SHA

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class sha {
    public static String hashWithSHA512(String input) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-512");
           byte[] hashedBytes = md.digest(input.getBytes(StandardCharsets.UTF_8));
            StringBuilder hexString = new StringBuilder();
            for (byte b : hashedBytes) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }
            return hexString.toString();

        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("SHA-512 algorithm not found!", e);
        }
    }
    public static void main(String[] args) {
        String input = "koushik";
        String sha512Hash = hashWithSHA512(input);
        System.out.println("Input: " + input);
        System.out.println("SHA-512 Hash: " + sha512Hash);
    }
}
```

DES

```
import java.util.ArrayList;

import java.util.List;


public class DESKeyGeneration {

    private static final int[] PC1 = {

        57, 49, 41, 33, 25, 17, 9,

        1, 58, 50, 42, 34, 26, 18,

        10, 2, 59, 51, 43, 35, 27,

        19, 11, 3, 60, 52, 44, 36,

        63, 55, 47, 39, 31, 23, 15,

        7, 62, 54, 46, 38, 30, 22,

        14, 6, 61, 53, 45, 37, 29,

        21, 13, 5, 28, 20, 12, 4

    };


    private static final int[] PC2 = {

        14, 17, 11, 24, 1, 5,

        3, 28, 15, 6, 21, 10,

        23, 19, 12, 4, 26, 8,

        16, 7, 27, 20, 13, 2,

        41, 52, 31, 37, 47, 55,

        30, 40, 51, 45, 33, 48,

        44, 49, 39, 56, 34, 53,

        46, 42, 50, 36, 29, 32

    };


    private static final int[] SHIFTS = {

        1, 1, 2, 2, 2, 2, 2, 2,

        1, 2, 2, 2, 2, 2, 2, 1
```

```
};

private static String leftCircularShift(String input, int shift) {
    return input.substring(shift) + input.substring(0, shift);
}

private static String permute(String input, int[] table) {
    StringBuilder output = new StringBuilder();
    for (int index : table) {
        output.append(input.charAt(index - 1));
    }
    return output.toString();
}

public static List<String> generateKeys(String key64Bit) {
    String permutedKey = permute(key64Bit, PC1);
    String left = permutedKey.substring(0, 28);
    String right = permutedKey.substring(28);
    List<String> keys = new ArrayList<>();
    for (int i = 0; i < 16; i++) {
        left = leftCircularShift(left, SHIFTS[i]);
        right = leftCircularShift(right, SHIFTS[i]);
        String combinedKey = left + right;
        String roundKey = permute(combinedKey, PC2);
        keys.add(roundKey);
    }
    return keys;
}

public static void main(String[] args) {
        String keyHex = "133457799BBCDFF1";
        String key64Bit = new java.math.BigInteger(keyHex, 16).toString(2);
        key64Bit = String.format("%64s", key64Bit).replace(' ', '0');
        List<String> keys = generateKeys(key64Bit);
        for (int i = 0; i < keys.size(); i++) {
            System.out.printf("Round %2d Key: %s%n", i + 1, keys.get(i));
        }
    }
}
```

DSA

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.util.Scanner;

public class DigitalSignatureGenerator {
    public static void main(String[] args) {
        try {
            Scanner userInputScanner = new Scanner(System.in);
            System.out.print("Enter input: ");
            String userMessage = userInputScanner.nextLine();

            KeyPairGenerator keyGenerator = KeyPairGenerator.getInstance("DSA");
            keyGenerator.initialize(1024);
            KeyPair keyPair = keyGenerator.generateKeyPair();
            PrivateKey privateKey = keyPair.getPrivate();
            PublicKey publicKey = keyPair.getPublic();

            byte[] digitalSignature = generateSignature(userMessage, privateKey);
            System.out.println("Digital Signature: " + bytesToHexadecimal(digitalSignature));

            boolean isSignatureVerified = verifyDigitalSignature(userMessage, digitalSignature,
publicKey);
            System.out.println("Signature Verified: " + isSignatureVerified);

            userInputScanner.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static byte[] generateSignature(String data, PrivateKey privateKey) throws Exception {
        Signature signatureGenerator = Signature.getInstance("SHA1withDSA");
        signatureGenerator.initSign(privateKey);
        signatureGenerator.update(data.getBytes());
        return signatureGenerator.sign();
    }

    public static boolean verifyDigitalSignature(String data, byte[] signature, PublicKey publicKey)
throws Exception {
        Signature signatureVerifier = Signature.getInstance("SHA1withDSA");
        signatureVerifier.initVerify(publicKey);
        signatureVerifier.update(data.getBytes());
        return signatureVerifier.verify(signature);
    }
    public static String bytesToHexadecimal(byte[] bytes) {
        StringBuilder hexadecimalString = new StringBuilder();
        for (byte b : bytes) {
            String hexadecimal = Integer.toHexString(0xff & b);
            if (hexadecimal.length() == 1) hexadecimalString.append('0');
            hexadecimalString.append(hexadecimal);
        }
        return hexadecimalString.toString();
    }
}
```

MD5

```
import java.nio.charset.StandardCharsets;

public class MD5Algorithm {
    private static final int[] SHIFT_AMOUNTS = {
        7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
        5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
        4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
        6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21
    };

    private static final int[] TABLE_T = new int[64];

    static {
        for (int i = 0; i < 64; i++) {
            TABLE_T[i] = (int) (long) ((1L << 32) * Math.abs(Math.sin(i + 1)));
        }
```

```
}

private static int leftRotate(int x, int amount) {
   return (x << amount) | (x >>> (32 - amount));
}

private static byte[] padMessage(byte[] message) {
   int messageLength = message.length;
   int remainder = messageLength % 64;
   int paddingLength = (remainder < 56) ? (56 - remainder) : (64 + 56 - remainder);
   byte[] paddedMessage = new byte[messageLength + paddingLength + 8];
   System.arraycopy(message, 0, paddedMessage, 0, messageLength);
   paddedMessage[messageLength] = (byte) 0x80;
   long messageBitsLength = (long) messageLength * 8;
   for (int i = 0; i < 8; i++) {
      paddedMessage[paddedMessage.length - 8 + i] = (byte) (messageBitsLength >>> (8 * i));
   }
   return paddedMessage;
}

public static String computeMD5(String input) {
   byte[] message = padMessage(input.getBytes(StandardCharsets.UTF_8));
   int[] h = { 0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476 };

   for (int i = 0; i < message.length / 64; i++) {
      int[] block = new int[16];
      for (int j = 0; j < 16; j++) {
         block[j] = ((message[i * 64 + j * 4] & 0xff)) |
                 ((message[i * 64 + j * 4 + 1] & 0xff) << 8) |
                 ((message[i * 64 + j * 4 + 2] & 0xff) << 16) |
            ((message[i * 64 + j * 4 + 3] & 0xff) << 24);
}

int a = h[0], b = h[1], c = h[2], d = h[3];
for (int j = 0; j < 64; j++) {
  int f, g;
  if (j < 16) {
    f = (b & c) | (~b & d);
    g = j;
  } else if (j < 32) {
    f = (d & b) | (~d & c);
    g = (5 * j + 1) % 16;
  } else if (j < 48) {
    f = b ^ c ^ d;
    g = (3 * j + 5) % 16;
  } else {
    f = c ^ (b | ~d);
    g = (7 * j) % 16;
  }
  int temp = d;
  d = c;
  c = b;
  b = b + leftRotate(a + f + block[g] + TABLE_T[j], SHIFT_AMOUNTS[j]);
  a = temp;
}

h[0] += a;
h[1] += b;
h[2] += c;
h[3] += d;
```

```
        }

        StringBuilder md5 = new StringBuilder();
        for (int value : h) {
            for (int i = 0; i < 4; i++) {
                md5.append(String.format("%02x", (value >>> (i * 8)) & 0xff));
            }
        }
        return md5.toString();
    }

    public static void main(String[] args) {
        String input = "hello world";
        System.out.println("Input: " + input);
        System.out.println("MD5: " + computeMD5(input));
    }
}
```