Program 1:  
  
#include <stdio.h>

int main(){

    char \*str = "Hello";

    for (int i = 0; str[i] != '\0'; i++){

        char result = str[i] ^ 0;

        printf("%c", result);

    }

    return 0;

}

Program 2:

#include <stdio.h>

int main() {

    char \*str = "Hello";

    for (int i = 0; str[i] != '\0'; i++) {

        char ch = str[i];

printf("Character '%c' AND 127 = %c (ASCII: %d)\n", ch, ch & 127, ch & 127);

}

    printf("\n");

    for (int i = 0; str[i] != '\0'; i++) {

        char ch = str[i];

printf("Character '%c' AND 127 = %c (ASCII: %d)\n", ch, ch & 127, ch ^ 127);

}

    return 0;

}

Ceaser Cipher:

import java.util.\*;

public class CeaserCipher{

    public static String encrypt(String str, int shift){

        StringBuilder result = new StringBuilder();

        for (int i = 0; i < str.length(); i++){

            char ch = str.charAt(i);

            int x = 97;

            if (Character.isUpperCase(ch)){

                x = 65;

            }

            char c = (char)((ch + shift - x) % 26 + x);

            result.append(c);

        }

        return result.toString();

    }

    public static String decrypt(String str, int shift){

        return encrypt(str, 26 - shift);

    }

    public static void main(String[] args){

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the text for Ceaser Cipher: ");

        String str = sc.nextLine();

        System.out.println("Enter the shift value: ");

        int shift = sc.nextInt();

        String encrypted = encrypt(str, shift);

        System.out.println("Encrypted text: " + encrypted);

        String decrypted = decrypt(encrypted, shift);

        System.out.println("Decrypted text: " + decrypted);

    }

}

Substitution Cipher:

import java.io.\*;

import java.util.\*;

public class SubstitutionCipher {

    public static void main(String[] args) throws IOException {

        String a = "abcdefghijklmnopqrstuvwxyz";

        String b = "zyxwvutsrqponmlkjihgfedcba";

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter any string: ");

        String str = sc.next();

        String decrypt = "";

        char c;

        for (int i = 0; i < str.length(); i++) {

            c = str.charAt(i);

            int j = a.indexOf(c);

            decrypt = decrypt + b.charAt(j);

        }

        System.out.println("The encrypted data is: " + decrypt);

        sc.close();

    }

}

Hill Cipher:

import java.io.\*;

import java.util.\*;

public class HillCipher {

    static float[][] decrypt = new float[3][1];

    static float[][] a = new float[3][3];

    static float[][] b = new float[3][3];

    static float[][] mes = new float[3][1];

    static float[][] res = new float[3][1];

    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) throws IOException {

        getkeymes();

        for (int i = 0; i < 3; i++)

            for (int j = 0; j < 1; j++)

                for (int k = 0; k < 3; k++) {

                    res[i][j] = res[i][j] + a[i][k] \* mes[k][j];

                }

        System.out.print("\nEncrypted string is : ");

        for (int i = 0; i < 3; i++) {

            System.out.print((char) (res[i][0] % 26 + 97));

            res[i][0] = res[i][0];

        }

        inverse();

        for (int i = 0; i < 3; i++)

            for (int j = 0; j < 1; j++)

                for (int k = 0; k < 3; k++) {

                    decrypt[i][j] = decrypt[i][j] + b[i][k] \* res[k][j];

                }

        System.out.print("\nDecrypted string is : ");

        for (int i = 0; i < 3; i++) {

            System.out.print((char) (decrypt[i][0] % 26 + 97));

        }

        System.out.print("\n");

    }

    public static void getkeymes() throws IOException {

        System.out.println("Enter 3x3 matrix for key (It should be inversible): ");

        for (int i = 0; i < 3; i++)

            for (int j = 0; j < 3; j++)

                a[i][j] = sc.nextFloat();

        System.out.print("\nEnter a 3 letter string: ");

        String msg = sc.next();

        for (int i = 0; i < 3; i++)

            mes[i][0] = msg.charAt(i) - 97;

    }

    public static void inverse() {

        float p, q;

        float[][] c = a;

        for (int i = 0; i < 3; i++)

            for (int j = 0; j < 3; j++) {

                // a[i][j]=sc.nextFloat();

                if (i == j)

                    b[i][j] = 1;

                else

                    b[i][j] = 0;

            }

        for (int k = 0; k < 3; k++) {

            for (int i = 0; i < 3; i++) {

                p = c[i][k];

                q = c[k][k];

                for (int j = 0; j < 3; j++) {

                    if (i != k) {

                        c[i][j] = c[i][j] \* q - p \* c[k][j];

                        b[i][j] = b[i][j] \* q - p \* b[k][j];

                    }

                }

            }

        }

        for (int i = 0; i < 3; i++)

            for (int j = 0; j < 3; j++) {

                b[i][j] = b[i][j] / c[i][i];

            }

        System.out.println("");

        System.out.println("\nInverse Matrix is : ");

        for (int i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++)

                System.out.print(b[i][j] + " ");

            System.out.print("\n");

        }

    }

}

DES:

import java.util.\*;

import javax.crypto.\*;

public class DES {

    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);

    }

    public static void main(String[] args) {

        try{

            KeyGenerator keyGenerator = KeyGenerator.getInstance("DES");

            SecretKey secretKey = keyGenerator.generateKey();

            String plainText = "Hello, DES Algorithm";

            String encryptedText = encrypt(plainText, secretKey);

            System.out.println("Encrypted text: " + encryptedText);

            String decryptedText = decrypt(encryptedText, secretKey);

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

        }

        catch(Exception e){

            e.printStackTrace();

        }

    }

}

Blowfish:

import javax.crypto.\*;

import java.util.\*;

public class Blowfish {

    public static String encrypt(String plainText, SecretKey secretKey) throws Exception {

        Cipher cipher = Cipher.getInstance("Blowfish");

        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("Blowfish");

        cipher.init(Cipher.DECRYPT\_MODE, secretKey);

        byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));

        return new String(decryptedBytes);

    }

    public static void main(String[] args) {

        try {

            KeyGenerator keyGenerator = KeyGenerator.getInstance("Blowfish");

            SecretKey secretKey = keyGenerator.generateKey();

            String plainText = "Hello, Blowfish!";

            String encryptedText = encrypt(plainText, secretKey);

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

            String decryptedText = decrypt(encryptedText, secretKey);

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

        }

        catch (Exception e) {

            e.printStackTrace();

        }

    }

}

Rijndael:

import javax.crypto.\*;

import javax.crypto.spec.\*;

public class Rijndael {

    public static String asHex(byte buf[]) {

        StringBuffer strbuf = new StringBuffer(buf.length \* 2);

        for (int i = 0; i < buf.length; i++) {

            if (((int) buf[i] & 0xff) < 0x10)

                strbuf.append("0");

            strbuf.append(Long.toString((int) buf[i] & 0xff, 16));

        }

        return strbuf.toString();

    }

    public static void main(String[] args) throws Exception {

        String message = "AES still rocks!!";

        KeyGenerator kgen = KeyGenerator.getInstance("AES");

        kgen.init(128);

        SecretKey skey = kgen.generateKey();

        byte[] raw = skey.getEncoded();

        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

        Cipher cipher = Cipher.getInstance("AES");

        cipher.init(Cipher.ENCRYPT\_MODE, skeySpec);

        byte[] encrypted = cipher.doFinal(message.getBytes());

        System.out.println("encrypted string: " + asHex(encrypted));

        cipher.init(Cipher.DECRYPT\_MODE, skeySpec);

        byte[] original = cipher.doFinal(encrypted);

        String originalString = new String(original);

        System.out.println("Original string: " + originalString + " " + asHex(original));

    }

}  
  
RC4:

import javax.crypto.\*;

import java.util.\*;

public class RC4 {

    public static void main(String[] args) throws Exception {

        KeyGenerator keygenerator = KeyGenerator.getInstance("Blowfish");

        SecretKey secretkey = keygenerator.generateKey();

        Scanner sc = new Scanner(System.in);

        Cipher cipher = Cipher.getInstance("Blowfish");

        cipher.init(Cipher.ENCRYPT\_MODE, secretkey);

        System.out.print("Input your message: ");

        String inputText = sc.next();

        byte[] encrypted = cipher.doFinal(inputText.getBytes());

        cipher.init(Cipher.DECRYPT\_MODE, secretkey);

        byte[] decrypted = cipher.doFinal(encrypted);

        System.out.println("Encrypted text: " + new String(encrypted) + "\n" + "\nDecrypted text: " + new String(decrypted));

        sc.close();

    }

}

Diffie-Hellman Key Exchange:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Diffie-Hellman Key Exchange</title>

</head>

<body>

    <h1>Diffie-Hellman Key Exchange</h1>

    <div>

        <label for="prime">Prime Number (P): </label>

        <input type="number" id="prime" value="23">

        <br>

        <label for="base">Base (G): </label>

        <input type="number" id="base" value="5">

        <br>

        <label for="privateKey1">Private Key (User A): </label>

        <input type="number" id="privateKey1" value="6">

        <br>

        <label for="privateKey2">Private Key (User B): </label>

        <input type="number" id="privateKey2" value="15">

    </div>

    <button onclick="performKeyExchange()">Exchange Keys</button>

    <h2>Results</h2>

    <p id="result"></p>

    <script>

        function modularExponentiation(base, exp, mod) {

            let result = 1;

            base = base % mod;

            while (exp > 0) {

                if (exp % 2 === 1) {

                    result = (result \* base) % mod;

                }

                exp = Math.floor(exp / 2);

                base = (base \* base) % mod;

            }

            return result;

        }

        function performKeyExchange() {

            const prime = parseInt(document.getElementById('prime').value);

            const base = parseInt(document.getElementById('base').value);

            const privateKey1 = parseInt(document.getElementById('privateKey1').value);

            const privateKey2 = parseInt(document.getElementById('privateKey2').value);

            const publicKey1 = modularExponentiation(base, privateKey1, prime);

            const publicKey2 = modularExponentiation(base, privateKey2, prime);

            const sharedKey1 = modularExponentiation(publicKey2, privateKey1, prime);

            const sharedKey2 = modularExponentiation(publicKey1, privateKey2, prime);

            const resultElement = document.getElementById('result');

            resultElement.innerHTML = `

                <strong>Public Key (User A):</strong> ${publicKey1}<br>

                <strong>Public Key (User B):</strong> ${publicKey2}<br>

                <strong>Shared Secret Key (User A):</strong> ${sharedKey1}<br>

                <strong>Shared Secret Key (User B):</strong> ${sharedKey2}<br>

            `;

            if (sharedKey1 === sharedKey2) {

                resultElement.innerHTML += "<strong>Keys Match! Secure Communication Established.</strong>";

            } else {

                resultElement.innerHTML += "<strong>Keys do not match. Something went wrong.</strong>";

            }

        }

    </script>

</body>

</html>

SHA-1:

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Scanner;

public class SHA {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the text to generate SHA-1 hash:");

        String inputText = scanner.nextLine();

        scanner.close();

        try {

            MessageDigest md = MessageDigest.getInstance("SHA-1");

            byte[] messageDigest = md.digest(inputText.getBytes());

            StringBuilder hexString = new StringBuilder();

            for (byte b : messageDigest) {

                String hex = Integer.toHexString(0xff & b);

                if (hex.length() == 1) {

                    hexString.append('0');

                }

                hexString.append(hex);

            }

            System.out.println("SHA-1 Hash: " + hexString.toString());

        }

        catch (NoSuchAlgorithmException e) {

            System.err.println("SHA-1 algorithm not found!");

        }

    }

}

MD5:

import java.security.\*;

public class MD5 {

    public static void main(String[] a) {

        try {

            MessageDigest md = MessageDigest.getInstance("MD5");

            System.out.println("Message digest object info: ");

            System.out.println(" Algorithm = " + md.getAlgorithm());

            System.out.println(" Provider = " + md.getProvider());

            System.out.println(" ToString = " + md.toString());

            String input = "abc";

            md.update(input.getBytes());

            byte[] output = md.digest();

            System.out.println();

            System.out.println("MD5(\"" + input + "\") = " + bytesToHex(output));

        } 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();

    }

}