AES

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class Main {

    private static SecretKeySpec secretKey;
    private static byte[] key;

    public static void setKey(String myKey)
    {
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16); // use only first 128 bit
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            System.out.println("[ERR]: " + e.toString());
        }
        catch (UnsupportedEncodingException e) {
            System.out.println("[ERR]: " + e.toString());

        }
    }

    public static String encrypt(String text, String key)
    {
        try
        {
            setKey(key);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return
Base64.getEncoder().encodeToString(cipher.doFinal(text.getBytes("UTF-8"
)));
```

```
        }
        catch (Exception e)
        {
            System.out.println("[ERR]: " + e.toString());
        }
        return "";
    }

    public static String decrypt(String text, String key)
    {
        // text is already base64 encoded
        try
        {
            setKey(key);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new
String(cipher.doFinal(Base64.getDecoder().decode(text)));
        }
        catch (Exception e)
        {
            System.out.println("[ERR]: " + e.toString());
        }
        return "";
    }
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        String text, key;
        System.out.print("Enter text to encrypt: ");
        text = s.nextLine();

        System.out.print("Enter key: ");
        key = s.nextLine();

        String encrypted = Main.encrypt(text, key) ;
        System.out.println("ENCRYPTED TEXT: " + encrypted);

        String decrypted = Main.decrypt(encrypted, key) ;
        System.out.println("DECRYPTED TEXT: " + decrypted);
    }
}
```

AES key gen

```
import java.util.*;

public class Main {

    public static char Sbox[] = {
        0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01,
0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4,
0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5,
0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12,
0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b,
0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb,
0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9,
0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6,
0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
        0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7,
0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee,
0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
        0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3,
0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
        0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56,
0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
        0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd,
0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
        0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35,
0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
        0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e,
0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
        0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99,
0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16};

    public static char Rcon[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
0x40, 0x80, 0x1B, 0x36};

    public static void printMatrix(char[] matrix) {
```

```
        for (int i = 0; i < 16; ++i) {
            System.out.print("" + String.format("%02X", (int)
matrix[i]) + " ");
            if ((i + 1) % 4 == 0) {
                System.out.println();
            }
        }
    }

    public static char[] RotWord(char[] word) {
        return new char[]{word[1], word[2], word[3], word[0]};
    }

    public static char[] SubWord(char[] word) {
        char[] sub = new char[4];
        for (int i = 0; i < 4; i++) {
            sub[i] = Sbox[word[i]];
        }
        return sub;
    }

    public static char[] xor(char[] a, char[] b) {
        char[] ans = new char[4];
        for (int i = 0; i < 4; ++i) {
            ans[i] = (char) (a[i] ^ b[i]);
        }
        return ans;
    }

    public static char[][] KeyExpansion(char key[]) {
        char[] temp = new char[4];
        char[][] w = new char[44][4];

        for (int i = 0; i < 4; ++i) {
            for (int j = 0; j < 4; ++j) {
                w[i][j] = key[4 * i + j];
            }
        }

        for (int i = 4; i < 44; i++) {
            temp = w[i - 1];
            if (i % 4 == 0) {
                temp = SubWord(RotWord(temp));
```

```
                temp[0] = (char) (temp[0] ^ (char) Rcon[i / 4 - 1]);
            }
            w[i] = xor(w[i - 4], temp);
        }
        return w;
    }

    public static void main(String args[]) {
        // char plainText[][] = new char[1];
        char cipher[] = new char[36];
        System.out.print("Enter the key : ");
        Scanner in = new Scanner(System.in);
        String input = in.nextLine();

        // make the key 32 bytes
        if (input.length() > 32) {
            input = input.substring(0, 32);
        }
        while (input.length() < 32) {
            input += "0";
        }
        char t[] = input.toCharArray();
        for (int i = 0; i < 32; i += 2) {
            cipher[i / 2] = (char) Integer.parseInt("" + t[i] + t[i +
1], 16);
        }

        System.out.println("Cipher key :");
        printMatrix(cipher);

        char[][] w = KeyExpansion(cipher);
        for (int i = 0; i < 44; ++i) {
            System.out.print("Word " + i + " ");
            for (int j = 0; j < 4; j++) {
                System.out.print("" + String.format("%02X", (int)
w[i][j]) + " ");
            }
            System.out.println();
        }
    }

}
```

Caesar

```
import java.util.*;
import java.io.*;

public class Main {

    public static final String text =
"abcdefghijklmnopqrstuvwxyz";//initializing public string

    static String encrypt(String plain, int key) {
        int l = plain.length();
        String s = "";
        int i = 0;
        int x;
        char a, y;
        for (i = 0; i < l; i++) {
            a = plain.charAt(i);
            if (a != ' ') {
                x = text.indexOf(a);
                y = text.charAt((x + key) % 26);
                s += y;
            } else {
                s += a;
            }
        }
        return s;
    }

    static String decrypt(String cipher, int key) {
        int l = cipher.length();
        String s = "";
        int i = 0;
        int x, z;
        char a, y;
        for (i = 0; i < l; i++) {
            a = cipher.charAt(i);
            if (a != ' ') {
                x = text.indexOf(a);
                z = (x - key);
                if (z < 0)
                    z = z + text.length();
                y = text.charAt(z % 26);
                s += y;
```

```
            } else {
                s += a;
            }

        }
        return s;
    }

    public static void main(String args[]) throws IOException {
        int ch, key = 0;
        String plain = "", cipher = "", c, result;
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ir);
        do {
            System.out.println("Caesar Cipher");
            System.out.println("1. Read Plaintext");
            System.out.println("2. Read Key");
            System.out.println("3. Encrypt");
            System.out.println("4. Decrypt");
            ch = Integer.parseInt(br.readLine());
            switch (ch) {
                case 1: {
                    System.out.println("Enter Plaintext");
                    plain = br.readLine();
                    break;
                }
                case 2: {
                    System.out.println("Enter key");
                    key = Integer.parseInt(br.readLine());
                    break;
                }
                case 3: {
                    cipher = encrypt(plain.toLowerCase(), key);
                    System.out.println("Encypted String:" + cipher);
                    break;
                }
                case 4: {
                    result = decrypt(cipher.toLowerCase(), key);
                    System.out.println("Decrypted String:" + result);
                    break;
                }
            }
            System.out.println("Do you want to exit?(y/n)");
```

```
            c = br.readLine();
        } while (!(c.equals("y")));
    }
}
```

Encryption

```
import java.util.*;
import java.io.*;

public class Main {

    public static final String text = "abcdefghijklmnopqrstuvwxyz";

    static String encrypt(String plain, int key) {
        int l = plain.length();
        String s = "";
        int i = 0;
        int x;
        char a, y;
        for (i = 0; i < l; i++) {
            a = plain.charAt(i);
            if (a != ' ') {
                x = text.indexOf(a);
                y = text.charAt((x + key) % 26);
                s += y;
            } else {
                s += a;
            }
        }
        return s;
    }

    public static void main(String args[]) throws IOException {
        int ch, key = 0;
        String plain = "", cipher = "", c, result;
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ir);
        do {
            System.out.println("Caesar Cipher");
            System.out.println("1. Read Plaintext");
            System.out.println("2. Read Key");
            System.out.println("3. Encrypt");
            ch = Integer.parseInt(br.readLine());
```

```
            switch (ch) {
                case 1: {
                    System.out.println("Enter Plaintext");
                    plain = br.readLine();
                    break;
                }
                case 2: {
                    System.out.println("Enter key");
                    key = Integer.parseInt(br.readLine());
                    break;
                }
                case 3: {
                    cipher = encrypt(plain.toLowerCase(), key);
                    System.out.println("Encypted String:" + cipher);
                    break;
                }
            }
            System.out.println("Do you want to exit?(y/n)");
            c = br.readLine();
        } while (!(c.equals("y")));
    }
}
```

Decryption

```
import java.util.*;
import java.io.*;

public class Main {

    public static final String text = "abcdefghijklmnopqrstuvwxyz";

        static String decrypt(String cipher, int key) {
        int l = cipher.length();
        String s = "";
        int i = 0;
        int x, z;
        char a, y;
        for (i = 0; i < l; i++) {
            a = cipher.charAt(i);
            if (a != ' ') {
                x = text.indexOf(a);
                z = (x - key);
```

```
            if (z < 0)
                z = z + text.length();
            y = text.charAt(z % 26);
            s += y;
        } else {
            s += a;
        }

    }
    return s;
}

public static void main(String args[]) throws IOException {
    int ch, key = 0;
    String plain = "", cipher = "", c, result;
    InputStreamReader ir = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(ir);
    do {
        System.out.println("Caesar Cipher");
        System.out.println("1. Read encrypted text");
        System.out.println("2. Read Key");
        System.out.println("3. Decrypt");
        ch = Integer.parseInt(br.readLine());
        switch (ch) {
            case 1: {
                System.out.println("Enter Plaintext");
                plain = br.readLine();
                break;
            }
            case 2: {
                System.out.println("Enter key");
                key = Integer.parseInt(br.readLine());
                break;
            }
            case 3: {
                result = decrypt(plain.toLowerCase(), key);
                System.out.println("Decrypted String:" + result);
                break;
            }
        }
        System.out.println("Do you want to exit?(y/n)");
        c = br.readLine();
    } while (!(c.equals("y")));
```

```
    }
}
```

DES

```
import java.lang.StringBuffer;
import java.math.*;
import java.util.*;

public class Main {
    static 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 };

    static 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 };

    static int[] ip = { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36,
28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64,
            56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1,
59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29,
            21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7 };

    static int[] ip_inv = { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47,
15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30,
            37, 5, 45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60,
28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42, 10,
            50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25 };

    static int[] expansion = { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8,
9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17, 16,
            17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25, 26, 27,
28, 29, 28, 29, 30, 31, 32, 1 };

    static int[] permutation_32 = { 16, 7, 20, 21, 29, 12, 28, 17, 1,
15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27,
            3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };

    static int[][] s1 = { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12,
5, 9, 0, 7 },
```

```
            { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
            { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
            { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };

    static int[][] s2 = { { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12,
0, 5, 10 },
            { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
            { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
            { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };

    static int[][] s3 = { { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7,
11, 4, 2, 8 },
            { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
            { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
            { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };
    static int[][] s4 = { { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11,
12, 4, 15 },
            { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
            { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
            { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };
    static int[][] s5 = { { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13,
0, 14, 9 },
            { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
            { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
            { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };
    static int[][] s6 = { { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14,
7, 5, 11 },
            { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
            { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
            { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };
    static int[][] s7 = { { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5,
10, 6, 1 },
            { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
            { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
            { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };

    static int[][] s8 = { { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5,
0, 12, 7 },
            { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
            { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
            { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };

    static String[] subkeys = new String[16];
```

```
    public static String leftShift(String key, int amount) {
        for (int i = 0; i < amount; i++) {
            char first = key.charAt(0);
            key = key.substring(1, key.length()) + first;
        }
        return key;
    }

    public static String generatePC1(String key) {
        char[] array = new char[pc1.length];
        int c = 0;
        for (int i : pc1) {
            array[c] = key.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String generatePC2(String key) {
        char[] array = new char[pc2.length];
        int c = 0;
        for (int i : pc2) {
            array[c] = key.charAt(i - 1);
            c++;
        }

        return String.valueOf(array);
    }

    public static void keygen(String key) {
        String bin = new BigInteger(key, 16).toString(2); // hex to
binary
        String left, right;
        while (bin.length() != 64)
            bin = "0" + bin;
        key = generatePC1(bin);
        left = key.substring(0, key.length() / 2);
        right = key.substring(key.length() / 2, key.length());

        // GENERATE 16 SUBKEYS
        for (int i = 1; i < 17; i++) {
            if (i == 1 || i == 2 || i == 9 || i == 16) {
```

```
            // left shift once
            left = leftShift(left, 1);
            right = leftShift(right, 1);
        } else {
            // left shift twice
            left = leftShift(left, 2);
            right = leftShift(right, 2);
        }

        key = "" + left + right;
        // System.out.println(key);
        subkeys[i - 1] = generatePC2(key);
    }
}

public static String toHex(String input) {
    String hex = "";
    for (int i = 0; i < input.length(); i++)
        hex += Integer.toHexString(input.charAt(i));
    return hex;
}

public static String getIP(String input) {
    char[] array = new char[ip.length];
    int c = 0;
    for (int i : ip) {
        array[c] = input.charAt(i - 1);
        c++;
    }
    return String.valueOf(array);

}

public static String expansion(String input) {
    char[] array = new char[expansion.length];
    int c = 0;
    for (int i : expansion) {
        array[c] = input.charAt(i - 1);
        c++;
    }
    return String.valueOf(array);

}
```

```
public static String xorStrings(String s1, String s2) {
    String output = "";
    for (int i = 0; i < s1.length(); i++) {
        if (s1.charAt(i) == s2.charAt(i))
            output += "0";
        else
            output += "1";
    }
    return output;
}

public static String applySbox(String input, int number) {
    String row, col, output = "";
    int r, c;
    row = "" + input.charAt(0) + input.charAt(input.length() - 1);
    col = input.substring(1, 5);
    r = Integer.parseInt(row, 2);
    c = Integer.parseInt(col, 2);
    switch (number) {
        case 0:
            output = Integer.toBinaryString(s1[r][c]);
            break;
        case 1:
            output = Integer.toBinaryString(s2[r][c]);
            break;
        case 2:
            output = Integer.toBinaryString(s3[r][c]);
            break;
        case 3:
            output = Integer.toBinaryString(s4[r][c]);
            break;
        case 4:
            output = Integer.toBinaryString(s5[r][c]);
            break;
        case 5:
            output = Integer.toBinaryString(s6[r][c]);
            break;
        case 6:
            output = Integer.toBinaryString(s7[r][c]);
            break;
        case 7:
            output = Integer.toBinaryString(s8[r][c]);
```

```
            break;
        }
        while (output.length() < 4)
            output = "0" + output;
        return output;
    }

    public static String applySboxes(String input) {
        String output = "";
        for (int i = 0; i < 8; i++) {
            output += applySbox(input.substring(i * 6, i * 6 + 6), i);
        }
        return output;
    }

    public static String getPermutation32(String input) {
        char[] array = new char[permutation_32.length];
        int c = 0;
        for (int i : permutation_32) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String getIPinverse(String input) {
        char[] array = new char[ip_inv.length];
        int c = 0;
        for (int i : ip_inv) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String feistalRound(String right, String key) {
        right = expansion(right);
        String output = xorStrings(right, key);
        output = applySboxes(output);
        output = getPermutation32(output);
        return output;
    }
```

```
    public static String encryptDecrypt(String text, boolean decrypt) {
        String output = "", bin = "";
        String t = "" + text;
        int iter = t.length() / 16;
        if (t.length() % 16 != 0)
            iter++;

        for (int z = 0; z < iter; z++) {
            if (z + 1 < iter)
                bin = new BigInteger(t.substring(z * 16, z * 16 + 16),
16).toString(2);
            else
                bin = new BigInteger(t.substring(z * 16, t.length()),
16).toString(2);

            while (bin.length() < 64)
                bin = "0" + bin;
            bin = getIP(bin);
            String temp, left, right;
            left = bin.substring(0, bin.length() / 2);
            right = bin.substring(bin.length() / 2, bin.length());

            if (!decrypt)
                for (int i = 0; i < 16; i++) {
                    temp = left;
                    left = right;
                    right = xorStrings(temp, feistalRound(right,
subkeys[i]));
                }
            else
                for (int i = 0; i < 16; i++) {
                    temp = left;
                    left = right;
                    right = xorStrings(temp, feistalRound(right,
subkeys[15 - i]));
                }

            text = right + left;
            text = getIPinverse(text);
            text = new BigInteger(text, 2).toString(16);

            while (text.length() < 16)
                text = "0" + text;
```

```
            output += text;
        }

        return output;
    }

    public static String stringToHex(String str) {
        StringBuffer sb = new StringBuffer();
        // Converting string to character array
        char charArray[] = str.toCharArray();
        for (int i = 0; i < charArray.length; i++) {
            String hexString = Integer.toHexString(charArray[i]);
            sb.append(hexString);
        }
        return sb.toString();
    }

    public static String hexToString(String str) {
        String result = new String();
        char[] charArray = str.toCharArray();
        for (int i = 0; i < charArray.length; i = i + 2) {
            String st = "" + charArray[i] + "" + charArray[i + 1];
            char ch = (char) Integer.parseInt(st, 16);
            result = result + ch;
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        String text, key, strText;

        System.out.print("Enter 64 bit key: ");
        key = s.nextLine();
        while (!key.matches("^[a-f0-9]{16}$")) {
            System.out.print("\nInvalid key, try again: ");
            key = s.nextLine();
        }

        keygen(key); // GENERATES 16 SUBKEYS

        // PRINT SUBKEYS
```

```
        System.out.print("\n-----SUB KEYS-----");
        for (int i = 0; i < 16; i++)
            System.out.print("\nSUB KEY " + String.valueOf(i + 1) + ":
" + subkeys[i]);

        System.out.print("\n\nEnter text to encrypt: ");
        strText = s.nextLine();
        text = stringToHex(strText);
        String encrypted = encryptDecrypt(text, false);
        // System.out.println("\nHEX: " + text);
        System.out.println("ENCRYPTED TEXT: " + encrypted);

        String decrypted = encryptDecrypt(encrypted, true);
        // System.out.println("\nHEX: " + decrypted);
        System.out.println("\nDECRYPTED TEXT: " +
hexToString(decrypted));
    }
}
```

Keygen

```
import java.lang.StringBuffer;
import java.math.*;
import java.util.*;

public class Main {
    static 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 };

    static 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 };

    static int[] ip = { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36,
28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64,
            56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1,
59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29,
            21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7 };
```

```
    static int[] ip_inv = { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47,
15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30,
            37, 5, 45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60,
28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42, 10,
            50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25 };

    static int[] expansion = { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8,
9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17, 16,
            17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25, 26, 27,
28, 29, 28, 29, 30, 31, 32, 1 };

    static int[] permutation_32 = { 16, 7, 20, 21, 29, 12, 28, 17, 1,
15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27,
            3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };

    static int[][] s1 = { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12,
5, 9, 0, 7 },
            { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
            { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
            { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };

    static int[][] s2 = { { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12,
0, 5, 10 },
            { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
            { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
            { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };

    static int[][] s3 = { { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7,
11, 4, 2, 8 },
            { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
            { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
            { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };
    static int[][] s4 = { { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11,
12, 4, 15 },
            { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
            { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
            { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };
    static int[][] s5 = { { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13,
0, 14, 9 },
            { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
            { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
            { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };
```

```
    static int[][] s6 = { { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14,
7, 5, 11 },
            { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
            { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
            { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };
    static int[][] s7 = { { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5,
10, 6, 1 },
            { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
            { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
            { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };

    static int[][] s8 = { { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5,
0, 12, 7 },
            { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
            { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
            { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };

    static String[] subkeys = new String[16];

    public static String leftShift(String key, int amount) {
        for (int i = 0; i < amount; i++) {
            char first = key.charAt(0);
            key = key.substring(1, key.length()) + first;
        }
        return key;
    }

    public static String generatePC1(String key) {
        char[] array = new char[pc1.length];
        int c = 0;
        for (int i : pc1) {
            array[c] = key.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String generatePC2(String key) {
        char[] array = new char[pc2.length];
        int c = 0;
        for (int i : pc2) {
            array[c] = key.charAt(i - 1);
            c++;
```

```
        }

        return String.valueOf(array);
    }

    public static void keygen(String key) {
        String bin = new BigInteger(key, 16).toString(2); // hex to
binary
        String left, right;
        while (bin.length() != 64)
            bin = "0" + bin;
        key = generatePC1(bin);
        left = key.substring(0, key.length() / 2);
        right = key.substring(key.length() / 2, key.length());

        // GENERATE 16 SUBKEYS
        for (int i = 1; i < 17; i++) {
            if (i == 1 || i == 2 || i == 9 || i == 16) {
                // left shift once
                left = leftShift(left, 1);
                right = leftShift(right, 1);
            } else {
                // left shift twice
                left = leftShift(left, 2);
                right = leftShift(right, 2);
            }

            key = "" + left + right;
            // System.out.println(key);
            subkeys[i - 1] = generatePC2(key);
        }
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        String text, key, strText;

        System.out.print("Enter 64 bit key: ");
        key = s.nextLine();
        while (!key.matches("^[a-f0-9]{16}$")) {
            System.out.print("\nInvalid key, try again: ");
            key = s.nextLine();
        }
```

```
        keygen(key); // GENERATES 16 SUBKEYS

        // PRINT SUBKEYS
        System.out.print("\n-----SUB KEYS-----");
        for (int i = 0; i < 16; i++)
            System.out.print("\nSUB KEY " + String.valueOf(i + 1) + ":
" + subkeys[i]);
    }
}
```

Encryption

```
import java.lang.StringBuffer;
import java.math.*;
import java.util.*;

public class Main {
    static 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 };

    static 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 };

    static int[] ip = { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36,
28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64,
            56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1,
59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29,
            21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7 };

    static int[] ip_inv = { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47,
15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30,
            37, 5, 45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60,
28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42, 10,
            50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25 };

    static int[] expansion = { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8,
9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17, 16,
```

```
            17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25, 26, 27,
28, 29, 28, 29, 30, 31, 32, 1 };

    static int[] permutation_32 = { 16, 7, 20, 21, 29, 12, 28, 17, 1,
15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27,
            3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };

    static int[][] s1 = { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12,
5, 9, 0, 7 },
            { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
            { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
            { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };

    static int[][] s2 = { { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12,
0, 5, 10 },
            { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
            { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
            { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };

    static int[][] s3 = { { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7,
11, 4, 2, 8 },
            { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
            { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
            { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };
    static int[][] s4 = { { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11,
12, 4, 15 },
            { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
            { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
            { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };
    static int[][] s5 = { { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13,
0, 14, 9 },
            { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
            { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
            { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };
    static int[][] s6 = { { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14,
7, 5, 11 },
            { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
            { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
            { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };
    static int[][] s7 = { { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5,
10, 6, 1 },
            { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
            { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
```

```
            { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };

    static int[][] s8 = { { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5,
0, 12, 7 },
            { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
            { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
            { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };

    static String[] subkeys = new String[16];

    public static String leftShift(String key, int amount) {
        for (int i = 0; i < amount; i++) {
            char first = key.charAt(0);
            key = key.substring(1, key.length()) + first;
        }
        return key;
    }

    public static String generatePC1(String key) {
        char[] array = new char[pc1.length];
        int c = 0;
        for (int i : pc1) {
            array[c] = key.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String generatePC2(String key) {
        char[] array = new char[pc2.length];
        int c = 0;
        for (int i : pc2) {
            array[c] = key.charAt(i - 1);
            c++;
        }

        return String.valueOf(array);
    }

    public static void keygen(String key) {
        String bin = new BigInteger(key, 16).toString(2); // hex to
binary
        String left, right;
```

```
        while (bin.length() != 64)
            bin = "0" + bin;
        key = generatePC1(bin);
        left = key.substring(0, key.length() / 2);
        right = key.substring(key.length() / 2, key.length());

        // GENERATE 16 SUBKEYS
        for (int i = 1; i < 17; i++) {
            if (i == 1 || i == 2 || i == 9 || i == 16) {
                // left shift once
                left = leftShift(left, 1);
                right = leftShift(right, 1);
            } else {
                // left shift twice
                left = leftShift(left, 2);
                right = leftShift(right, 2);
            }

            key = "" + left + right;
            // System.out.println(key);
            subkeys[i - 1] = generatePC2(key);
        }
    }

    public static String toHex(String input) {
        String hex = "";
        for (int i = 0; i < input.length(); i++)
            hex += Integer.toHexString(input.charAt(i));
        return hex;
    }

    public static String getIP(String input) {
        char[] array = new char[ip.length];
        int c = 0;
        for (int i : ip) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);

    }

    public static String expansion(String input) {
```

```
        char[] array = new char[expansion.length];
        int c = 0;
        for (int i : expansion) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);

    }

    public static String xorStrings(String s1, String s2) {
        String output = "";
        for (int i = 0; i < s1.length(); i++) {
            if (s1.charAt(i) == s2.charAt(i))
                output += "0";
            else
                output += "1";
        }
        return output;
    }

    public static String applySbox(String input, int number) {
        String row, col, output = "";
        int r, c;
        row = "" + input.charAt(0) + input.charAt(input.length() - 1);
        col = input.substring(1, 5);
        r = Integer.parseInt(row, 2);
        c = Integer.parseInt(col, 2);
        switch (number) {
            case 0:
                output = Integer.toBinaryString(s1[r][c]);
                break;
            case 1:
                output = Integer.toBinaryString(s2[r][c]);
                break;
            case 2:
                output = Integer.toBinaryString(s3[r][c]);
                break;
            case 3:
                output = Integer.toBinaryString(s4[r][c]);
                break;
            case 4:
                output = Integer.toBinaryString(s5[r][c]);
```

```
                break;
            case 5:
                output = Integer.toBinaryString(s6[r][c]);
                break;
            case 6:
                output = Integer.toBinaryString(s7[r][c]);
                break;
            case 7:
                output = Integer.toBinaryString(s8[r][c]);
                break;
        }
        while (output.length() < 4)
            output = "0" + output;
        return output;
    }

    public static String applySboxes(String input) {
        String output = "";
        for (int i = 0; i < 8; i++) {
            output += applySbox(input.substring(i * 6, i * 6 + 6), i);
        }
        return output;
    }

    public static String getPermutation32(String input) {
        char[] array = new char[permutation_32.length];
        int c = 0;
        for (int i : permutation_32) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String getIPinverse(String input) {
        char[] array = new char[ip_inv.length];
        int c = 0;
        for (int i : ip_inv) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }
```

```
    public static String feistalRound(String right, String key) {
        right = expansion(right);
        String output = xorStrings(right, key);
        output = applySboxes(output);
        output = getPermutation32(output);
        return output;
    }

    public static String encryptDecrypt(String text, boolean decrypt) {
        String output = "", bin = "";
        String t = "" + text;
        int iter = t.length() / 16;
        if (t.length() % 16 != 0)
            iter++;

        for (int z = 0; z < iter; z++) {
            if (z + 1 < iter)
                bin = new BigInteger(t.substring(z * 16, z * 16 + 16),
16).toString(2);
            else
                bin = new BigInteger(t.substring(z * 16, t.length()),
16).toString(2);

            while (bin.length() < 64)
                bin = "0" + bin;
            bin = getIP(bin);
            String temp, left, right;
            left = bin.substring(0, bin.length() / 2);
            right = bin.substring(bin.length() / 2, bin.length());
            for (int i = 0; i < 16; i++) {
                temp = left;
                left = right;
                right = xorStrings(temp, feistalRound(right,
subkeys[i]));
            }


            text = right + left;
            text = getIPinverse(text);
            text = new BigInteger(text, 2).toString(16);

            while (text.length() < 16)
```

```
                text = "0" + text;

            output += text;
        }

        return output;
    }

    public static String stringToHex(String str) {
        StringBuffer sb = new StringBuffer();
        // Converting string to character array
        char charArray[] = str.toCharArray();
        for (int i = 0; i < charArray.length; i++) {
            String hexString = Integer.toHexString(charArray[i]);
            sb.append(hexString);
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        String text, key, strText;

        System.out.print("Enter 64 bit key: ");
        key = s.nextLine();
        while (!key.matches("^[a-f0-9]{16}$")) {
            System.out.print("\nInvalid key, try again: ");
            key = s.nextLine();
        }

        keygen(key); // GENERATES 16 SUBKEYS

        // PRINT SUBKEYS
        System.out.print("\n-----SUB KEYS-----");
        for (int i = 0; i < 16; i++)
            System.out.print("\nSUB KEY " + String.valueOf(i + 1) + ":
" + subkeys[i]);

        System.out.print("\n\nEnter text to encrypt: ");
        strText = s.nextLine();
        text = stringToHex(strText);
        String encrypted = encryptDecrypt(text, false);
        // System.out.println("\nHEX: " + text);
```

```
        System.out.println("ENCRYPTED TEXT: " + encrypted);
    }
}
```

Decryption

```
import java.lang.StringBuffer;
import java.math.*;
import java.util.*;

public class Main {
    static 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 };

    static 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 };

    static int[] ip = { 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36,
28, 20, 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64,
            56, 48, 40, 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1,
59, 51, 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29,
            21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7 };

    static int[] ip_inv = { 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47,
15, 55, 23, 63, 31, 38, 6, 46, 14, 54, 22, 62, 30,
            37, 5, 45, 13, 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60,
28, 35, 3, 43, 11, 51, 19, 59, 27, 34, 2, 42, 10,
            50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25 };

    static int[] expansion = { 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8,
9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17, 16,
            17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25, 24, 25, 26, 27,
28, 29, 28, 29, 30, 31, 32, 1 };

    static int[] permutation_32 = { 16, 7, 20, 21, 29, 12, 28, 17, 1,
15, 23, 26, 5, 18, 31, 10, 2, 8, 24, 14, 32, 27,
```

```
            3, 9, 19, 13, 30, 6, 22, 11, 4, 25 };

    static int[][] s1 = { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12,
5, 9, 0, 7 },
            { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
            { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
            { 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };

    static int[][] s2 = { { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12,
0, 5, 10 },
            { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
            { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
            { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };

    static int[][] s3 = { { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7,
11, 4, 2, 8 },
            { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
            { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
            { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };
    static int[][] s4 = { { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11,
12, 4, 15 },
            { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
            { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
            { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };
    static int[][] s5 = { { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13,
0, 14, 9 },
            { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
            { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
            { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };
    static int[][] s6 = { { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14,
7, 5, 11 },
            { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
            { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
            { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };
    static int[][] s7 = { { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5,
10, 6, 1 },
            { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
            { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
            { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };

    static int[][] s8 = { { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5,
0, 12, 7 },
            { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
```

```
            { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
            { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };

    static String[] subkeys = new String[16];

    public static String leftShift(String key, int amount) {
        for (int i = 0; i < amount; i++) {
            char first = key.charAt(0);
            key = key.substring(1, key.length()) + first;
        }
        return key;
    }

    public static String generatePC1(String key) {
        char[] array = new char[pc1.length];
        int c = 0;
        for (int i : pc1) {
            array[c] = key.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String generatePC2(String key) {
        char[] array = new char[pc2.length];
        int c = 0;
        for (int i : pc2) {
            array[c] = key.charAt(i - 1);
            c++;
        }

        return String.valueOf(array);
    }

    public static void keygen(String key) {
        String bin = new BigInteger(key, 16).toString(2); // hex to
binary
        String left, right;
        while (bin.length() != 64)
            bin = "0" + bin;
        key = generatePC1(bin);
        left = key.substring(0, key.length() / 2);
        right = key.substring(key.length() / 2, key.length());
```

```
    // GENERATE 16 SUBKEYS
    for (int i = 1; i < 17; i++) {
        if (i == 1 || i == 2 || i == 9 || i == 16) {
            // left shift once
            left = leftShift(left, 1);
            right = leftShift(right, 1);
        } else {
            // left shift twice
            left = leftShift(left, 2);
            right = leftShift(right, 2);
        }

        key = "" + left + right;
        // System.out.println(key);
        subkeys[i - 1] = generatePC2(key);
    }
}

public static String toHex(String input) {
    String hex = "";
    for (int i = 0; i < input.length(); i++)
        hex += Integer.toHexString(input.charAt(i));
    return hex;
}

public static String getIP(String input) {
    char[] array = new char[ip.length];
    int c = 0;
    for (int i : ip) {
        array[c] = input.charAt(i - 1);
        c++;
    }
    return String.valueOf(array);

}

public static String expansion(String input) {
    char[] array = new char[expansion.length];
    int c = 0;
    for (int i : expansion) {
        array[c] = input.charAt(i - 1);
        c++;
```

```
        }
        return String.valueOf(array);

    }

    public static String xorStrings(String s1, String s2) {
        String output = "";
        for (int i = 0; i < s1.length(); i++) {
            if (s1.charAt(i) == s2.charAt(i))
                output += "0";
            else
                output += "1";
        }
        return output;
    }

    public static String applySbox(String input, int number) {
        String row, col, output = "";
        int r, c;
        row = "" + input.charAt(0) + input.charAt(input.length() - 1);
        col = input.substring(1, 5);
        r = Integer.parseInt(row, 2);
        c = Integer.parseInt(col, 2);
        switch (number) {
            case 0:
                output = Integer.toBinaryString(s1[r][c]);
                break;
            case 1:
                output = Integer.toBinaryString(s2[r][c]);
                break;
            case 2:
                output = Integer.toBinaryString(s3[r][c]);
                break;
            case 3:
                output = Integer.toBinaryString(s4[r][c]);
                break;
            case 4:
                output = Integer.toBinaryString(s5[r][c]);
                break;
            case 5:
                output = Integer.toBinaryString(s6[r][c]);
                break;
            case 6:
```

```
                output = Integer.toBinaryString(s7[r][c]);
                break;
            case 7:
                output = Integer.toBinaryString(s8[r][c]);
                break;
        }
        while (output.length() < 4)
            output = "0" + output;
        return output;
    }

    public static String applySboxes(String input) {
        String output = "";
        for (int i = 0; i < 8; i++) {
            output += applySbox(input.substring(i * 6, i * 6 + 6), i);
        }
        return output;
    }

    public static String getPermutation32(String input) {
        char[] array = new char[permutation_32.length];
        int c = 0;
        for (int i : permutation_32) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String getIPinverse(String input) {
        char[] array = new char[ip_inv.length];
        int c = 0;
        for (int i : ip_inv) {
            array[c] = input.charAt(i - 1);
            c++;
        }
        return String.valueOf(array);
    }

    public static String feistalRound(String right, String key) {
        right = expansion(right);
        String output = xorStrings(right, key);
        output = applySboxes(output);
```

```
        output = getPermutation32(output);
        return output;
    }

    public static String encryptDecrypt(String text, boolean decrypt) {
        String output = "", bin = "";
        String t = "" + text;
        int iter = t.length() / 16;
        if (t.length() % 16 != 0)
            iter++;

        for (int z = 0; z < iter; z++) {
            if (z + 1 < iter)
                bin = new BigInteger(t.substring(z * 16, z * 16 + 16),
16).toString(2);
            else
                bin = new BigInteger(t.substring(z * 16, t.length()),
16).toString(2);

            while (bin.length() < 64)
                bin = "0" + bin;
            bin = getIP(bin);
            String temp, left, right;
            left = bin.substring(0, bin.length() / 2);
            right = bin.substring(bin.length() / 2, bin.length());


            for (int i = 0; i < 16; i++) {
                temp = left;
                left = right;
                right = xorStrings(temp, feistalRound(right, subkeys[15
- i]));
            }

            text = right + left;
            text = getIPinverse(text);
            text = new BigInteger(text, 2).toString(16);

            while (text.length() < 16)
                text = "0" + text;

            output += text;
        }
```

```
        return output;
    }

    public static String hexToString(String str) {
        String result = new String();
        char[] charArray = str.toCharArray();
        for (int i = 0; i < charArray.length; i = i + 2) {
            String st = "" + charArray[i] + "" + charArray[i + 1];
            char ch = (char) Integer.parseInt(st, 16);
            result = result + ch;
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        String text, key, strText;

        System.out.print("Enter 64 bit key: ");
        key = s.nextLine();
        while (!key.matches("^[a-f0-9]{16}$")) {
            System.out.print("\nInvalid key, try again: ");
            key = s.nextLine();
        }

        keygen(key); // GENERATES 16 SUBKEYS

        // PRINT SUBKEYS
        System.out.print("\n-----SUB KEYS-----");
        for (int i = 0; i < 16; i++)
            System.out.print("\nSUB KEY " + String.valueOf(i + 1) + ":
" + subkeys[i]);

        System.out.print("\n\nEnter text to decrypt: ");
        String encrypted = s.nextLine();

        String decrypted = encryptDecrypt(encrypted, true);
        // System.out.println("\nHEX: " + decrypted);
        System.out.println("\nDECRYPTED TEXT: " +
hexToString(decrypted));
    }
}
```

CRT

```
import java.io.*;
import java.util.*;

class Main{

    static int inv(int a, int m)
    {
        int m0 = m, t, q;
        int x0 = 0, x1 = 1;

        if (m == 1)
        return 0;

        // Apply extended Euclid Algorithm
        while (a > 1)
        {
            // q is quotient
            q = a / m;

            t = m;

            // m is remainder now, process
            // same as euclid's algo
            m = a % m;a = t;

            t = x0;

            x0 = x1 - q * x0;

            x1 = t;
        }

        // Make x1 positive
        if (x1 < 0)
        x1 += m0;

        return x1;
    }

    // coprime (gcd for every pair is 1)
```

```
    static int findMinX(int num[], int rem[], int k)
    {
        // Compute product of all numbers
        int prod = 1;
        for (int i = 0; i < k; i++)
            prod *= num[i];

        // Initialize result
        int result = 0;

        // Apply above formula
        for (int i = 0; i < k; i++)
        {
            int pp = prod / num[i];
            result += rem[i] * inv(pp, num[i]) * pp;
        }

        return result % prod;
    }

    public static void main(String args[])
    {
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter number of remainders");
    int n=sc.nextInt();
        int num[] = new int[n];
        int rem[] = new int[n];
    System.out.println("Enter the relatively prime numbers");
    for(int i=0;i<n;i++)
      num[i]=sc.nextInt();
    System.out.println("Enter the remainders");
    for(int i=0;i<n;i++)
      rem[i]=sc.nextInt();
        System.out.println("x is " +findMinX(num, rem, n));
    }
}
```

DHKE

```
import java.math.BigInteger;
import java.util.*;
public class Main {
    public static void main(String[] args) {
```

```
        Scanner in = new Scanner(System.in);
        int P = BigInteger.probablePrime(30, new Random()).intValue();
        int G = primitiveRoot(P);
        System.out.println("P = "+P+" G = "+G);
        BigInteger g = new BigInteger(""+G);
        BigInteger p = new BigInteger(""+P);

        System.out.println("Enter A's secret key ");
        BigInteger xa = new BigInteger(in.next());
        BigInteger ya = g.modPow(xa, p);
        System.out.println("A's public key = " + ya);

        System.out.println("Enter B's secret key ");
        BigInteger xb = new BigInteger(in.next());
        BigInteger yb = g.modPow(xb, p);
        System.out.println("B's public key = " + yb);

        BigInteger A_sh = yb.modPow(xa, p);
        BigInteger B_sh = ya.modPow(xb, p);

        System.out.println("A's shared secret = "+A_sh);
        System.out.println("B's shared secret = "+B_sh);


        in.close();

    }

    public static int primitiveRoot(int p)
    {
        if (!(new BigInteger(""+p).isProbablePrime(1))) return -1;

        int phi = p-1;
        ArrayList<Integer> fact = primeFactors(phi);
        for (int res=2; res<=p; ++res) {
            Boolean ok = true;
            for (int i=0; i<fact.size() && ok; ++i)
                ok &= powmod (res, phi / fact.get(i), p) != 1;
            if (ok)  return res;
        }
        return -1;
    }
```

```
    public static ArrayList<Integer> primeFactors(int n)
    {
        ArrayList<Integer> facts = new ArrayList<>();
        for (int i=2; i*i<=n; ++i)
            if (n % i == 0) {
                facts.add(i);
                while (n % i == 0)
                    n /= i;
            }

        return facts;

    }
    public static int powmod(int a,int b,int mod)
    {
        if (b==0)
            return 1;
        if (b==1)
            return a%mod;
        int temp = powmod(a,b/2,mod);
        temp = (temp*temp);
        if (b%2!=0)
            temp*= a;
        return temp%mod;
    }
}
```

DHKE_miller

```
import java.math.BigInteger;
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int P = BigInteger.probablePrime(15, new Random()).intValue();
        int G = findPrimitive(P);
        if (isPrime(P, 1))
            System.out.println("P = " + P + "is prime");
        else {
            System.out.println("P = " + P + " is composite");
            while (!isPrime(P, 1)) {
                P = BigInteger.probablePrime(15, new
Random()).intValue();
```

```
            }
        }

        System.out.println("G = " + G);
        BigInteger g = new BigInteger("" + G);
        BigInteger p = new BigInteger("" + P);
        System.out.println("Enter A's secret key ");
        BigInteger xa = new BigInteger( in .next());
        BigInteger ya = g.modPow(xa, p);
        System.out.println("A's public key = " + ya);
        System.out.println("Enter B's secret key ");
        BigInteger xb = new BigInteger( in .next());
        BigInteger yb = g.modPow(xb, p);
        System.out.println("B's public key = " + yb);
        BigInteger A_sh = yb.modPow(xa, p);
        BigInteger B_sh = ya.modPow(xb, p);
        System.out.println();
        System.out.println("A's shared secret = " + A_sh);
        System.out.println("B's shared secret = " + B_sh); in .close();
    }
    static boolean isPrime(int n) {
        if (n <= 1)
            return false;
        if (n <= 3)
            return true;
        if (n % 2 == 0 || n % 3 == 0)
            return false;
        for (int i = 5; i * i <= n; i = i + 6)
            if (n % i == 0 || n % (i + 2) == 0)
                return false;
        return true;
    }
    static int power(int x, int y, int p) {
        int res = 1;
        while (y > 0) {
            if (y % 2 == 1)
                res = (res * x) % p;
            y = y >> 1;
            x = (x * x) % p;
        }
        return res;
    }
    static void findPrimefactors(HashSet < Integer > s, int n) {
```

```
    while (n % 2 == 0) {
        s.add(2);
        n = n / 2;
    }
    for (int i = 3; i <= Math.sqrt(n); i = i + 2) {
        while (n % i == 0) {
            s.add(i);
            n = n / i;
        }
    }
    if (n > 2)
        s.add(n);
}
static int findPrimitive(int n) {
    HashSet < Integer > s = new HashSet < Integer > ();
    if (isPrime(n) == false)
        return -1;
    int phi = n - 1;
    findPrimefactors(s, phi);
    for (int r = 2; r <= phi; r++) {
        boolean flag = false;
        for (Integer a: s) {
            if (power(r, phi / (a), n) == 1) {
                flag = true;
                break;
            }
        }
        if (flag == false)
            return r;
    }
    return -1;
}
static boolean miillerTest(int d, int n) {
    int a = 2 + (int)(Math.random() % (n - 4));
    int x = power(a, d, n);
    if (x == 1 || x == n - 1)
        return true;
    while (d != n - 1) {
        x = (x * x) % n;
        d *= 2;
        if (x == 1)
            return false;
        if (x == n - 1)
```

```
                return true;
        }
        return false;
    }
    static boolean isPrime(int n, int k) {
        if (n <= 1 || n == 4)
            return false;
        if (n <= 3)
            return true;
        int d = n - 1;
        while (d % 2 == 0)
            d /= 2;
        for (int i = 0; i < k; i++)
            if (!miillerTest(d, n))
                return false;
        return true;
    }
}
```

DSS

```
import java.io.UnsupportedEncodingException;
import java.util.*;
import java.security.*;

public class Main {
    public static void main(String[] args)
            throws NoSuchAlgorithmException, InvalidKeyException,
SignatureException, UnsupportedEncodingException {

        Scanner sc = new Scanner(System.in);
        System.out.print("Text : ");
        String msg = sc.nextLine();

        KeyPairGenerator keys =
KeyPairGenerator.getInstance("DSA");//accepts a String variable
representing the required key-generating algorithm and returns a
KeyPairGenerator object that generates keys.
        keys.initialize(1024);//keysize

        KeyPair key = keys.generateKeyPair();
        PrivateKey pk = key.getPrivate();
        PublicKey puk = key.getPublic();
```

```
        Signature sign = Signature.getInstance("SHA256withDSA");//he
String passed as parameter to the getInstance() method is the name of
the digital signature algorithm to use.
        sign.initSign(pk); //Initializing the signature

        byte[] text = msg.getBytes();
        sign.update(text);//adding data to signature
        byte[] signature = sign.sign();//calculating the signature
        sign.initVerify(puk);//initializing the signature

        System.out.print("Data : ");
        String data = sc.nextLine();
        sign.update(data.getBytes());

        if (sign.verify(signature))//verifying
            System.out.println("Signature Verified !");
        else
            System.out.println("Signature Invalid !");
    }
}
```

Hill Cipher

```
import java.io.*;
public class Main {
    int key[][] = new int[3][3];
    int inversemat[][] = new int[3][3];
    public int getkeyrix(String ck) {
        try {
            int det;
            if (ck.length() != 9)
                throw new Exception("INVALID: matrix of size 3 cannot
be formed");
            ck = ck.toUpperCase();
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {
                    if (Character.isLetter(ck.charAt(i * 3 + j)) ==
false)
                        throw new Exception("Invalid key characters");
                    key[i][j] = ck.charAt(i * 3 + j) % 65;
                }
            }
```

```
            det = key[0][0] * ((key[1][1] * key[2][2]) - (key[2][1] *
key[1][2])) - key[0][1] * (key[1][0] *
                key[2][2] - key[2][0] * key[1][2]) + key[0][2] *
(key[1][0] * key[2][1] - key[2][0] *
                key[1][1]);
            det = (det < 0) ? (26 - (Math.abs(det) % 26)) % 26 : (det %
26);
            if (det == 0)
                throw new Exception("Invalid:Non-invertible Matrix");
            int detinv = 0;
            for (int i = 1; i <= 25; i++)
                if ((i * det) % 26 == 1) {
                    detinv = i;
                    break;
                }
            if (detinv == 0)
                throw new Exception("Invalid key, no inverse for
det.Decryption Not Possible");
            for (int i = 0; i < 3; i++)
                for (int j = 0; j < 3; j++) {
                    inversemat[i][j] = ((key[(j + 1) % 3][(i + 1) % 3]
* key[(j + 2) % 3][(i + 2) % 3]) -
                            (key[(j + 1) % 3][(i + 2) % 3] * key[(j + 2) %
3][(i + 1) % 3]));
                    inversemat[i][j] = (inversemat[i][j] < 0) ? (26 -
(Math.abs(inversemat[i][j]) % 26)) % 26 :
                            (inversemat[i][j] % 26);
                    inversemat[i][j] = (detinv * inversemat[i][j]) %
26;
                }
            System.out.println("Key matrix: ");
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++)
                    System.out.print(key[i][j] + " ");
                System.out.println();
            }

            System.out.println("Inverse of Key Matrix: ");
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++)
                    System.out.print(inversemat[i][j] + " ");
                System.out.println();
            }
```

```
            return 1;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        return 0;
    }
    public StringBuilder encrypt(String text) {
        text = text.toUpperCase();
        StringBuilder ciptex = new StringBuilder();
        int x, y, z;
        int cipmat[][] = new int[3][1];
        int mesvec[][] = new int[3][1];
        try {
            if (text.length() % 3 == 1)
                text = text + "XX";
            else if (text.length() % 3 == 2)
                text = text + "X";
            for (int i = 0; i < text.length(); i = i + 3) {
                if (Character.isLetter(text.charAt(i)) == false ||
Character.isLetter(text.charAt(i + 1)) == false ||
Character.isLetter(text.charAt(i + 1)) == false)
                    throw new Exception("Invalid Message");
                for (int p = 0; p < 3; p++)
                    mesvec[p][0] = (text.charAt(i + p)) % 65;
                for (x = 0; x < 3; x++) {
                    for (y = 0; y < 1; y++) {
                        cipmat[x][y] = 0;
                        for (z = 0; z < 3; z++) {
                            cipmat[x][y] +=
                                key[x][z] * mesvec[z][y];
                        }
                        cipmat[x][y] = cipmat[x][y] % 26;
                    }
                }
                char n;
                for (int p = 0; p < 3; p++) {
                    n = (char)(cipmat[p][0] + 65);
                    ciptex.append(n);

                }
            }
            return ciptex;
        } catch (Exception ex) {
```

```
            System.out.println(ex.getMessage());
        }
        return null;
    }
    public StringBuilder decrypt(String text) {
        text = text.toUpperCase();
        StringBuilder orgtex = new StringBuilder();
        int x, y, z;
        int cipmat[][] = new int[3][1];
        int mesvec[][] = new int[3][1];
        try {
            if (text.length() % 3 != 0)
                throw new Exception("Invalid Cipher text length ");
            for (int i = 0; i < text.length(); i = i + 3) {
                if (Character.isLetter(text.charAt(i)) == false ||
Character.isLetter(text.charAt(i + 1)) == false ||
Character.isLetter(text.charAt(i + 1)) == false)
                    throw new Exception("Invalid Cipher text");
                for (int p = 0; p < 3; p++)
                    mesvec[p][0] = (text.charAt(i + p)) % 65;
                for (x = 0; x < 3; x++) {
                    for (y = 0; y < 1; y++) {
                        cipmat[x][y] = 0;
                        for (z = 0; z < 3; z++) {
                            cipmat[x][y] +=
                                inversemat[x][z] * mesvec[z][y];
                        }
                        cipmat[x][y] = cipmat[x][y] % 26;
                    }
                }
                char n;
                for (int p = 0; p < 3; p++) {
                    n = (char)(cipmat[p][0] + 65);
                    orgtex.append(n);
                }
            }
            return orgtex;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

        return null;
    }
```

```
    public static void main(String[] args) throws IOException {
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ir);
        System.out.println("Enter cipher key : ");
        String ck = br.readLine();
        Main hill = new Main();
        String text;
        if (hill.getkeyrix(ck) == 1) {
            System.out.println("Enter plaintext to encrypt: ");
            text = br.readLine();
            String ciptex = hill.encrypt(text.replace(" ",
"")).toString();
            System.out.println(ciptex);
            System.out.println("Decrypting the cipher text");
            System.out.println(hill.decrypt(ciptex));
        }
    }
}
```

Encryption

```
import java.io.*;
public class Main {
    int key[][] = new int[3][3];
    int inversemat[][] = new int[3][3];
    public int getkeyrix(String ck) {
        try {
            int det;
            if (ck.length() != 9)
                throw new Exception("INVALID: matrix of size 3 cannot
be formed");
            ck = ck.toUpperCase();
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {
                    if (Character.isLetter(ck.charAt(i * 3 + j)) ==
false)
                        throw new Exception("Invalid key characters");
                    key[i][j] = ck.charAt(i * 3 + j) % 65;
                }
            }
            System.out.println("Key matrix: ");
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++)
```

```
                    System.out.print(key[i][j] + " ");
                System.out.println();
            }
            return 1;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        return 0;
    }
    public StringBuilder encrypt(String text) {
        text = text.toUpperCase();
        StringBuilder ciptex = new StringBuilder();
        int x, y, z;
        int cipmat[][] = new int[3][1];
        int mesvec[][] = new int[3][1];
        try {
            if (text.length() % 3 == 1)
                text = text + "XX";
            else if (text.length() % 3 == 2)
                text = text + "X";
            for (int i = 0; i < text.length(); i = i + 3) {
                if (Character.isLetter(text.charAt(i)) == false ||
Character.isLetter(text.charAt(i + 1)) == false ||
Character.isLetter(text.charAt(i + 1)) == false)
                    throw new Exception("Invalid Message");
                for (int p = 0; p < 3; p++)
                    mesvec[p][0] = (text.charAt(i + p)) % 65;
                for (x = 0; x < 3; x++) {
                    for (y = 0; y < 1; y++) {
                        cipmat[x][y] = 0;
                        for (z = 0; z < 3; z++) {
                            cipmat[x][y] +=
                                key[x][z] * mesvec[z][y];
                        }
                        cipmat[x][y] = cipmat[x][y] % 26;
                    }
                }
                char n;
                for (int p = 0; p < 3; p++) {
                    n = (char)(cipmat[p][0] + 65);
                    ciptex.append(n);

                }
```

```
            }
            return ciptex;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        return null;
    }

    public static void main(String[] args) throws IOException {
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ir);
        System.out.println("Enter cipher key : ");
        String ck = br.readLine();
        Main hill = new Main();
        String text;
        if (hill.getkeyrix(ck) == 1) {
            System.out.println("Enter plaintext to encrypt: ");
            text = br.readLine();
            String ciptex = hill.encrypt(text.replace(" ",
"")).toString();
            System.out.println("Encrypted Text:");
            System.out.println(ciptex);
        }
    }
}
```

Decryption

```
import java.io.*;
public class Main {
    int key[][] = new int[3][3];
    int inversemat[][] = new int[3][3];
    public int getkeyrix(String ck) {
        try {
            int det;
            if (ck.length() != 9)
                throw new Exception("INVALID: matrix of size 3 cannot
be formed");
            ck = ck.toUpperCase();
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++) {
                    if (Character.isLetter(ck.charAt(i * 3 + j)) ==
false)
                        throw new Exception("Invalid key characters");
```

```
                key[i][j] = ck.charAt(i * 3 + j) % 65;
            }
        }
        det = key[0][0] * ((key[1][1] * key[2][2]) - (key[2][1] *
key[1][2])) - key[0][1] * (key[1][0] *
            key[2][2] - key[2][0] * key[1][2]) + key[0][2] *
(key[1][0] * key[2][1] - key[2][0] *
            key[1][1]);
        det = (det < 0) ? (26 - (Math.abs(det) % 26)) % 26 : (det %
26);
        if (det == 0)
            throw new Exception("Invalid:Non-invertible Matrix");
        int detinv = 0;
        for (int i = 1; i <= 25; i++)
            if ((i * det) % 26 == 1) {
                detinv = i;
                break;
            }
        if (detinv == 0)
            throw new Exception("Invalid key, no inverse for
det.Decryption Not Possible");
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++) {
                inversemat[i][j] = ((key[(j + 1) % 3][(i + 1) % 3]
* key[(j + 2) % 3][(i + 2) % 3]) -
                        (key[(j + 1) % 3][(i + 2) % 3] * key[(j + 2) %
3][(i + 1) % 3]));
                inversemat[i][j] = (inversemat[i][j] < 0) ? (26 -
(Math.abs(inversemat[i][j]) % 26)) % 26 :
                        (inversemat[i][j] % 26);
                inversemat[i][j] = (detinv * inversemat[i][j]) %
26;
            }
        System.out.println("Key matrix: ");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
                System.out.print(key[i][j] + " ");
            System.out.println();
        }

        System.out.println("Inverse of Key Matrix: ");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
```

```
                    System.out.print(inversemat[i][j] + " ");
                System.out.println();
            }
            return 1;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        return 0;
    }

    public StringBuilder decrypt(String text) {
        text = text.toUpperCase();
        StringBuilder orgtex = new StringBuilder();
        int x, y, z;
        int cipmat[][] = new int[3][1];
        int mesvec[][] = new int[3][1];
        try {
            if (text.length() % 3 != 0)
                throw new Exception("Invalid Cipher text length ");
            for (int i = 0; i < text.length(); i = i + 3) {
                if (Character.isLetter(text.charAt(i)) == false ||
Character.isLetter(text.charAt(i + 1)) == false ||
Character.isLetter(text.charAt(i + 1)) == false)
                    throw new Exception("Invalid Cipher text");
                for (int p = 0; p < 3; p++)
                    mesvec[p][0] = (text.charAt(i + p)) % 65;
                for (x = 0; x < 3; x++) {
                    for (y = 0; y < 1; y++) {
                        cipmat[x][y] = 0;
                        for (z = 0; z < 3; z++) {
                            cipmat[x][y] +=
                                inversemat[x][z] * mesvec[z][y];
                        }
                        cipmat[x][y] = cipmat[x][y] % 26;
                    }
                }
                char n;
                for (int p = 0; p < 3; p++) {
                    n = (char)(cipmat[p][0] + 65);
                    orgtex.append(n);
                }
            }
            return orgtex;
```

```
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

        return null;
    }
    public static void main(String[] args) throws IOException {
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ir);
        System.out.println("Enter cipher key : ");
        String ck = br.readLine();
        Main hill = new Main();
        String text;
        if (hill.getkeyrix(ck) == 1) {
            System.out.println("Enter Encrytped Text");
            String ciptex = br.readLine();
            System.out.println("Decrypting the cipher text");
            System.out.println(hill.decrypt(ciptex));
        }
    }
}
```

Hill Number

```
import java.io.*;
public class Main {
    public static int key[][] = new int[3][3];
    int inversemat[][] = new int[3][3];
    public int getkeyrix() {
        try {
            int det;
            det = key[0][0] * ((key[1][1] * key[2][2]) - (key[2][1] *
key[1][2])) - key[0][1] * (key[1][0] *
                key[2][2] - key[2][0] * key[1][2]) + key[0][2] *
(key[1][0] * key[2][1] - key[2][0] *
                key[1][1]);
            det = (det < 0) ? (26 - (Math.abs(det) % 26)) % 26 : (det %
26);
            if (det == 0)
                throw new Exception("Invalid:Non-invertible Matrix");
            int detinv = 0;
            for (int i = 1; i <= 25; i++)
                if ((i * det) % 26 == 1) {
                    detinv = i;
```

```
                    break;
                }
            if (detinv == 0)
                throw new Exception("Invalid key, no inverse for
det.Decryption Not Possible");
            for (int i = 0; i < 3; i++)
                for (int j = 0; j < 3; j++) {
                    inversemat[i][j] = ((key[(j + 1) % 3][(i + 1) % 3]
* key[(j + 2) % 3][(i + 2) % 3]) -
                            (key[(j + 1) % 3][(i + 2) % 3] * key[(j + 2) %
3][(i + 1) % 3]));
                    inversemat[i][j] = (inversemat[i][j] < 0) ? (26 -
(Math.abs(inversemat[i][j]) % 26)) % 26 :
                            (inversemat[i][j] % 26);
                    inversemat[i][j] = (detinv * inversemat[i][j]) %
26;
                }
            System.out.println("Key matrix: ");
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++)
                    System.out.print(key[i][j] + " ");
                System.out.println();
            }

            System.out.println("Inverse of Key Matrix: ");
            for (int i = 0; i < 3; i++) {
                for (int j = 0; j < 3; j++)
                    System.out.print(inversemat[i][j] + " ");
                System.out.println();
            }
            return 1;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        return 0;
    }
    public StringBuilder encrypt(String text) {
        text = text.toUpperCase();
        StringBuilder ciptex = new StringBuilder();
        int x, y, z;
        int cipmat[][] = new int[3][1];
        int mesvec[][] = new int[3][1];
        try {
```

```
            if (text.length() % 3 == 1)
                text = text + "XX";
            else if (text.length() % 3 == 2)
                text = text + "X";
            for (int i = 0; i < text.length(); i = i + 3) {
                if (Character.isLetter(text.charAt(i)) == false ||
Character.isLetter(text.charAt(i + 1)) == false ||
Character.isLetter(text.charAt(i + 1)) == false)
                    throw new Exception("Invalid Message");
                for (int p = 0; p < 3; p++)
                    mesvec[p][0] = (text.charAt(i + p)) % 65;
                for (x = 0; x < 3; x++) {
                    for (y = 0; y < 1; y++) {
                        cipmat[x][y] = 0;
                        for (z = 0; z < 3; z++) {
                            cipmat[x][y] +=
                                key[x][z] * mesvec[z][y];
                        }
                        cipmat[x][y] = cipmat[x][y] % 26;
                    }
                }
                char n;
                for (int p = 0; p < 3; p++) {
                    n = (char)(cipmat[p][0] + 65);
                    ciptex.append(n);

                }
            }
            return ciptex;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
        return null;
    }
    public StringBuilder decrypt(String text) {
        text = text.toUpperCase();
        StringBuilder orgtex = new StringBuilder();
        int x, y, z;
        int cipmat[][] = new int[3][1];
        int mesvec[][] = new int[3][1];
        try {
            if (text.length() % 3 != 0)
                throw new Exception("Invalid Cipher text length ");
```

```
            for (int i = 0; i < text.length(); i = i + 3) {
                if (Character.isLetter(text.charAt(i)) == false ||
Character.isLetter(text.charAt(i + 1)) == false ||
Character.isLetter(text.charAt(i + 1)) == false)
                    throw new Exception("Invalid Cipher text");
                for (int p = 0; p < 3; p++)
                    mesvec[p][0] = (text.charAt(i + p)) % 65;
                for (x = 0; x < 3; x++) {
                    for (y = 0; y < 1; y++) {
                        cipmat[x][y] = 0;
                        for (z = 0; z < 3; z++) {
                            cipmat[x][y] +=
                                inversemat[x][z] * mesvec[z][y];
                        }
                        cipmat[x][y] = cipmat[x][y] % 26;
                    }
                }
                char n;
                for (int p = 0; p < 3; p++) {
                    n = (char)(cipmat[p][0] + 65);
                    orgtex.append(n);
                }
            }
            return orgtex;
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }

        return null;
    }
    public static void main(String[] args) throws IOException {
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ir);
        Main hill = new Main();
        String text;
        System.out.println("Enter size : ");
        int n=Integer.parseInt(br.readLine());
        System.out.println("Enter matrix");
            for(int i=0;i<n;i++)
              for(int j=0;j<n;j++)
                key[i][j]=Integer.parseInt(br.readLine());
        hill.getkeyrix();
            System.out.println("Enter plaintext to encrypt: ");
```

```
            text = br.readLine();
            String ciptex = hill.encrypt(text.replace(" ",
"")).toString();
            System.out.println(ciptex);
            System.out.println("Decrypting the cipher text");
            System.out.println(hill.decrypt(ciptex));
        }
    }
```

Inverse matrix:

```
import java.io.*;
public class Main {
   public static int key[][] = new int[3][3];
   int inversemat[][] = new int[3][3];
   public int getkeyrix() {
      try {
         int det;
         det = key[0][0] * ((key[1][1] * key[2][2]) - (key[2][1] * key[1][2])) - key[0][1] * (key[1][0] *
            key[2][2] - key[2][0] * key[1][2]) + key[0][2] * (key[1][0] * key[2][1] - key[2][0] *
            key[1][1]);
         det = (det < 0) ? (26 - (Math.abs(det) % 26)) % 26 : (det % 26);
         if (det == 0)
            throw new Exception("Invalid:Non-invertible Matrix");
         int detinv = 0;
         for (int i = 1; i <= 25; i++)
            if ((i * det) % 26 == 1) {
               detinv = i;
               break;
            }
         if (detinv == 0)
            throw new Exception("Invalid key, no inverse for det.Decryption Not Possible");
         for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++) {
               inversemat[i][j] = ((key[(j + 1) % 3][(i + 1) % 3] * key[(j + 2) % 3][(i + 2) % 3]) -
                  (key[(j + 1) % 3][(i + 2) % 3] * key[(j + 2) % 3][(i + 1) % 3]));
               inversemat[i][j] = (inversemat[i][j] < 0) ? (26 - (Math.abs(inversemat[i][j]) % 26))
% 26 :
                  (inversemat[i][j] % 26);
               inversemat[i][j] = (detinv * inversemat[i][j]) % 26;
            }
         System.out.println("Key matrix: ");
         for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
               System.out.print(key[i][j] + " ");
            System.out.println();
```

```
        }

        System.out.println("Inverse of Key Matrix: ");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
                System.out.print(inversemat[i][j] + " ");
            System.out.println();
        }
        return 1;
      } catch (Exception ex) {
        System.out.println(ex.getMessage());
      }
      return 0;
    }


    public static void main(String[] args) throws IOException {
      InputStreamReader ir = new InputStreamReader(System.in);
      BufferedReader br = new BufferedReader(ir);
      System.out.println("Enter size : ");
      int n=Integer.parseInt(br.readLine());
      Main hill = new Main();

      System.out.println("Enter matrix");
        for(int i=0;i<n;i++)
          for(int j=0;j<n;j++)
            key[i][j]=Integer.parseInt(br.readLine());
      hill.getkeyrix();
    }
}
```

MD5

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

// Java program to calculate MD5 hash value
public class Main {
    public static String getMd5(String input)
    {
        try {

            // Static getInstance method is called with hashing MD5
            MessageDigest md = MessageDigest.getInstance("MD5");
```

```
            // digest() method is called to calculate message digest
            // of an input digest() return array of byte
            byte[] messageDigest = md.digest(input.getBytes());

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

            // Convert message digest into hex value
            String hashtext = no.toString(16);
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            return hashtext;
        }

        // For specifying wrong message digest algorithms
        catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }

    // Driver code
    public static void main(String args[]) throws
NoSuchAlgorithmException
    {
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter String");
        String s =sc.nextLine() ;
        System.out.println("Your HashCode Generated by MD5 is: " +
getMd5(s));
    }
}
```

MillerRabin

```
import java.io.*;
import java.math.*;
import java.util.Scanner;

class Main {
    static int power(int x, int y, int p) {

        int res = 1;
        x = x % p;
```

```
    while (y > 0) {

        // If y is odd, multiply x with result
        if ((y & 1) == 1)
            res = (res * x) % p;

        // y must be even now
        y = y >> 1; // y = y/2
        x = (x * x) % p;
    }

    return res;
}

static boolean miillerTest(int d, int n) {

    int a = 2 + (int) (Math.random() % (n - 4));

    int x = power(a, d, n);

    if (x == 1 || x == n - 1)
        return true;

    while (d != n - 1) {
        x = (x * x) % n;
        d *= 2;

        if (x == 1)
            return false;
        if (x == n - 1)
            return true;
    }

    return false;
}

static boolean isPrime(int n, int k) {

    // Corner cases
    if (n <= 1 || n == 4)
        return false;
    if (n <= 3)
```

```
            return true;

        int d = n - 1;

        while (d % 2 == 0)
            d /= 2;

        for (int i = 0; i < k; i++)
            if (!miillerTest(d, n))
                return false;

        return true;
    }

    public static void main(String args[]) {

        int k = 4;
        int n;
        Scanner s = new Scanner(System.in);

        System.out.print("Enter n value: ");
        n = s.nextInt();
        // System.out.println(n + " is prime? " + isPrime(n, k));
        if(isPrime(n,k))
          System.out.println(n+" is a prime number");
        else
          System.out.println(n+" is not a prime number");
    }
}
```

Modular

```
import java.io.*;
import java.util.*;
class Main {

    // A naive method to find modulor
    // multiplicative inverse of 'a'
    // under modulo 'm'
    static int modInverse(int a, int m)
    {
        a = a % m;
        for (int x = 1; x < m; x++)
```

```
            if ((a * x) % m == 1)
                return x;
        return 1;
    }

    // Driver Code
    public static void main(String args[])
    {
      Scanner sc=new Scanner(System.in);
      System.out.println("Enter number whose inverse is to be found");
      int a=sc.nextInt();
      System.out.println("Enter number ");
      int m=sc.nextInt();
        System.out.println("Inverse is "+modInverse(a, m));
    }
}
```

Playfair

```
import java.util.Scanner;
import java.lang.String.*;
import java.io.*;

public class Main {
    private String KeyWord = new String();
    private String Key = new String();
    private char matrix_arr[][] = new char[5][5];

    public static void main(String[] args)throws IOException {
        InputStreamReader ir=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(ir);
        String keyword = "", plainText = "",cipher="";
        int c;
        Main object = new Main();
        do {
            System.out.println("Playfair Cipher");
            System.out.println("Enter the choice");
            System.out.println("Menu");
            System.out.println("1. Read input");
            System.out.println("2. Read Key");
            System.out.println("3. Matrix");
            System.out.println("4. Encryption");
            System.out.println("5. Decryption");
            int ch = Integer.parseInt(br.readLine());
```

```
            switch (ch) {
                case 1:
                    System.out.println("Enter a Text : ");
                    plainText = br.readLine();
                    break;
                case 2:
                    System.out.println("Enter key");
                    keyword = br.readLine();
                    break;
                case 3:
                    object.remove_duplicates(keyword);
                    object.generate_key();
                    if (plainText.replace(" ","").length() % 2 != 0) {
                        plainText = plainText + "x";
                    }
                    break;
                case 4:
                    System.out.println("Encryption : " +
object.encrypt(plainText.replace(" ",""),plainText));
                    cipher = object.encrypt(plainText.replace("
",""),plainText);
                    break;
                case 5:
                    System.out.println("Decryption : " +
object.decrypt(cipher.replace(" ",""),plainText));
                    break;
            }
            System.out.println("Enter 1 to continue");
            c = Integer.parseInt(br.readLine());
        } while (c == 1);
    }

    public void remove_duplicates(String k) {
        String new_key = new String();
        boolean flag = false;
        new_key = new_key + k.charAt(0);
        for (int i = 1; i < k.length(); i++) {
            for (int j = 0; j < new_key.length(); j++) {
                if (k.charAt(i) == new_key.charAt(j)) {
                    flag = true;
                }
            }
            if (flag == false)
```

```
            new_key = new_key + k.charAt(i);
        flag = false;
    }
    KeyWord = new_key;
}

public void generate_key() {
    boolean flag = true;
    char curr;
    Key = KeyWord;
    for (int i = 0; i < 26; i++) {
        curr = (char) (i + 97);
        if (curr == 'j')
            continue;
        for (int j = 0; j < KeyWord.length(); j++) {
            if (curr == KeyWord.charAt(j)) {
                flag = false;
                break;
            }
        }
        if (flag)
            Key = Key + curr;
        flag = true;
    }
    System.out.println(Key);
    matrix();
}

private void matrix() {
    int counter = 0;
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            matrix_arr[i][j] = Key.charAt(counter);
            System.out.print(matrix_arr[i][j] + " ");
            counter++;
        }
        System.out.println();
    }
}

private String format(String old_text) {
    int i = 0;
    int len = 0;
```

```
        String text = new String();
        len = old_text.length();
        for (int tmp = 0; tmp < len; tmp++) {
            if (old_text.charAt(tmp) == 'j') {
                text = text + 'i';
            } else
                text = text + old_text.charAt(tmp);
        }
        len = text.length();
        for (i = 0; i < len; i = i + 2) {
            if (text.charAt(i + 1) == text.charAt(i)) {
                text = text.substring(0, i + 1) + 'x' +
text.substring(i + 1);
            }
        }
        return text;
    }

    private String[] make_pairs(String new_string) {
        String my_string = format(new_string);
        int size = my_string.length();
        if (size % 2 != 0) {
            size++;
            my_string = my_string + 'x';
        }
        String x[] = new String[size / 2];
        int counter = 0;
        for (int i = 0; i < size / 2; i++) {
            x[i] = my_string.substring(counter, counter + 2);
            counter = counter + 2;
        }
        return x;
    }

    public int[] getdimensions(char letter) {
        int[] key = new int[2];
        if (letter == 'j')
            letter = 'i';
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (matrix_arr[i][j] == letter) {
                    key[0] = i;
                    key[1] = j;
```

```
                break;
            }
        }
    }
    return key;
}

public String decrypt(String src,String plain) {
    String arr[] = make_pairs(src);
    String code = new String();
    char one;
    char two;
    int arr1[] = new int[2];
    int arr2[] = new int[2];
    String withspace="";
    for (int i = 0; i < arr.length; i++) {
        one = arr[i].charAt(0);
        two = arr[i].charAt(1);
        arr1 = getdimensions(one);
        arr2 = getdimensions(two);
        if (arr1[0] == arr2[0]) {
            if (arr1[1] > 0)
                arr1[1]--;
            else
                arr1[1] = 4;
            if (arr2[1] > 0)
                arr2[1]--;
            else
                arr2[1] = 4;
        } else if (arr1[1] == arr2[1]) {
            if (arr1[0] > 0)
                arr1[0]--;
            else
                arr1[0] = 4;
            if (arr2[0] > 0)
                arr2[0]--;
            else
                arr2[0] = 4;
        } else {
            int temp = arr1[1];
            arr1[1] = arr2[1];
            arr2[1] = temp;
        }
```

```
            code = code + matrix_arr[arr1[0]][arr1[1]] +
matrix_arr[arr2[0]][arr2[1]];
        }
        int length = code.length();
        String text = "";
        for (int i = 0; i < length; i++) {
            if (code.charAt(i) != 'x')
                text += code.charAt(i);
        }
        int k=0;
        for(int i=0;i<plain.length();i++)
        {
          if(plain.charAt(i)!=' ' && k<text.length())
          {
            withspace+=text.charAt(k);
            k+=1;
          }
          else
          {
            withspace+=" ";
          }
        }
        return withspace;
    }

    public String encrypt(String src,String plain) {
        String arr[] = make_pairs(src);
        String code = new String();
        String newcode="";
        char one;
        char two;
        int arr1[] = new int[2];
        int arr2[] = new int[2];
        for (int i = 0; i < arr.length; i++) {
            one = arr[i].charAt(0);
            two = arr[i].charAt(1);
            arr1 = getdimensions(one);
            arr2 = getdimensions(two);
            if (arr1[0] == arr2[0]) {
                if (arr1[1] < 4)
                    arr1[1]++;
                else
                    arr1[1] = 0;
```

```
                if (arr2[1] < 4)
                    arr2[1]++;
                else
                    arr2[1] = 0;
            } else if (arr1[1] == arr2[1]) {
                if (arr1[0] < 4)
                    arr1[0]++;
                else
                    arr1[0] = 0;
                if (arr2[0] < 4)
                    arr2[0]++;
                else
                    arr2[0] = 0;
            } else {
                int temp = arr1[1];
                arr1[1] = arr2[1];
                arr2[1] = temp;
            }
            code = code + matrix_arr[arr1[0]][arr1[1]] +
matrix_arr[arr2[0]][arr2[1]];
        }
        int k=0;
        for(int i=0;i<plain.length();i++)
        {
          if(plain.charAt(i)!=' ' && k<code.length())
          {
            newcode+=code.charAt(k);
            k+=1;
          }
          else
          {
            newcode+=" ";
          }
        }
        return newcode;
    }
}
```

Encryption

```
import java.util.Scanner;
import java.lang.String.*;
import java.io.*;
```

```
public class Main {
    private String KeyWord = new String();
    private String Key = new String();
    private char matrix_arr[][] = new char[5][5];

    public static void main(String[] args)throws IOException {
        InputStreamReader ir=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(ir);
        String keyword = "", plainText = "",cipher="";
        int c;
        Main object = new Main();
        do {
            System.out.println("Playfair Cipher");
            System.out.println("Enter the choice");
            System.out.println("Menu");
            System.out.println("1. Read input");
            System.out.println("2. Read Key");
            System.out.println("3. Matrix");
            System.out.println("4. Encryption");
            int ch = Integer.parseInt(br.readLine());
            switch (ch) {
                case 1:
                    System.out.println("Enter a Text : ");
                    plainText = br.readLine();
                    break;
                case 2:
                    System.out.println("Enter key");
                    keyword = br.readLine();
                    break;
                case 3:
                    object.remove_duplicates(keyword);
                    object.generate_key();
                    if (plainText.replace(" ","").length() % 2 != 0) {
                        plainText = plainText + "x";
                    }
                    break;
                case 4:
                    System.out.println("Encryption : " +
object.encrypt(plainText.replace(" ",""),plainText));
                    break;
            }
            System.out.println("Enter 1 to continue");
            c = Integer.parseInt(br.readLine());
```

```
    } while (c == 1);
}

// remove duplicates in the key
public void remove_duplicates(String k) {
    String new_key = new String();
    boolean flag = false;
    new_key = new_key + k.charAt(0);
    for (int i = 1; i < k.length(); i++) {
        for (int j = 0; j < new_key.length(); j++) {
            if (k.charAt(i) == new_key.charAt(j)) {
                flag = true;
            }
        }
        if (flag == false)
            new_key = new_key + k.charAt(i);
        flag = false;
    }
    KeyWord = new_key;
}

// generate the key that has to be filled in the matrix
// eliminating i if j is present and vice versa
public void generate_key() {
    boolean flag = true;
    char curr;
    Key = KeyWord;
    for (int i = 0; i < 26; i++) {
        curr = (char) (i + 97);
        if (curr == 'j')
            continue;
        for (int j = 0; j < KeyWord.length(); j++) {
            if (curr == KeyWord.charAt(j)) {
                flag = false;
                break;
            }
        }
        if (flag)
            Key = Key + curr;
        flag = true;
    }
    System.out.println(Key);
    matrix();
```

```
    }

    // fill the 5X5 matrix using the key
    private void matrix() {
        int counter = 0;
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                matrix_arr[i][j] = Key.charAt(counter);
                System.out.print(matrix_arr[i][j] + " ");
                counter++;
            }
            System.out.println();
        }
    }

    // format done based on i and j
    // if same letter appears twice include a 'x' in between
    private String format(String old_text) {
        int i = 0;
        int len = 0;
        String text = new String();
        len = old_text.length();
        for (int tmp = 0; tmp < len; tmp++) {
            if (old_text.charAt(tmp) == 'j') {
                text = text + 'i';
            } else
                text = text + old_text.charAt(tmp);
        }
        len = text.length();
        for (i = 0; i < len; i = i + 2) {
            if (text.charAt(i + 1) == text.charAt(i)) {
                text = text.substring(0, i + 1) + 'x' +
text.substring(i + 1);
            }
        }
        return text;
    }

    // plaintext is grouped into pairs of two
    private String[] make_pairs(String new_string) {
        String my_string = format(new_string);
        int size = my_string.length();
        if (size % 2 != 0) {
```

```
            size++;
            my_string = my_string + 'x';
        }
        String x[] = new String[size / 2];
        int counter = 0;
        for (int i = 0; i < size / 2; i++) {
            x[i] = my_string.substring(counter, counter + 2);
            counter = counter + 2;
        }
        return x;
    }

    // return the row and column of the letter
    public int[] getdimensions(char letter) {
        int[] key = new int[2];
        if (letter == 'j')
            letter = 'i';
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (matrix_arr[i][j] == letter) {
                    key[0] = i;
                    key[1] = j;
                    break;
                }
            }
        }
        return key;
    }

    public String encrypt(String src,String plain) {
        String arr[] = make_pairs(src);
        String code = new String();
        String newcode="";
        char one;
        char two;
        int arr1[] = new int[2];
        int arr2[] = new int[2];
        for (int i = 0; i < arr.length; i++) {
            one = arr[i].charAt(0);
            two = arr[i].charAt(1);
            arr1 = getdimensions(one);
            arr2 = getdimensions(two);
            if (arr1[0] == arr2[0]) {
```

```
                if (arr1[1] < 4)
                    arr1[1]++;
                else
                    arr1[1] = 0;
                if (arr2[1] < 4)
                    arr2[1]++;
                else
                    arr2[1] = 0;
            } else if (arr1[1] == arr2[1]) {
                if (arr1[0] < 4)
                    arr1[0]++;
                else
                    arr1[0] = 0;
                if (arr2[0] < 4)
                    arr2[0]++;
                else
                    arr2[0] = 0;
            } else {
                int temp = arr1[1];
                arr1[1] = arr2[1];
                arr2[1] = temp;
            }
            code = code + matrix_arr[arr1[0]][arr1[1]] +
matrix_arr[arr2[0]][arr2[1]];
        }
        int k=0;
        for(int i=0;i<plain.length();i++)
        {
          if(plain.charAt(i)!=' ' && k<code.length())
          {
            newcode+=code.charAt(k);
            k+=1;
          }
          else
          {
            newcode+=" ";
          }
        }
        return newcode;
    }
}
```

Decryption

```
import java.util.Scanner;
import java.lang.String.*;
import java.io.*;

public class Main {
    private String KeyWord = new String();
    private String Key = new String();
    private char matrix_arr[][] = new char[5][5];

    public static void main(String[] args)throws IOException {
        InputStreamReader ir=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(ir);
        String keyword = "", plainText = "",cipher="";
        int c;
        Main object = new Main();
        do {
            System.out.println("Playfair Cipher");
            System.out.println("Enter the choice");
            System.out.println("Menu");
            System.out.println("1. Read input");
            System.out.println("2. Read Key");
            System.out.println("3. Matrix");
            System.out.println("4. Decryption");
            int ch = Integer.parseInt(br.readLine());
            switch (ch) {
                case 1:
                    System.out.println("Enter encytped text : ");
                    cipher = br.readLine();
                    break;
                case 2:
                    System.out.println("Enter key");
                    keyword = br.readLine();
                    break;
                case 3:
                    object.remove_duplicates(keyword);
                    object.generate_key();
                    break;
                case 4:
                    System.out.println("Decryption : " +
object.decrypt(cipher.replace(" ",""),cipher));
                    break;
            }
            System.out.println("Enter 1 to continue");
```

```
            c = Integer.parseInt(br.readLine());
        } while (c == 1);
    }

    // remove duplicates in the key
    public void remove_duplicates(String k) {
        String new_key = new String();
        boolean flag = false;
        new_key = new_key + k.charAt(0);
        for (int i = 1; i < k.length(); i++) {
            for (int j = 0; j < new_key.length(); j++) {
                if (k.charAt(i) == new_key.charAt(j)) {
                    flag = true;
                }
            }
            if (flag == false)
                new_key = new_key + k.charAt(i);
            flag = false;
        }
        KeyWord = new_key;
    }

    // generate the key that has to be filled in the matrix
    // eliminating i if j is present and vice versa
    public void generate_key() {
        boolean flag = true;
        char curr;
        Key = KeyWord;
        for (int i = 0; i < 26; i++) {
            curr = (char) (i + 97);
            if (curr == 'j')
                continue;
            for (int j = 0; j < KeyWord.length(); j++) {
                if (curr == KeyWord.charAt(j)) {
                    flag = false;
                    break;
                }
            }
            if (flag)
                Key = Key + curr;
            flag = true;
        }
        System.out.println(Key);
```

```
        matrix();
    }

    // fill the 5X5 matrix using the key
    private void matrix() {
        int counter = 0;
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                matrix_arr[i][j] = Key.charAt(counter);
                System.out.print(matrix_arr[i][j] + " ");
                counter++;
            }
            System.out.println();
        }
    }

    // format done based on i and j
    // if same letter appears twice include a 'x' in between
    private String format(String old_text) {
        int i = 0;
        int len = 0;
        String text = new String();
        len = old_text.length();
        for (int tmp = 0; tmp < len; tmp++) {
            if (old_text.charAt(tmp) == 'j') {
                text = text + 'i';
            } else
                text = text + old_text.charAt(tmp);
        }
        len = text.length();
        for (i = 0; i < len; i = i + 2) {
            if (text.charAt(i + 1) == text.charAt(i)) {
                text = text.substring(0, i + 1) + 'x' +
text.substring(i + 1);
            }
        }
        return text;
    }

    // plaintext is grouped into pairs of two
    private String[] make_pairs(String new_string) {
        String my_string = format(new_string);
        int size = my_string.length();
```

```
    if (size % 2 != 0) {
        size++;
        my_string = my_string + 'x';
    }
    String x[] = new String[size / 2];
    int counter = 0;
    for (int i = 0; i < size / 2; i++) {
        x[i] = my_string.substring(counter, counter + 2);
        counter = counter + 2;
    }
    return x;
}

// return the row and column of the letter
public int[] getdimensions(char letter) {
    int[] key = new int[2];
    if (letter == 'j')
        letter = 'i';
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (matrix_arr[i][j] == letter) {
                key[0] = i;
                key[1] = j;
                break;
            }
        }
    }
    return key;
}

//decrypt the string using the encrypted message as plaintext
public String decrypt(String src,String plain) {
    String arr[] = make_pairs(src);
    String code = new String();
    char one;
    char two;
    int arr1[] = new int[2];
    int arr2[] = new int[2];
    String withspace="";
    for (int i = 0; i < arr.length; i++) {
        one = arr[i].charAt(0);
        two = arr[i].charAt(1);
        arr1 = getdimensions(one);
```

```
            arr2 = getdimensions(two);
            if (arr1[0] == arr2[0]) {
                if (arr1[1] > 0)
                    arr1[1]--;
                else
                    arr1[1] = 4;
                if (arr2[1] > 0)
                    arr2[1]--;
                else
                    arr2[1] = 4;
            } else if (arr1[1] == arr2[1]) {
                if (arr1[0] > 0)
                    arr1[0]--;
                else
                    arr1[0] = 4;
                if (arr2[0] > 0)
                    arr2[0]--;
                else
                    arr2[0] = 4;
            } else {
                int temp = arr1[1];
                arr1[1] = arr2[1];
                arr2[1] = temp;
            }
            code = code + matrix_arr[arr1[0]][arr1[1]] +
matrix_arr[arr2[0]][arr2[1]];
        }
        int length = code.length();
        String text = "";
        for (int i = 0; i < length; i++) {
            if (code.charAt(i) != 'x')
                text += code.charAt(i);
        }
        int k=0;
        for(int i=0;i<plain.length();i++)
        {
          if(plain.charAt(i)!=' ' && k<text.length())
          {
            withspace+=text.charAt(k);
            k+=1;
          }
          else
          {
```

```
            withspace+=" ";
          }
        }
        return withspace;
    }

}
```

Rail Fence

```
import java.util.*;
public class Main {
    static ArrayList < ArrayList < String > > fence = new ArrayList <
ArrayList < String >> ();
    public static String Encrypt(String text, int key) {
        String res = "";
        fence = new ArrayList < ArrayList < String >> (key);

        for (int i = 0; i < key; i++) {
            fence.add(new ArrayList < String > (text.length()));
        }

        int down = 1, up = 0;
        int i = 0, x = 0;
        for (char c: text.toCharArray()) {
            fence.get(i).add(x, Character.toString(c));

            for (int z = 0; z < key; z++) {
                if (z == i)
                    continue;
                else
                    fence.get(z).add(x, " ");
            }

            if (down == 1)
                i += 1;
            else
                i -= 1;

            if (i == key) {
                down = 0;
                up = 1;
                i -= 2;
            }
```

```
            if (i == -1) {
                down = 1;
                up = 0;
                i += 2;
            }
            x += 1;
        }

        for (ArrayList < String > arrayList: fence)
            System.out.println(arrayList);
        System.out.println("");
        for (i = 0; i < key; i++) {
            for (String c: fence.get(i)) {
                res += c;
            }
        }
        return res;
    }

    public static String Decrypt(String enc, int key) {
        String res = "";
        int down = 1, up = 0;
        int i = 0, x = 0;
        for (i = 0; i < enc.length(); i++) {
            res += ((ArrayList) fence.get(x)).get(i);

            if (down == 1)
                x += 1;
            else
                x -= 1;

            if (x == key) {
                down = 0;
                up = 1;
                x -= 2;
            }

            if (x == -1) {
                down = 1;
                up = 0;
                x += 2;
            }
```

```
        }
        return res;
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print(("Text : "));
        String text = sc.nextLine();
        text = text.replace(" ", "");
        text = text.toUpperCase();
        System.out.print("Key : ");
        Integer key = sc.nextInt();

        System.out.println("");
        System.out.println("1. Encrypt");
        System.out.println("2. Decrypt");
        System.out.println("0. Exit");
        System.out.print("Enter Choice : ");
        int op = sc.nextInt();
        while (op != 0) {

            switch (op) {
                case 1:
                    System.out.println("");
                    String enc = Encrypt(text, key);
                    System.out.println("Encrypted Text : " +
enc.replace(" ", ""));
                    break;
                case 2:
                    String dec = Decrypt(text, key);
                    System.out.println("");
                    System.out.println("Decrypted Text : " + dec);
                    break;
                default:
                    System.out.println("Invalid Choice !!");
            }

            System.out.println("");
            System.out.println("1. Encrypt");
            System.out.println("2. Decrypt");
            System.out.println("0. Exit");
            System.out.print("Enter Choice : ");
            op = sc.nextInt();
```

```
        }
    }
}
```

Encryption

```
import java.util.*;
public class Main {
    static ArrayList < ArrayList < String > > fence = new ArrayList <
ArrayList < String >> ();
    public static String Encrypt(String text, int key) {
        String res = "";
        fence = new ArrayList < ArrayList < String >> (key);

        for (int i = 0; i < key; i++) {
            fence.add(new ArrayList < String > (text.length()));
        }

        int down = 1, up = 0;
        int i = 0, x = 0;
        for (char c: text.toCharArray()) {
            fence.get(i).add(x, Character.toString(c));

            for (int z = 0; z < key; z++) {
                if (z == i)
                    continue;
                else
                    fence.get(z).add(x, " ");
            }

            if (down == 1)
                i += 1;
            else
                i -= 1;

            if (i == key) {
                down = 0;
                up = 1;
                i -= 2;
            }

            if (i == -1) {
                down = 1;
```

```
                up = 0;
                i += 2;
            }
            x += 1;
        }

        for (ArrayList < String > arrayList: fence)
            System.out.println(arrayList);
        System.out.println("");
        for (i = 0; i < key; i++) {
            for (String c: fence.get(i)) {
                res += c;
            }
        }
        return res;
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print(("Text : "));
        String text = sc.nextLine();
        text = text.replace(" ", "");
        text = text.toUpperCase();
        System.out.print("Key : ");
        Integer key = sc.nextInt();

        System.out.println("");
        System.out.println("Encrypt");
        String enc = Encrypt(text, key);
        System.out.println("Encrypted Text : " + enc.replace(" ", ""));
        }
    }
```

Question 8

```
import java.util.*;
public class Main {
    public static String Encrypt(String text, int key) {
        String res = "";
        ArrayList < ArrayList < String > > fence = new ArrayList <
ArrayList < String >> (key);

        for (int i = 0; i < key; i++) {
```

```
        fence.add(new ArrayList < String > (text.length()));
    }

    int down = 1, up = 0;
    int i = 0, x = 0;
    for (char c: text.toCharArray()) {
        fence.get(i).add(x, Character.toString(c));

        for (int z = 0; z < key; z++) {
            if (z == i)
                continue;
            else
                fence.get(z).add(x, " ");
        }

        if (down == 1)
            i += 1;
        else
            i -= 1;

        if (i == key) {
            down = 0;
            up = 1;
            i -= 2;
        }

        if (i == -1) {
            down = 1;
            up = 0;
            i += 2;
        }
        x += 1;
    }

    for (ArrayList < String > arrayList: fence)
        System.out.println(arrayList);
    System.out.println("");
    for (i = 0; i < key; i++) {
        for (String c: fence.get(i)) {
            res += c;
        }
    }
    return res;
```

```
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print(("Text : "));
        String text = sc.nextLine();
        text = text.replace(" ", "");
        text = text.toUpperCase();
        System.out.print("Key : ");
        Integer key = sc.nextInt();

        System.out.println("");
        System.out.println("Encrypt");
        String enc = Encrypt(text, key);
        System.out.println("Encrypted Text : " + enc.replace(" ", ""));


        System.out.print("Key : ");
        key = sc.nextInt();

        System.out.println("");
        System.out.println("Encrypt");
        enc = Encrypt(enc.replace(" ",""), key);
        System.out.println("Encrypted Text : " + enc.replace(" ", ""));
    }
    }
```

Row

```
import java.util.*;
public class Main {
    static ArrayList < ArrayList < String > > fence = new ArrayList <
ArrayList < String >> ();
    public static String Encrypt(String text, int key) {
        String res = "";
        fence = new ArrayList < ArrayList < String >> (key);

        for (int i = 0; i < key; i++) {
            fence.add(new ArrayList < String > (text.length()));
        }

        int down = 1, up = 0;
        int i = 0, x = 0;
```

```
        for (char c: text.toCharArray()) {
            fence.get(i).add(x, Character.toString(c));

            for (int z = 0; z < key; z++) {
                if (z == i)
                    continue;
                else
                    fence.get(z).add(x, " ");
            }

            if (down == 1)
                i += 1;
            else
                i -= 1;

            if (i == key) {
                down = 0;
                up = 1;
                i -= 2;
            }

            if (i == -1) {
                down = 1;
                up = 0;
                i += 2;
            }
            x += 1;
        }

        for (ArrayList < String > arrayList: fence)
            System.out.println(arrayList);
        System.out.println("");
        for (i = 0; i < key; i++) {
            for (String c: fence.get(i)) {
                res += c;
            }
        }
        return res;
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print(("Text : "));
```

```
        String text = sc.nextLine();
        text = text.replace(" ", "");
        text = text.toUpperCase();
        System.out.print("Key : ");
        Integer key = sc.nextInt();

        System.out.println("");
        System.out.println("Encrypt");
        String enc = Encrypt(text, key);
        System.out.println("Encrypted Text : " + enc.replace(" ", ""));
        }
    }
```

Encryption

```
import java.util.*;

public class Main {
    public static String Encrypt(String text, String key) {
        String res = "";
        String alpha = "ZYXWVUTSRQPONMLKJIHGFEDCBA";

        int n;
        if (text.length() % key.length() > 0)
            n = (text.length() / key.length()) + 1;
        else
            n = text.length() / key.length();

        char[][] rc = new char[n][key.length()];

        int i = 0, j = 0;
        for (char c : text.toCharArray()) {
            rc[i][j] = c;
            j += 1;

            if (j == key.length()) {
                j = 0;
                i += 1;
            }
        }

        int x = key.length() - j;
        while (i != n && j != key.length()) {
            rc[i][j] = alpha.charAt(x - 1);
```

```
            x -= 1;
            j += 1;
        }
        for (i = 0; i < n; i++) {
            for (j = 0; j < key.length(); j++) {
                System.out.print(rc[i][j] + " ");
            }
            System.out.println("");
        }
        System.out.println("");

        for (i = 1; i <= key.length(); i++)
            for (j = 0; j < n; j++) {
                res += rc[j][key.indexOf(Integer.toString(i))];
            }
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print(("Text : "));
        String text = sc.nextLine();
        text = text.replace(" ", "");
        text = text.toUpperCase();
        System.out.print("Key : ");
        String key = sc.nextLine();

        String enc = "";

        System.out.println("");
        System.out.println("1. Encrypt");
        System.out.println("0. Exit");
        System.out.print("Enter Choice : ");
        int op = sc.nextInt();
        while (op != 0) {

            switch (op) {
                case 1:
                    System.out.println("");
                    enc = Encrypt(text, key);
                    System.out.println("Encrypted Text : " +
enc.replace(" ", ""));
                    break;
```

```
                default:
                    System.out.println("Invalid Choice !!");
            }

            System.out.println("");
            System.out.println("1. Encrypt");
            System.out.println("0.Exit");
            System.out.print("Enter Choice : ");
            op = sc.nextInt();
        }
    }
}
```

Decryption

```
import java.util.*;

public class Main {

    public static String Decrypt(String enc, String key) {
        String res = "";
        int x = 0;
        int n = enc.length() / key.length();
        char[][] rc = new char[n][key.length()];

        for (int i = 1; i <= key.length(); i++) {
            for (int j = 0; j < n; j++) {
                rc[j][key.indexOf(Integer.toString(i))] =
enc.charAt(x);
                x += 1;
            }
        }

        for (int i = 0; i < n; i++)
            for (int j = 0; j < key.length(); j++)
                res += (rc[i][j]);

        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print(("Text : "));
        String text = sc.nextLine();
```

```
        text = text.replace(" ", "");
        text = text.toUpperCase();
        System.out.print("Key : ");
        String key = sc.nextLine();

        String enc = "";

        System.out.println("");
        System.out.println("1. Decrypt");
        System.out.println("0. Exit");
        System.out.print("Enter Choice : ");
        int op = sc.nextInt();
        while (op != 0) {

            switch (op) {
                case 1:
                    System.out.println("");
                    String dec = Decrypt(text, key);
                    System.out.println("Decrypted Text : " +
dec.replace(" ", "").substring(0, text.length()));
                    break;
                default:
                    System.out.println("Invalid Choice !!");
            }

            System.out.println("");
            System.out.println("1. Decrypt");
            System.out.println("0. Exit");
            System.out.print("Enter Choice : ");
            op = sc.nextInt();
        }
    }
}
```

Question 9

```
import java.util.*;

public class Main {
    public static String Encrypt(String text, String key) {
        String res = "";
        String alpha = "ZYXWVUTSRQPONMLKJIHGFEDCBA";

        int n;
```

```
    if (text.length() % key.length() > 0)
        n = (text.length() / key.length()) + 1;
    else
        n = text.length() / key.length();

    char[][] rc = new char[n][key.length()];

    int i = 0, j = 0;
    for (char c : text.toCharArray()) {
        rc[i][j] = c;
        j += 1;

        if (j == key.length()) {
            j = 0;
            i += 1;
        }
    }

    int x = key.length() - j;
    while (i != n && j != key.length()) {
        rc[i][j] = alpha.charAt(x - 1);
        x -= 1;
        j += 1;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < key.length(); j++) {
            System.out.print(rc[i][j] + " ");
        }
        System.out.println("");
    }
    System.out.println("");

    for (i = 1; i <= key.length(); i++)
        for (j = 0; j < n; j++) {
            res += rc[j][key.indexOf(Integer.toString(i))];
        }
    return res;
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print(("Text : "));
    String text = sc.nextLine();
```

```
        text = text.replace(" ", "");
        text = text.toUpperCase();
        System.out.print("Key : ");
        String key = sc.nextLine();

        String enc = "";

        System.out.println("");
        System.out.println("Encrypt");
        System.out.println("");
        enc = Encrypt(text, key);
        System.out.println("Encrypted Text : " + enc.replace(" ", ""));

        System.out.print("Key : ");
        key = sc.nextLine();

        System.out.println("");
        System.out.println("Encrypt");
        System.out.println("");
        enc = Encrypt(enc.replace(" ", ""), key);
        System.out.println("Encrypted Text : " + enc.replace(" ", ""));
    }
}
```

RSA

```
import java.util.*;
import java.io.*;
import java.math.*;

public class Main {

    static BigInteger N, p, q, phi, e, d;
    static Random r;
    static int bitlength = 1024;

    public static void init(){
        r = new Random();
        p = BigInteger.probablePrime(bitlength, r);
        q = BigInteger.probablePrime(bitlength, r);
        N = p.multiply(q);
        phi =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
```

```
        e = BigInteger.probablePrime(bitlength / 2, r);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 &&
e.compareTo(phi) < 0)
        {
            e.add(BigInteger.ONE);
        }
        d = e.modInverse(phi);
    }
    public static byte[] encrypt(byte[] text)
    {
        return (new BigInteger(text)).modPow(e, N).toByteArray();
    }

    public static byte[] decrypt(byte[] text)
    {
        return (new BigInteger(text)).modPow(d, N).toByteArray();
    }

    private static String bytesToString(byte[] byteArray)
    {
        String text = "";
        for (byte b : byteArray)
        {
            text += Byte.toString(b);
        }
        return text;
    }

    public static void main(String[] args){
        String text;
        Scanner s = new Scanner(System.in);
        init();
        System.out.print("Enter text to encrypt:");
        text = s.nextLine();

        System.out.println("BYTES TO ENCRYPT:" +
bytesToString(text.getBytes()));

        byte[] encrypted = encrypt(text.getBytes());
        // System.out.println("ENCRYPTED BYTES: " +
bytesToString(encrypted));

        byte[] decrypted = decrypt(encrypted);
```

```
        System.out.println("BYTES DECRYPTED:" +
bytesToString(decrypted));
        System.out.println("DECRYPTED TEXT: " + new String(decrypted));
    }
}
```

SHA

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.io.*;
import java.util.*;


public class Main {
    public static String encrypt(String input)
    {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-1");

            // digest() method is called to calculate message digest
            // of the input string returned as array of byte
            byte[] messageDigest = md.digest(input.getBytes());

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

            // Convert message digest into hex value
            String hashtext = no.toString(16);

            // Add preceding 0s to make it 32 bit
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }

            // return the HashText
            return hashtext;
        }

        // For specifying wrong message digest algorithms
        catch (NoSuchAlgorithmException e) {
            System.out.println("[ERR]:" + e.getMessage());
            return "";
```

```
        }
    }

    public static void main(String args[]){
        Scanner s = new Scanner(System.in);
        System.out.print("Enter text to encrypt:");
        String text = s.next();
        System.out.println("MESSAGE DIGEST:" + encrypt(text));
    }
}
```

Vigenere

```
import java.util.*;
import java.io.*;
import java.lang.Math;
public class Main {
    public static final String text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    static String encrypt(String plain, String keyword) {
        int p=plain.length();
        String s="";
        int k=0;
        int z=0;
        for(int i=0;i<p;i++)
        {
            char a=plain.charAt(i);
            if(a==' ')
            {
                s=s+a;
                continue;
            }
            char b=keyword.charAt(z);
            z+=1;
            k=text.indexOf(a)+text.indexOf(b);
            if(k>25)
                k=k%26;
            s=s+text.charAt(k);
        }
        return s;
    }

    static String decrypt(String cipher, String keyword) {
        int c=cipher.length();
```

```
        String s="";
        int k=0;
        int z=0;
        for(int i=0;i<c;i++)
        {
            char a=cipher.charAt(i);
            if(a==' ')
            {
                s=s+a;
                continue;
            }
            char b=keyword.charAt(z);
            z+=1;
            k=text.indexOf(a)-text.indexOf(b);
            k=(k+26)%26;
            s=s+text.charAt(k);
        }
        return s;
    }

    static String getkey(String plain,String key) {
        int l1,l2;
        l1=plain.length();
        l2=key.length();
        int l=l1/l2;
        String s="";
        for(int i=0;i<l*l2;i=i+l2)
        {
            s=s+key;
        }
        int x=l1-s.length();
        for(int i=0;i<x;i++)
            s=s+key.charAt(i);
        return(s);
    }

    static void print_table()
    {
        String str="";
        String s ="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
";
        String a=" | 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";
```

```
        String
b="_____________________________________________________________";
        str=""+s;
        System.out.println(a);
        System.out.println(b);
        for(int i=0;i<=50;i=i+2)
        {
            str=s.substring(i) + s.substring(0,i);
            System.out.print(str.substring(0,1)+"| ");
            System.out.println(str);
        }
    }

    public static void main(String args[]) throws IOException {
        int ch;
        String plain = "", cipher = "", c, result;
        String key="",keyword="";
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ir);
        do {
            System.out.println("Vigenere Cipher");
            System.out.println("1. Read Plaintext");
            System.out.println("2. Read Key");
            System.out.println("3. Display Table");
            System.out.println("4. Encrypt");
            System.out.println("5. Decrypt");
            ch = Integer.parseInt(br.readLine());
            switch (ch) {
                case 1: {
                    System.out.println("Enter Plaintext");
                    plain = br.readLine();
                    break;
                }
                case 2: {
                    System.out.println("Enter key");
                    key = br.readLine();
                    keyword=getkey(plain.replace(" ",""),key);
                    System.out.println("Keyword is:"+keyword);
                    break;
                }
                case 3: {
                    print_table();
                    break;
```

```
                }
                case 4: {
                    cipher = encrypt(plain.toUpperCase(), keyword);
                    System.out.println("Encypted String:" + cipher);
                    break;
                }
                case 5: {
                    result = decrypt(cipher.toUpperCase(), keyword);
                    System.out.println("Decrypted String:" + result);
                    break;
                }
            }
            System.out.println("Do you want to exit?(y/n)");
            c = br.readLine();
        } while (!(c.equals("y")));
    }
}
```

Encryption

```
import java.util.*;
import java.io.*;
import java.lang.Math;
public class Main {
    public static final String text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    static String encrypt(String plain, String keyword) {
        int p=plain.length();
        String s="";
        int k=0;
        int z=0;
        for(int i=0;i<p;i++)
        {
            char a=plain.charAt(i);
            if(a==' ')
            {
                s=s+a;
                continue;
            }
            char b=keyword.charAt(z);
            z+=1;
            k=text.indexOf(a)+text.indexOf(b);
            if(k>25)
```

```
                k=k%26;
            s=s+text.charAt(k);
        }
        return s;
    }

    static String getkey(String plain,String key) {
        int l1,l2;
        l1=plain.length();
        l2=key.length();
        int l=l1/l2;
        String s="";
        for(int i=0;i<l*l2;i=i+l2)
        {
            s=s+key;
        }
        int x=l1-s.length();
        for(int i=0;i<x;i++)
            s=s+key.charAt(i);
        return(s);
    }

    static void print_table()
    {
        String str="";
        String s ="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
";
        String a=" | 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";
        String
b="_____________________________________________________";
        str=""+s;
        System.out.println(a);
        System.out.println(b);
        for(int i=0;i<=50;i=i+2)
        {
            str=s.substring(i) + s.substring(0,i);
            System.out.print(str.substring(0,1)+"| ");
            System.out.println(str);
        }
    }

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

```
        int ch;
        String plain = "", cipher = "", c, result;
        String key="",keyword="";
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ir);
        do {
            System.out.println("Vigenere Cipher");
            System.out.println("1. Read Plaintext");
            System.out.println("2. Read Key");
            System.out.println("3. Display Table");
            System.out.println("4. Encrypt");
            ch = Integer.parseInt(br.readLine());
            switch (ch) {
                case 1: {
                    System.out.println("Enter Plaintext");
                    plain = br.readLine();
                    break;
                }
                case 2: {
                    System.out.println("Enter key");
                    key = br.readLine();
                    keyword=getkey(plain.replace(" ",""),key);
                    System.out.println("Keyword is:"+keyword);
                    break;
                }
                case 3: {
                    print_table();
                    break;
                }
                case 4: {
                    cipher = encrypt(plain.toUpperCase(), keyword);
                    System.out.println("Encypted String:" + cipher);
                    break;
                }
            }
            System.out.println("Do you want to exit?(y/n)");
            c = br.readLine();
        } while (!(c.equals("y")));
    }
}
```

Decryption

```
import java.util.*;
import java.io.*;
import java.lang.Math;
public class Main {
    public static final String text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    static String decrypt(String cipher, String keyword) {
        int c=cipher.length();
        String s="";
        int k=0;
        int z=0;
        for(int i=0;i<c;i++)
        {
            char a=cipher.charAt(i);
            if(a==' ')
            {
                s=s+a;
                continue;
            }
            char b=keyword.charAt(z);
            z+=1;
            k=text.indexOf(a)-text.indexOf(b);
            k=(k+26)%26;
            s=s+text.charAt(k);
        }
        return s;
    }

    static String getkey(String plain,String key) {
        int l1,l2;
        l1=plain.length();
        l2=key.length();
        int l=l1/l2;
        String s="";
        for(int i=0;i<l*l2;i=i+l2)
        {
            s=s+key;
        }
        int x=l1-s.length();
        for(int i=0;i<x;i++)
            s=s+key.charAt(i);
        return(s);
```

```
    }

    static void print_table()
    {
        String str="";
        String s ="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
";
        String a=" | 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";
        String
b="_____________________________________________________";
        str=""+s;
        System.out.println(a);
        System.out.println(b);
        for(int i=0;i<=50;i=i+2)
        {
            str=s.substring(i) + s.substring(0,i);
            System.out.print(str.substring(0,1)+"| ");
            System.out.println(str);
        }
    }


    public static void main(String args[]) throws IOException {
        int ch;
        String plain = "", cipher = "", c, result;
        String key="",keyword="";
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ir);
        do {
            System.out.println("Vigenere Cipher");
            System.out.println("1. Read encrypted text");
            System.out.println("2. Read Key");
            System.out.println("3. Display Table");
            System.out.println("4. Decrypt");
            ch = Integer.parseInt(br.readLine());
            switch (ch) {
                case 1: {
                    System.out.println("Enter encrypted text");
                    cipher = br.readLine();
                    break;
                }
                case 2: {
```

```
                    System.out.println("Enter key");
                    key = br.readLine();
                    keyword=getkey(cipher.replace(" ",""),key);
                    System.out.println("Keyword is:"+keyword);
                    break;
                }
                case 3: {
                    print_table();
                    break;
                }
                case 4: {
                    result = decrypt(cipher.toUpperCase(), keyword);
                    System.out.println("Decrypted String:" + result);
                    break;
                }
            }
            System.out.println("Do you want to exit?(y/n)");
            c = br.readLine();
        } while (!(c.equals("y")));
    }

}
```

ALGORITHMS:

AES

1. Derive the set of round keys from the cipher key.
2. Initialize the state array with the block data (plaintext).
3. Add the initial round key to the starting state array.
4. Substitute bytes in the state matrix with the help of sbox.
5. Shift bytes in each row depending on row number
6. Multiply the state matrix with a predefined matrix.
7. This matrix is xored with the round key.
8. Perform nine rounds of state manipulation.
9. Perform the tenth and final round of state manipulation.
10. Copy the final state array out as the encrypted data (ciphertext)

MD5

https://www.educba.com/md5-alogrithm/

CRT:

The following is a general construction to find a solution to a system of congruences using the Chinese remainder theorem:

1. Compute $N = n_1 \times n_2 \times \cdots \times n_k$.
2. For each $i = 1, 2, \ldots, k$, compute

$$y_i = \frac{N}{n_i} = n_1 n_2 \cdots n_{i-1} n_{i+1} \cdots n_k.$$

3. For each $i = 1, 2, \ldots, k$, compute $z_i \equiv y_i^{-1} \bmod n_i$ using Euclid's extended algorithm (z_i exists since $n_1, n_2, \ldots, n_k$ are pairwise coprime).
4. The integer $x = \sum_{i=1}^{k} a_i y_i z_i$ is a solution to the system of congruences, and $x \bmod N$ is the unique solution modulo N.

Miller rabin:

Miller-Rabin Primality Testing:

Similar to Fermat primality test, Miller-Rabin primality test could only determine if a number is a probable prime.

It is based on a basic principle where if $X^2 \equiv Y^2 mod N$, but $X! \equiv \pm Y mod N$, then N is composite.

The algorithm in simple steps can be written as,

Given a number $N(> 2)$ for which primality is to be tested,
Step 1: Find $N - 1 = 2^R . D$
Step 2: Choose A in range $[2, N - 2]$
Step 3: Compute $X_0 = A^D mod N$. If X_0 is ± 1, N can be prime.
Step 4: Compute $X_i = X_i - 1 mod N$. If $X_i = 1$, N is composite.
If $X_i = -1$, N is prime.
Step 5: Repeat **Step 4** for $R - 1$ times.
Step 6: If neither -1 or $+1$ appeared for X_i, N is composite.

Pseudocode for Miller-Rabin primality testing is given below.

SSN College of Engineering,
Department of Computer Science and Engineering
IT8761 Security Laboratory

Exercise 1:

To implement the substitution techniques: Caesar Cipher and Playfair Cipher

Programming Language: Java

Hints:

1a. Encryption Procedure for Caesar Cipher:

1. Read the plain text message
2. Read the key value (displacement)
3. To generate the cipher text , replace each letter of plaintext by a letter at the position specified by the key value down the alphabetical stream.
4. Display the cipher text.

Decryption Procedure for Caesar Cipher:

1. Use the cipher text as input
2. Use the same key value as displacement
3. To retrieve the plaintext text from cipher text, replace a letter of cipher text by the letter at the position specified by the key value in the reverse alphabetical stream.
4. Display the plain text.

1b. Encryption Procedure for Playfair Cipher:

1. Read the plain text message
2. Read the key value (a string without any repetition letters)
3. Construct a 5 X 5 matrix and fill in the key text in row wise manner.
4. Fill in the remaining cells of the matrix with the rest of the alphabets sans the letters of the key.
5. Split the plain text into two letter words without repetition.
6. If a pair has a repeated letter, insert filler like 'X'
7. If both letters fall in the same row, replace each with letter to right (wrapping back to start from end)
8. If both letters fall in the same column, replace each with the letter below it (wrapping to top from bottom)
9. Otherwise each letter is replaced by the letter in the same row and in the column of the other letter of the pair
10. Display the cipher text.

Decryption Procedure for Playfair Cipher:

1. Use the cipher text as input
2. Use the same key value
3. To retrieve the plaintext text from cipher text, split the plain text into two letter words without repetition.
4. Repeat the steps 7 to 9 of encryption to generate the plaintext
5. Display the plain text.

SSN College of Engineering,
Department of Computer Science and Engineering
IT8761 Security Laboratory

Exercise 2a:

To implement the substitution technique: Hill Cipher

Hints:

Encryption Procedure for Hill Cipher:

E: C= KP mod 26

1. Read the plain text message
2. Read the key (a square matrix, say order of $n \times n$)
3. Split the plain text into chunks of size n, convert them into their numerical equivalent and form a columnar matrix.
4. Perform matrix multiplication on K and plain text vector mod 26.
5. Generate cipher text by decoding the outcome of step 4 into equivalent alphabets.
6. Display the cipher text.

Decryption Procedure for Hill Cipher:

D: P= K^{-1}C mod 26

1. Use the cipher text as input
2. Compute the inverse matrix of K, that is K^{-1}
3. Encode the cipher text into their numerical equivalent and form a columnar matrix with respect to the order of K^{-1}.
4. Perform matrix multiplication on K^{-1} and cipher text vector mod 26.
5. Retrieve plain text by decoding the outcome of step 4 into equivalent alphabets.
6. Display the plain text.

Exercise 2b:

To implement the substitution technique: Vigenère Cipher

Programming Language: Java

Hints:

Encryption Procedure for Vigenère Cipher:

1. Read the plain text message
2. Read the key phrase
3. Construct a reference Vigenère table, where each row of table consists of all letters of the English alphabet.
    a. The first row starts with the letter a, and each following row is shifted by one letter (second row starts with b, third with c...).
4. To encrypt, pick a letter in the plaintext and its corresponding letter in the keyword, use the keyword letter and the plaintext letter as the row index and column index, respectively, and the entry at the row-column intersection is the letter in the cipher text.
5. Display the cipher text.

	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
A	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
B	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	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	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

Vigenère table

Decryption Procedure for Vigenère Cipher:

1. Use the cipher text as input
2. Use the same key phrase
3. To decrypt the cipher text, find in the row corresponding to the n-th letter of the key phrase a cell in which the n-th letter of the cipher text resides.
    a. Its column is denoted by the n-th letter of the plain text.
4. Display the plain text.

SSN College of Engineering,
Department of Computer Science and Engineering
IT8761 Security Laboratory

Exercise 3a:

To implement the transposition techniques: Rail fence Cipher

Programming Language: Java

Hints:

Encryption Procedure for Rail fence Cipher:

1. Read the plain text message
2. Read the key value (The key for the rail fence cipher is just the number of rails / levels)
3. Write the letters of the plaintext diagonally down to the right until you reach the number of rows specified by the key.
    - Then bounce back up diagonally until you hit the first row again. This continues until the end of the plaintext.
4. To generate the cipher text, read off along the rows one by one.
5. Display the cipher text.

Decryption Procedure for Rail fence Cipher:

1. Use the cipher text as input and number of rails as key value
2. Break up the cipher text letters into equal groups for each rail.
3. To retrieve the plaintext text from cipher text, read off the message vertically.
4. Display the plain text.

Exercise 3b:

To implement the transposition techniques: Row & Column Ciphers

Programming Language: Java

Hints:

Encryption Procedure for Row & Column Cipher:

1. Read the plain text message
2. Write the plaintext in row by row forming a matrix / grid of *m X n*.
3. Read the key value (a number with permuted digits ranging from*1* to number of columns in the plaintext matrix)
4. Use the key as column headers
5. Generate the cipher text with respect to the order of the column headers.
6. Display the cipher text.

Decryption Procedure for Row & Column Cipher:

1. Use the cipher text as input
2. The number of rows and columns of the cipher text matrix be now *n X m*
3. Write the cipher text row by row in *n* rows
4. Let the row headers be the permuted key values of 1 to n
    - If encryption key is 4 3 1 2 5 6 7
    - Decryption key will be 3 4 2 1 5 6 7
5. Take the letters in the order of key from the first column to retrieve the plain text. Repeat extracting letters in the other columns according to the key.
6. Display the plain text.

SSN College of Engineering,
Department of Computer Science and Engineering
IT8761 Security Laboratory

Exercise 4:

To implement the Data Encryption Standard (DES) Algorithm.

Programming Language: Java

Hints:

Key Generation

1. Initialize the permutation tables, left shift schedules.
2. Read the 64 bit key.
3. 64 bits goes through a permutation called PC-1(permuted choice) resulting 56 bits.
4. 56 bits are divided into two halves
5. Each half will be rotated left by 1 or 2 bits depending on the round
6. Both sides go through permute choice 2 (PC-2) which selects 24 bits from left and right resulting a 48 bit round key.

Encryption Procedure for DES:

1. Initialize the permutation tables, S boxes, expansion tables, left shift schedules.
2. A block of 64 bits is permuted by an initial permutation called IP.
3. Resulting 64 bits are divided in two halves of 32 bits, left and right.
4. Right half goes through a function F (Feistel function)
5. Left half is XOR-ed with output from F function above.
6. Left and right are swapped(except last round).
7. If last round, apply an inverse permutation IP-1 on both halves and that's the output else, goto step 3.
8. Display the cipher text.

Feistel function F:

1. Expansion – 32 bits to 48 bits based on an expansion table.
2. Key mixing – round key combined with 48 bits from previous step by XOR operation.
3. Substitution – previous result divided into 8x6bits blocks before processed by s-boxes(substitution boxes)
4. Permutation based on a fixed permutation table.

Decryption Procedure for DES:

1. Use the cipher text as input.
2. Apply the same set of operations from step 2 to 7 of encryption procedure.
3. Use the keys K_i in reverse order (use K_{16} on the first iteration, K_{15} on the second until K_1 on last iteration).
4. Display the plain text.

Advanced Encryption Standard

Aim:

To implement Advanced Encryption Standard (AES) Algorithm.

1 Key Generation:

The Key interface represents keys for cryptographic operations. Keys are opaque containers that hold an encoded key, the key's encoding format, and its cryptographic algorithm.

SecretKey secretKey = new SecretKeySpec(keyBytes, "AES");

```
1. key = myKey.getBytes("UTF-8");
2.              System.out.println(key.length);
3.              sha = MessageDigest.getInstance("SHA-1");
4.              key = sha.digest(key);
5.           key = Arrays.copyOf(key, 16); // use only first 128 bit
6.           System.out.println(key.length);
```

To generate 128 bit keys

2 Encryption:

The doFinal() method to perform the encryption or decryption operation.

Get the cipher Instance

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

where it use AES in Electronic Code book and it uses publickey cryptography system padding.

SecretKey secretKey = new SecretKeySpec(keyBytes, "AES");

cipher.init(Cipher.ENCRYPT_MODE, secretKey);

cipher.doFinal(message)

3 Decryption:

ENCRYPT_MODE: initialize cipher object to encryption mode

DECRYPT_MODE: initialize cipher object to decryption mode

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

SecretKey secretKey = new SecretKeySpec(keyBytes, "AES");

```
 cipher.init(Cipher.DECRYPT_MODE, secretKey);

return cipher.doFinal(encryptedMessage);
```

SSN College of Engineering,
Department of Computer Science and Engineering
IT8761 Security Laboratory

Exercise 6:

To implement the Rivest-Shamir-Adleman (RSA) Algorithm.

Programming Language: Java

Hints:

Key Generation

1. Generate two large random primes, p and q, of approximately equal size such that their product n = pq.
2. Compute n = pq and (phi) φ = (p-1)(q-1).
3. Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1.
4. Compute the secret exponent d, 1 < d < phi, such that ed ≡ 1 (mod phi).
5. The public key is (n, e) and the private key (d, p, q).
6.

Encryption Procedure for RSA:

Sender does the following:-

1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer *m*, 1 < m <.
3. Computes the cipher text **$C = m^e \bmod n$.**
4. Display the cipher text C.

Decryption Procedure for RSA:

1. Use the cipher text as input.
2. Uses his private key (n, d) to compute **$m = C^d \bmod n$**.
3. Extracts the plaintext from the message representative *m*.
4. Display the plain text.

SSN College of Engineering,
Department of Computer Science and Engineering
CS IT8761 Security Laboratory

Exercise 7:

To implement the Diffie Helman Key exchange algorithm

Programming Language: Java

Hints:

1. Choose a prime number p and g is a primitive root modulo p.
2. Check for the primality of the number p (using Miller Rabin Method)
3. Read X_A, the secret key of A, such that $X_A < p$.
4. Compute the public key of A, $Y_A = g^{X_A} \mod p$
5. Read X_B, the secret key of B, such that $X_B < p$..
6. Compute the public key of B, $Y_B = g^{X_A} \mod p$
7. Compute A's shared secret key, $K = Y_B^{X_A} \mod p$
8. Compute B's shared secret key, $K = Y_A^{X_B} \mod p$
9. Display A and B's shred secret keys.

SSN College of Engineering,
Department of Computer Science and Engineering
IT 8761 Security Laboratory

Exercise 8:

To implement the message digest SHA1

Programming Language: Java

Hints:

1. Read the message
2. Divide the message into 512 bit blocks.
3. Append padding bits,
    - A single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message equals to 448 mod 512.
4. Append length
    - A 64-bit representation of the length of the message is appended
5. Initialize MD buffers, A, B, C, D,E
    - word A: 67452301
    - word B: efcdab89
    - word C: 98badcfe
    - word D: 10325476
    - word E: c3d2e1f0
6. Invoke the compress function for four times
7. Display the message digest from the buffers.

SHA1 Compression function:

1. (A,B,C,D,E) <- (E+f(t,B,C,D)+(A<< 5)+W_t+K t), A, (B<<30), C, D)

   - t is the step number
   - W_t is derived from the message block
   - K_t is a constant value derived from sin table.
     - *K(t) = 5A827999 (0 <= t <= 19)*
     - *K(t) = 6ED9EBA1 (20 <= t <= 39)*
     - *K(t) = 8F1BBCDC (40 <= t <= 59)*
     - *K(t) = CA62C1D6 (60 <= t <= 79)*
   - Each f(t), 0 <= t <= 79, operates on three 32-bit words B, C, D and produces a 32-bit word as output.
     - f(t;B,C,D) is defined as follows: for words B, C, D,
     - f(t;B,C,D) = (B *AND* C) OR ((*NO*T B) *AND* D) (0 <= t <= 19)
     - f(t;B,C,D) = B *XOR* C *XOR* D (20 <= t <= 39)
     - f(t;B,C,D) = (B *AND* C) *OR* (B *AND* D) *OR* (C *AND* D) (40 <= t <= 59)
     - f(t;B,C,D) = B *XOR* C *XOR* D (60 <= t <= 79).
   - <<<s circular left shift of 32 bit argument by s bits
   - + is addition modulo 2^{32}

2. Perform a 32 bit circular right shift such that;

   - a=e; b=a; c=b; d=c; e=d;

<u>For further clarification, read the associated pdf</u>

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E5A51	T[35] = 699D6122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57COFAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBC8	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F7537E82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

T table

SSN College of Engineering,
Department of Computer Science and Engineering
IT 8761 Security Laboratory

Exercise 9:

To implement the Signature Scheme - Digital Signature Standard

Programming Language: Java

Hints:

Module 1: Creating the digital signature

1. Create a KeyPairGenerator object

The **KeyPairGenerator** class provides **getInstance()** method which accepts a String variable representing the required key-generating algorithm and returns a KeyPairGenerator object that generates keys.

2. Initialize the KeyPairGenerator object

The **KeyPairGenerator** class provides a method named **initialize()** this method is used to initialize the key pair generator. This method accepts an integer value representing the key size.

3. Generate the KeyPairGenerator

Generate the **KeyPair** using the **generateKeyPair()** method.

4. Get the private key from the pair

Get the private key from the generated KeyPair object using the **getPrivate()** method.

5. Create a signature object

The **getInstance()** method of the **Signature** class accepts a string parameter representing required signature algorithm and returns the respective Signature object.

6. Initialize the Signature object

The **initSign()** method of the Signature class accepts a **PrivateKey** object and initializes the current Signature object.

7. Add data to the Signature object

The **update()** method of the Signature class accepts a byte array representing the data to be signed or verified and updates the current object with the data given.

8. Calculate the Signature

The **sign()** method of the **Signature** class returns the signature bytes of the updated data.

Module 2: Verifying Signature

1. Create a KeyPairGenerator object

The **KeyPairGenerator** class provides **getInstance()** method which accepts a String variable representing the required key-generating algorithm and returns a KeyPairGenerator object that generates keys.

2. Initialize the KeyPairGenerator object

The **KeyPairGenerator** class provides a method named **initialize()** method. This method is used to initialize the key pair generator. This method accepts an integer value representing the key size.

3. Generate the KeyPairGenerator

Generate the **KeyPair** using the **generateKeyPair()** method.

4. Get the private key from the pair

Get the private key from the generated KeyPair object using the **getPrivate()** method.

5. Create a signature object

The **getInstance()** method of the **Signature** class accepts a string parameter representing required signature algorithm and returns the respective Signature object.

6. Initialize the Signature object

The **initSign()** method of the Signature class accepts a **PrivateKey** object and initializes the current Signature object.

7. Add data to the Signature object

The **update()** method of the Signature class accepts a byte array representing the data to be signed or verified and updates the current object with the data given.

8. Calculate the Signature

The **sign()** method of the **Signature** class returns the signature bytes of the updated data.

9. Initialize the signature object for verification

To verify a Signature object you need to initialize it first using the **initVerify()** method it method accepts a **PublicKey** object.

10. Update the data to be verified

Update the initialized (for verification) object with the data the data to be verified

11. Verify the Signature

The **verify()** method of the Signature class accepts another signature object and verifies it with the current one. If a match occurs, it returns true else it returns false.

12. Check the Boolean output for whether sign is verified or not