PROGRAM:1 - Ceaser Cipher

```
import java.util.*;

class CaesarCipher {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int shift, i, n, p, key;
        String str;
        String str1 = "";
        System.out.println("Enter the Plain Text");
        str = sc.nextLine();
        str = str.toLowerCase();
        n = str.length();
        char ch1[] = str.toCharArray();
        char ch4;
        System.out.println("Enter the value by which each letter of the string is to be shifted");
        shift = sc.nextInt();
        System.out.println();
        System.out.println("Encrypted text is:");
        for (i = 0; i < n; i++) {
            if (Character.isLetter(ch1[i])) {
                ch4 = (char) (((int) ch1[i] + shift - 97) % 26 + 97);
                str1 = str1 + ch4;
            } else if (ch1[i] == ' ') {
                str1 = str1 + ch1[i];
            }
        }
        System.out.println(str1);
        System.out.println("Cipher Text:" + str1);
        n = str1.length();
        char ch2[] = str1.toCharArray();
        char ch3;
        System.out.println();
        System.out.println("Possible Plain text is");
        str1 = "";
        for (key = 26; key >= 1; key--) {
            for (i = 0; i < n; i++) {
                if (Character.isLetter(ch2[i])) {
                    ch3 = (char) (((int) ch2[i] + key - 97) % 26 + 97);
                    str1 = str1 + ch3;
                } else if (ch2[i] == ' ') {
                    str1 = str1 + ch2[i];
                }
            }
            p = 26 - key;
            System.out.println("For Key " + p + ":" + str1);
            str1 = "";
        }
    }
}
```

PROGRAM:2 – One time PaD ENCRYPTION ALGORITHM

```
import java.util.Scanner;

public class OTP {


    private static String adjustKeyLength(String text, String key) {
        int textLength = text.length();
        int keyLength = key.length();

        if (keyLength < textLength) {
            key = key.repeat((textLength / keyLength) + 1);
        }
        return key.substring(0, textLength);
    }

    public static String stringEncryption(String text, String key) {
        key = adjustKeyLength(text, key);
        StringBuilder cipherText = new StringBuilder();

        for (int i = 0; i < text.length(); i++) {
            int cipherValue = (text.charAt(i) - 'A' + key.charAt(i) - 'A') % 26;
            cipherText.append((char) (cipherValue + 'A'));
        }
        return cipherText.toString();
    }

    public static String stringDecryption(String cipherText, String key) {
        key = adjustKeyLength(cipherText, key);
        StringBuilder plainText = new StringBuilder();

        for (int i = 0; i < cipherText.length(); i++) {
            int plainValue = (cipherText.charAt(i) - 'A' - (key.charAt(i) - 'A') + 26) % 26;
            plainText.append((char) (plainValue + 'A'));
        }
        return plainText.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter plain text (A-Z only): ");
        String plainText = sc.nextLine().toUpperCase().replaceAll("[^A-Z]", "");

        System.out.print("Enter key (A-Z only): ");
        String key = sc.nextLine().toUpperCase().replaceAll("[^A-Z]", "");

        if (plainText.isEmpty() || key.isEmpty()) {
            System.out.println("Invalid input! Only A-Z characters are allowed.");
            sc.close();
            return;
        }

        String encryptedText = stringEncryption(plainText, key);
        System.out.println("Cipher Text - " + encryptedText);

        String decryptedMessage = stringDecryption(encryptedText, key);
        System.out.println("Decrypted Message - " + decryptedMessage);

        sc.close();
    }
}
```

PROGRAM:3 – Monoalphabetic Cipher

```
import java.util.Scanner;

public class MonoalphabeticCipher {
    public static char p[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
            'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
            'w', 'x', 'y', 'z' };
    public static char ch[] = { 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O',
            'P', 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'Z', 'X', 'C',
            'V', 'B', 'N', 'M' };
    static String str;
    public static String doEncryption(String s) {
        char c[] = new char[(s.length())];
        for (int i = 0; i < s.length(); i++) {
            for (int j = 0; j < 26; j++) {
                if (p[j] == s.charAt(i)) {
                    c[i] = ch[j];
                    break;
                }
            }
        }
        return (new String(c));
    }

    public static String doDecryption(String s) {
        char p1[] = new char[(s.length())];
        for (int i = 0; i < s.length(); i++) {
            for (int j = 0; j < 26; j++) {
                if (ch[j] == s.charAt(i)) {
                    p1[i] = p[j];
                    break;
                }
            }
        }
```

```
            // Decrypt the text
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            byte[] decryptedBytes =
cipher.doFinal(Base64.getDecoder().decode(encryptedText));
            String decryptedText = new String(decryptedBytes);
            System.out.println("Decrypted Text: " + decryptedText);

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

```
        return (new String(p1));
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the plain text: ");
        str = sc.next();
        String en = doEncryption(str.toLowerCase());
        System.out.println("Encrypted message: " + en);
        System.out.println("Decrypted message: " +
doDecryption(en));
        sc.close();
    }
}
```

PROGRAM:4 – DES Cipher

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

PROGRAM:5 – AES ENCRYTION ALGORITHM

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

public class Main {
    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(System.in);

            // Generate an AES key
            KeyGenerator keyGenerator =
KeyGenerator.getInstance("AES");
            keyGenerator.init(128); // AES-128 bit key
            SecretKey secretKey = keyGenerator.generateKey();
```

```
public class DESUserInput {
    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(System.in);

            // Generate a DES key
            KeyGenerator keyGenerator =
KeyGenerator.getInstance("DES");
            SecretKey secretKey = keyGenerator.generateKey();

            // Create a Cipher for DES encryption and decryption
            Cipher cipher = Cipher.getInstance("DES");

            // Take user input
            System.out.print("Enter text to encrypt: ");
            String plainText = scanner.nextLine();

            // Encrypt the text
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
            String encryptedText =
Base64.getEncoder().encodeToString(encryptedBytes);
            System.out.println("Encrypted Text: " + encryptedText);
```

```
            // Create a Cipher for AES encryption and decryption
            Cipher cipher = Cipher.getInstance("AES");

            // Take user input
            System.out.print("Enter text to encrypt: ");
            String plainText = scanner.nextLine();

            // Encrypt the text
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
            String encryptedText =
Base64.getEncoder().encodeToString(encryptedBytes);
            System.out.println("Encrypted Text: " + encryptedText);

            // Decrypt the text
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            byte[] decryptedBytes =
cipher.doFinal(Base64.getDecoder().decode(encryptedText));
            String decryptedText = new String(decryptedBytes);
            System.out.println("Decrypted Text: " + decryptedText);

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

PROGRAM:6 – Diffie – Hellman Key Establishment

```
import java.net.*;
import java.io.*;

public class DHClient {
```

```
    public static void main(String[] args) {

        try {

            String pstr, gstr, Astr;
            String serverName = "localhost";
            int port = 8088;

            // Declare p, g, and Key of client
            int p = 23;
            int g = 9;
            int a = 4;
            double Adash, serverB;

            System.out.println("Connecting to " + serverName + " on
port " + port);
            Socket client = new Socket(serverName, port);
            System.out.println("Just connected to " +
client.getRemoteSocketAddress());

            // Sends the data to client
            OutputStream outToServer = client.getOutputStream();
```

```
            DataOutputStream out = new
DataOutputStream(outToServer);

            pstr = Integer.toString(p);
            out.writeUTF(pstr); // Sending p

            gstr = Integer.toString(g);
            out.writeUTF(gstr); // Sending g

            double A = ((Math.pow(g, a)) % p);
            Astr = Double.toString(A);
            out.writeUTF(Astr); // Sending A

            // Client's Private Key
            System.out.println("From Client : Private Key = " + a);

            // Accepts the data
            DataInputStream in = new
DataInputStream(client.getInputStream());

            serverB = Double.parseDouble(in.readUTF());
            System.out.println("From Server : Public Key = " + serverB);
            Adash = ((Math.pow(serverB, a)) % p);
```

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


//Server side program
import java.net.*;
import java.io.*;

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

        try {

            int port = 8088;

            double clientP, clientG, clientA, B, Bdash;
            String Bstr;
            int b = 3;
```

```
            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Waiting for client on port " +
serverSocket.getLocalPort());

            Socket server = serverSocket.accept();
            System.out.println("Just connected to " +
server.getRemoteSocketAddress());

            System.out.println("From Server : Private Key = " + b);

            // Accepts the data from client
            DataInputStream in = new
DataInputStream(server.getInputStream());
            clientP = Integer.parseInt(in.readUTF());
            clientG = Integer.parseInt(in.readUTF());
            clientA = Double.parseDouble(in.readUTF());
            B = ((Math.pow(clientG, b)) % clientP);
            Bstr = Double.toString(B);
            OutputStream outToclient = server.getOutputStream();
            DataOutputStream out = new
DataOutputStream(outToclient);

            out.writeUTF(Bstr);
```

```
            Bdash = ((Math.pow(clientA, b)) % clientP);

            server.close();
        } catch (SocketTimeoutException s) {
            System.out.println("Socket timed out!");
        } catch (IOException e) {
        }
    }
}

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

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the text to sign: ");
        String msg = sc.nextLine();
        KeyPairGenerator keyPairGen =
KeyPairGenerator.getInstance("DSA");
        keyPairGen.initialize(2048);
        KeyPair pair = keyPairGen.generateKeyPair();
        PrivateKey privKey = pair.getPrivate();
        PublicKey pubKey = pair.getPublic();

        Signature sign = Signature.getInstance("SHA256withDSA");
        sign.initSign(privKey);
        sign.update(msg.getBytes());

        byte[] signature = sign.sign();
        System.out.println("\nGenerated Digital Signature:");
        for (byte b : signature) {
            System.out.printf("%02x", b);
        }
        System.out.println();

        sign.initVerify(pubKey);
        sign.update(msg.getBytes());
        boolean isValid = sign.verify(signature);
```

```
        System.out.println(isValid ? "\nSignature verified
successfully." : "\nSignature verification failed.");

        sc.close();
    }
}
```

PROGRAM:7 implement Cryptographic Hash Function (SHA-256)

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class SHA {

    public static String getSHA(String input) {
        try {
            // Creating MessageDigest instance for SHA-256
            MessageDigest hash = MessageDigest.getInstance("SHA-
256");

            // Calculating message digest of input
```

```
            byte[] messageDigest = hash.digest(input.getBytes());

            // Converting byte array into signum representation
            BigInteger no = new BigInteger(1, messageDigest);

            // Converting message digest into hexadecimal format
            String hashtext = no.toString(16);

            // Padding with leading zeros to make it 64 characters
            while (hashtext.length() < 64) {
                hashtext = "0" + hashtext;
            }

            return hashtext;

        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }

    public static void main(String[] args) {
        // Accepting dynamic input from user
        Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter the text to generate SHA-256 hash: ");
        String input = sc.nextLine();

        // Displaying the generated hash code
        System.out.println("Generated SHA-256 Hash: " +
getSHA(input));

        sc.close();
    }
}
```

PROGRAM 8: implement Message authentication codes (MD5)

```
import java.math.BigInteger;
```

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

public class MD5 {
    public static String getMd5(String input) {
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            byte[] messageDigest = md.digest(input.getBytes());
            BigInteger no = new BigInteger(1, messageDigest);
            String hashtext = no.toString(16);
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            return hashtext;
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the string to generate MD5 hash: ");
```

```
        String input = scanner.nextLine();
        System.out.println("Generated MD5 hash: " + getMd5(input));
        scanner.close();
    }
}
```

PROGRAM : 9 implement Public Key Cryptosystems (RSA)

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

public class RSADemo {
    private final static BigInteger one = new BigInteger("1");
    private final static SecureRandom random = new
SecureRandom();

    private BigInteger privateKey;
    private BigInteger publicKey;
    private BigInteger modulus;

    // Generate an N-bit (roughly) public and private key
    RSADemo(int N) {
        BigInteger p = BigInteger.probablePrime(N / 2, random);
        BigInteger q = BigInteger.probablePrime(N / 2, random);
        BigInteger phi = (p.subtract(one)).multiply(q.subtract(one));
        System.out.println("prime p = " + p);
        System.out.println("prime q = " + q);

        modulus = p.multiply(q);
        System.out.println("modulus = " + modulus);
        System.out.println("phi = " + phi);

        publicKey = new BigInteger("65537"); // common value in practice = 2^16 + 1
        privateKey = publicKey.modInverse(phi);
    }

    BigInteger encrypt(BigInteger message) {
        return message.modPow(publicKey, modulus);
    }

    BigInteger decrypt(BigInteger encrypted) {
        return encrypted.modPow(privateKey, modulus);
    }
```

```
    public String toString() {
        String s = "";
        s += "public = " + publicKey + "\n";
        s += "private = " + privateKey + "\n";
        s += "modulus  = " + modulus;

        return s;
    }

    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Usage: java RSADemo <key size in bits>");
            return;
        }

        int N = Integer.parseInt(args[0]);
        RSADemo key = new RSADemo(N);
        System.out.println(key);

        // Create random message, encrypt and decrypt
        BigInteger message = new BigInteger("8");
```

```
        // Create message by converting string to integer
        // String s = "test";
        // byte[] bytes = s.getBytes();
        // BigInteger message = new BigInteger(bytes);

        BigInteger encrypt = key.encrypt(message);
        BigInteger decrypt = key.decrypt(encrypt);
        System.out.println("message = " + message);
        System.out.println("encrypted = " + encrypt);
        System.out.println("decrypted = " + decrypt);
    }
}
```