedMessage;
}

function decrypt() {
    const encryptedMessage = document.getElementById('encrypted').innerText;
    const { _, privateKey } = generateKeyPair();
    const decryptedMessage = rsaDecrypt(encryptedMessage, privateKey);
    document.getElementById('decrypted').innerText = decryptedMessage;
}

function rsaEncrypt(message, publicKey) {
    // Encryption logic goes here
    // For simplicity, let's assume a basic encryption algorithm
    const encryptedMessage = message.split('').map(char => char.charCodeAt(0) ** publicKey.e % publicKey.n).join(',');
    return encryptedMessage;
}

function rsaDecrypt(encryptedMessage, privateKey) {
```

```
        // Decryption logic goes here
        // For simplicity, let's assume a basic decryption algorithm
        const decryptedMessage = encryptedMessage.split(',').map(code => String.fromCharCode(code ** privateKey.d % privateKey.n)).join('');
        return decryptedMessage;
      }
  </script>
</body>
</html>
```

---

write a java program to implement diffie hellman key exchange algorithm

```
import java.math.BigInteger;
import java.security.SecureRandom;

public class DiffieHellmanKeyExchange {

  public static void main(String[] args) {
    // Step 1: Select large prime numbers and a primitive root
    BigInteger p = new BigInteger("23");  // Example prime number
    BigInteger g = new BigInteger("5");   // Example primitive root

    // Step 2: Both parties generate a private key (a, b)
    BigInteger aPrivate = generatePrivateKey(p);
    BigInteger bPrivate = generatePrivateKey(p);
```

```
    // Step 3: Both parties calculate public key (A, B)
    BigInteger aPublic = calculatePublicKey(g, aPrivate, p);
    BigInteger bPublic = calculatePublicKey(g, bPrivate, p);

    // Step 4: Both parties exchange public keys

    // Step 5: Both parties calculate the shared secret key
    BigInteger sharedKeyA = calculateSharedKey(bPublic, aPrivate, p);
    BigInteger sharedKeyB = calculateSharedKey(aPublic, bPrivate, p);

    // Step 6: Verify that both shared keys are equal
    if (sharedKeyA.equals(sharedKeyB)) {
        System.out.println("Shared secret key: " + sharedKeyA);
    } else {
        System.out.println("Key exchange failed.");
    }
}

// Method to generate a random private key
private static BigInteger generatePrivateKey(BigInteger p) {
    SecureRandom random = new SecureRandom();
    return new BigInteger(p.bitLength() - 1, random).add(BigInteger.ONE);
}

// Method to calculate public key (A or B)
private static BigInteger calculatePublicKey(BigInteger g, BigInteger privateKey, BigInteger p) {
    return g.modPow(privateKey, p);
}
```

```
    // Method to calculate the shared secret key
    private static BigInteger calculateSharedKey(BigInteger publicKey, BigInteger privateKey, BigInteger p) {
        return publicKey.modPow(privateKey, p);
    }
}
```

---

write a java program to implement the signature scheme digital signature standard

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

public class DSSSignatureExample {

    public static void main(String[] args) {
        try {
            // Step 1: Generate key pair
            KeyPair keyPair = generateKeyPair();

            // Step 2: Sign a message
            String message = "Hello, this is a message to be signed!";
            byte[] signature = signMessage(message, keyPair.getPrivate());

            // Step 3: Verify the signature
            boolean isVerified = verifySignature(message, signature, keyPair.getPublic());
```

```
        System.out.println("Message: " + message);
        System.out.println("Signature: " + Base64.getEncoder().encodeToString(signature));
        System.out.println("Signature verified: " + isVerified);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// Method to generate a DSA key pair
private static KeyPair generateKeyPair() throws NoSuchAlgorithmException {
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
    SecureRandom secureRandom = new SecureRandom();
    keyPairGenerator.initialize(1024, secureRandom);
    return keyPairGenerator.generateKeyPair();
}

// Method to sign a message
private static byte[] signMessage(String message, PrivateKey privateKey)
        throws NoSuchAlgorithmException, InvalidKeyException, SignatureException {
    Signature signature = Signature.getInstance("SHA256withDSA");
    signature.initSign(privateKey);
    signature.update(message.getBytes());
    return signature.sign();
}

// Method to verify a signature
private static boolean verifySignature(String message, byte[] signature, PublicKey publicKey)
        throws NoSuchAlgorithmException, InvalidKeyException, SignatureException {
```

```
        Signature sig = Signature.getInstance("SHA256withDSA");
        sig.initVerify(publicKey);
        sig.update(message.getBytes());
        return sig.verify(signature);
    }
}
```

---

write a java program to calculate the message digest (hash) of a text using the SHA-1 algorithm

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

public class SHA1HashExample {

    public static void main(String[] args) {
        String inputText = "Hello, SHA-1!";

        try {
            // Step 1: Get a MessageDigest object with the SHA-1 algorithm
            MessageDigest sha1 = MessageDigest.getInstance("SHA-1");

            // Step 2: Update the message digest with the input text
            sha1.update(inputText.getBytes());

            // Step 3: Calculate the hash value
            byte[] hashBytes = sha1.digest();
```

```
            // Step 4: Convert the byte array to a hexadecimal string
            String hashHex = bytesToHex(hashBytes);

            System.out.println("Input Text: " + inputText);
            System.out.println("SHA-1 Hash: " + hashHex);

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }

    // Helper method to convert a byte array to a hexadecimal string
    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexStringBuilder = new StringBuilder();
        for (byte b : bytes) {
            hexStringBuilder.append(String.format("%02x", b));
        }
        return hexStringBuilder.toString();
    }
}
```

---

write a java program that simulates a dictionary attack on password by trying out a list of commonly used passwords and their variations

```
public class STANDictionaryAttack {
```

```
    public static void main(String[] args) {
        String[] commonPasswords = {"password", "123456", "qwerty", "admin", "letmein"};

        for (String password : commonPasswords) {
            System.out.println("Attempting password: " + password);
            boolean isPasswordCorrect = tryPassword(password);
            if (isPasswordCorrect) {
                System.out.println("Password found: " + password);
                break;
            }
        }
    }

    private static boolean tryPassword(String password) {
        // Simulate the password checking logic here
        // For demonstration purposes, let's say the correct password is "letmein"
        String correctPassword = "letmein";
        return password.equals(correctPassword);
    }
}
```

---

write a java program to implement MD5 algorithm

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

```
public class MD5Example {

    public static void main(String[] args) {
        String inputText = "Hello, MD5!";

        try {
            // Step 1: Get a MessageDigest object with the MD5 algorithm
            MessageDigest md5 = MessageDigest.getInstance("MD5");

            // Step 2: Update the message digest with the input text
            md5.update(inputText.getBytes());

            // Step 3: Calculate the hash value
            byte[] hashBytes = md5.digest();

            // Step 4: Convert the byte array to a hexadecimal string
            String hashHex = bytesToHex(hashBytes);

            System.out.println("Input Text: " + inputText);
            System.out.println("MD5 Hash: " + hashHex);

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }

    // Helper method to convert a byte array to a hexadecimal string
    private static String bytesToHex(byte[] bytes) {
```

```
        StringBuilder hexStringBuilder = new StringBuilder();
        for (byte b : bytes) {
            hexStringBuilder.append(String.format("%02x", b));
        }
        return hexStringBuilder.toString();
    }
}
```
